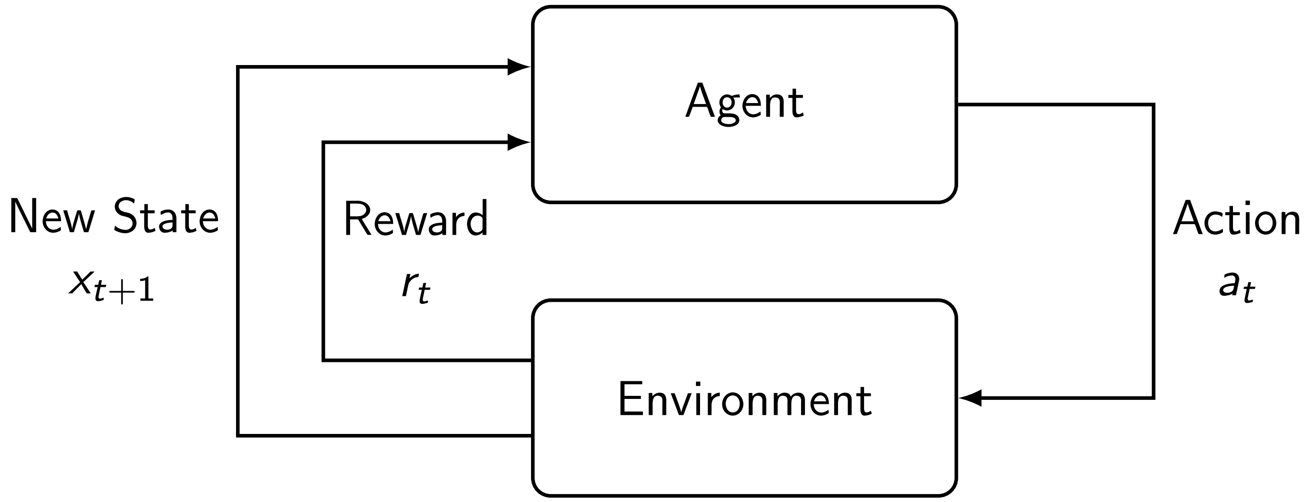
# Reinforcement Learning

Machine Learning (69152)

**Rubén Martínez Cantín**
Dpto. Informática e Ingeniería de Sistemas
Universidad de Zaragoza

1. Introduction
   - Markov Decision Processes (recap)
   - Learning by interaction
2. Prediction
   - Monte Carlo Evaluation
   - Temporal Differences
3. Control
   - Sampling control
   - Exploration vs Exploitation
   - Q-learning
   - Q-learning with features

# Markov Decision Processes

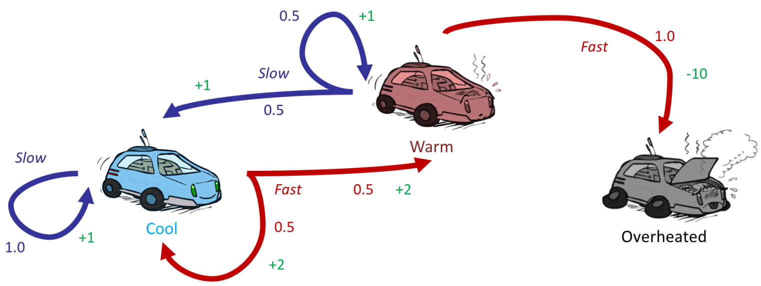



In particular, for Markov Decision Processes (MDPs):

- Markov property $x_{t+1} = f(x_t)$
- Full observability $y_t = x_{t+1}$
- Action $a_t \Leftarrow \pi(x_t)$

# Markov Decision Process

- A Markov Decision Process is a tuple $\langle \mathcal{X}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$
  - A set of states $x \in \mathcal{X}$.
  - A set of actions $a \in \mathcal{A}$.
  - A transition function $T(x, a, x') = p(x'|x, a)$
  - A reward function $R(x, a, x')$
  - Maybe a start state and a terminal state.




Credit: Dan Klein, Pieter Abbeel

# Markov Decision Process

- A Markov Decision Process is a tuple $\langle \mathcal{X}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$
  - A set of states $x \in \mathcal{X}$.
  - A set of actions $a \in \mathcal{A}$.
  - A transition function $T(x, a, x') = p(x'|x, a)$
  - A reward function $R(x, a, x')$
  - Maybe a start state and a terminal state.


- **Plot twist:** Now we don't know the transitions $T(\cdot)$ or rewards $R(\cdot)$
  - The agent must try actions to see the outcome.

Credit: Dan Klein, Pieter Abbeel

# Learning by interaction

Model-based reinforcement learning

- Learn an approximate model based on experiences
- Assume that the learned model is correct and solve it as if known MDP.

Model-free reinforcement learning

- Learn the policy and values directly from interaction
- No explicit model of the transitions $T(\cdot)$ or rewards $R(\cdot)$

# Example: Compute the expected age

We want to compute the expected age of all the students in the University.

### Known Model p(a)

$$\mathbb{E}(A) = \sum_a p(a) * a$$

If we $p(a)$ is unknown we can ask several students for the age $[a_1, a_2, \ldots]$

### Model based

$$\hat{p}(a) = \frac{num(a)}{N}$$

$$\mathbb{E}(A) \approx \sum_a \hat{p}(a) * a$$

### Model free

$$\mathbb{E}(A) \approx \frac{\sum_i a_i}{N}$$

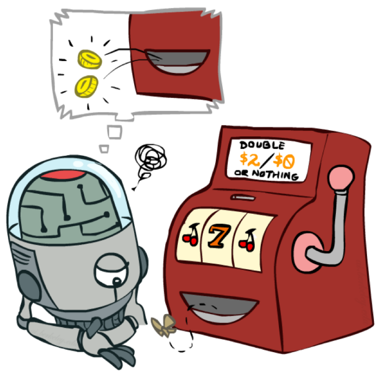
# Model based learning

- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct
- Step 1: Learn empirical MDP model
  - Count outcomes $x'$ for each $x$, $a$
  - Normalize to give an estimate of $\hat{p}(x'|x,a)$
  - Discover each $\hat{R}(x,a,x')$ when we experience $(x,a,x')$
- Step 2: Solve the learned MDP
  - For example, use value iteration, as before

Credit: Dan Klein, Pieter Abbeel

# Model-free learning

- Model-free Idea: why bother learning the transition or reward model?
  - Can we compute $V$, $Q$ or $\pi^*$ without a model?
  - Monte Carlo: Sample episodes, average rewards at the end.
  - Temporal differences: Use sampling to approximate the Bellman updates, compute new values during each learning step.




Credit: Dan Klein, Pieter Abbeel

# Model-free learning

- Passive reinforcement learning
  - Evaluation given a policy $\pi$, find the value $V^{\pi}$.
  - *Learner* has no choice. Just follow the policy
  - The agent is doing prediction.
- Active reinforcement learning
  - Find the optimal policy/values: $V^*, Q^*, \pi^*$.
  - *Learner* can choose. Exploration vs Exploitation.
  - The agent is doing control.

# Model-free learning

- Passive reinforcement learning
  - Evaluation given a policy $\pi$, find the value $V^{\pi}$.
  - *Learner* has no choice. Just follow the policy
  - The agent is doing prediction.
- Active reinforcement learning
  - Find the optimal policy/values: $V^*, Q^*, \pi^*$.
  - *Learner* can choose. Exploration vs Exploitation.
  - The agent is doing control.
- Remember: even if the learner can choose or not, it must take actions and interact with the world.

# Monte Carlo Direct Evaluation

- Monte Carlo (MC) passive reinforcement learning.
- Goal: learn $V^\pi$ from episodes of experience under policy $\pi$
- Act according to $\pi$.

$$x_1, a_1, R_2, x_2, \dots, x_n \sim \pi$$

- Recall that the utility is the total discounted reward:

$$U_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots R_n$$

- Recall that the value function is the expected return:

$$V^\pi(x) = \mathbb{E}_\pi[U_t | x_t = x]$$

- MC policy evaluation uses empirical mean return instead of expected return

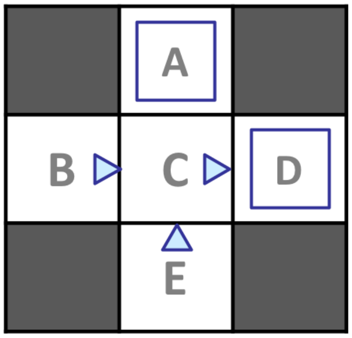
$$V^\pi(x) = \sum_i U_t^{(i)} \mathbb{1}_{x_t^{(i)} = x} \quad \forall i \in \text{episodes}$$

# Monte Carlo Direct Evaluation

- To evaluate state $x$
- Every time-step $t$ that state $x$ is visited in an episode,
- Increment counter $N(x) \leftarrow N(x) + 1$
- Increment total return $S(x) \leftarrow S(x) + U_t$
- Value is estimated by mean return $\hat{V}(x) = S(x)/N(x)$
- By law of large numbers, $\hat{V}(x) \rightarrow V^{\pi}(x)$ as $N(x) \rightarrow \infty$

# Monte Carlo Direct Evaluation

Input policy $\pi$




Assume: $\gamma = 1$

Observed Episodes

Episode 1

- B, east, C, -1
- C, east, D, -1
- D, exit, x, +10

Episode 2

- B, east, C, -1
- C, east, D, -1
- D, exit, x, +10

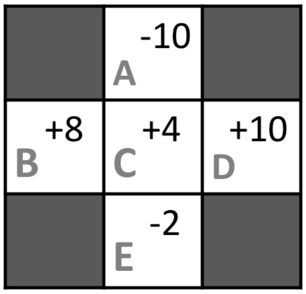
Episode 3

- E, north, C, -1
- C, east, D, -1
- D, exit, x, +10

Episode 4

- E, north, C, -1
- C, east, A, -1
- A, exit, x, -10

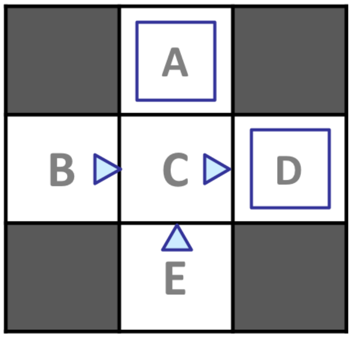
Estimated values




Credit: Dan Klein, Pieter Abbeel

# Monte Carlo Direct Evaluation

Input policy $\pi$




Assume: $\gamma = 1$

Observed Episodes

Episode 1

- B, east, C, -1
- C, east, D, -1
- D, exit, x, +10

Episode 2

- B, east, C, -1
- C, east, D, -1
- D, exit, x, +10

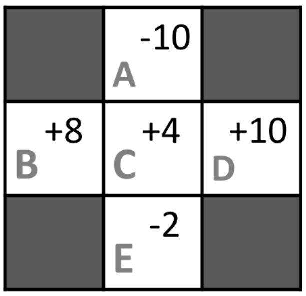
Episode 3

- E, north, C, -1
- C, east, D, -1
- D, exit, x, +10

Episode 4

- E, north, C, -1
- C, east, A, -1
- A, exit, x, -10

Estimated values




- How can B and E have different values if both go to C for this policy?

Credit: Dan Klein, Pieter Abbeel

# Incremental Monte Carlo

- Do we need to wait until the end of all episodes to compute the empirical mean?
- Remember the *expected age* example:
  - Model free approach: $\mathbb{E}(A) \approx \hat{A}_N = \frac{\sum_{i=1}^{N} a_i}{N}$
  - Incremental approach:

$$
\begin{aligned}
\hat{A}_N &\approx \frac{1}{N}\sum_{i=1}^{N} a_i \\
&= \frac{1}{N}\left(a_N + \sum_{i=1}^{N-1} a_i\right) \\
&= \frac{1}{N}\left(a_N + (N-1)\hat{A}_{N-1}\right) \\
&= \hat{A}_{N-1} + \frac{1}{N}\left(a_N - \hat{A}_{N-1}\right)
\end{aligned}
$$

# Incremental Monte Carlo for value function

- Update $V(x)$ incrementally after episode $x_1, a_1, R_2, x_2, \ldots, x_n$
- For each state $x_t$ with utility $U_t$

$$N(x_t) \leftarrow N(x_t) + 1$$
$$V(x_t) \leftarrow V(x_t) + \frac{1}{N(x_t)}(U_t - V(x_t))$$

- In non-stationary problems, it can be useful to track a running mean, that is, forget old episodes.

$$V(x_t) \leftarrow V(x_t) + \alpha(U_t - V(x_t))$$

# Problems with Monte Carlo Direct Evaluation

- Samples are based on full episodes.
- The problem needs to be episodic. Not for continuous.
- It ignores MDP structure.
- Is there a way to exploit MDP structure and use Bellman equations?

# Monte Carlo Policy Evaluation

- We had the Bellman expectation equation to update the value function:

$$V^{\pi}(x) = \sum_{a \in \mathcal{A}} \pi(a|x) \left( \sum_{x' \in \mathcal{X}} p(x'|x,a) \left( R(x,a,x') + \gamma V^{\pi}(x') \right) \right)$$

- This approach exploits the MDP structure and is valid for continuous problems. ☺
- But it requires $p(x'|x,a)$ and $R(x,a,x')$ to do it. ☹

# Sample based policy evaluation

- Instead of sampling full episodes, we can sample the value update.

$$V^\pi(x) = \sum_{a \in \mathcal{A}} \pi(a|x) \left( \sum_{x' \in \mathcal{X}} p(x'|x,a) \left( R(x,a,x') + \gamma V^\pi(x') \right) \right)$$

- From one state $x$, we sample just the next action $a_1, a_2, \ldots \sim \pi(x)$ and the next transition $x_1 \sim p(x'|a_1, x), x_2 \sim p(x'|a_2, x), \ldots$.
- Then, we collect the outcome for the action/transition:

$$\begin{aligned} sample_1 &= R(x, a_1, x_1') + \gamma V_k^\pi(x_1') \\ sample_2 &= R(x, a_2, x_2') + \gamma V_k^\pi(x_2') \\ &\ldots \end{aligned}$$

- Finally,

$$V_{k+1}^\pi(x) \leftarrow \frac{1}{N} \sum_{i=1}^{N} sample_i$$

# Temporal Difference

- Idea: learn from any interaction.
- Combine the sample based policy evaluation with the incremental Monte Carlo method.
- Remember: Incremental Monte Carlo updates the value with the *actual* utility $U_t$.

$$V(x_t) \leftarrow V(x_t) + \alpha(U_t - V(x_t))$$

- Temporal Difference updates the value with the *estimated* utility $R_{t+1} + \gamma V(x_{t+1})$

$$V(x_t) \leftarrow V(x_t) + \alpha(R_{t+1} + \gamma V(x_{t+1}) - V(x_t))$$

# Temporal Difference

- Temporal Difference (TD) can be learn after any interaction. No need to wait for the end of the episode.
- It can be used for continuous problems.
- It can even learn from incomplete sequences.
- It exploits MDP structure.

$$
\begin{aligned}
\mathit{sample}_k &= R(x, a, x') + \gamma V_k^{\pi}(x') \\
V_{k+1}^{\pi}(x) &\leftarrow (1-\alpha) V_k^{\pi}(x) + \alpha \cdot \mathit{sample}_k = \\
&= V_k^{\pi}(x) + \alpha(\mathit{sample}_k - V_k^{\pi}(x))
\end{aligned}
$$

- Remember that $\alpha \in [0, 1]$ is used to *forget*.

  $\gamma \rightarrow 0$ ignores the distant future.

  $\alpha \rightarrow 1$ forgets the distant past

# Forgetting the past

- Remember, we can use a running mean to forget old updates.
  - In MC, for non-stationary problems
  - In TD, recursive update of Bellman equation:

$$V(x_t) \leftarrow V(x_t) + \alpha(update_t - V(x_t))$$

- A running average $\hat{z}$ of $\{z\}_{i=1}^{N}$ elements:

$$\hat{z}_k = \hat{z}_{k-1} + \alpha(z_k - \hat{z}_{k-1}) = (1-\alpha)\hat{z}_{k-1} + \alpha z_k$$

- Weights recent samples more:

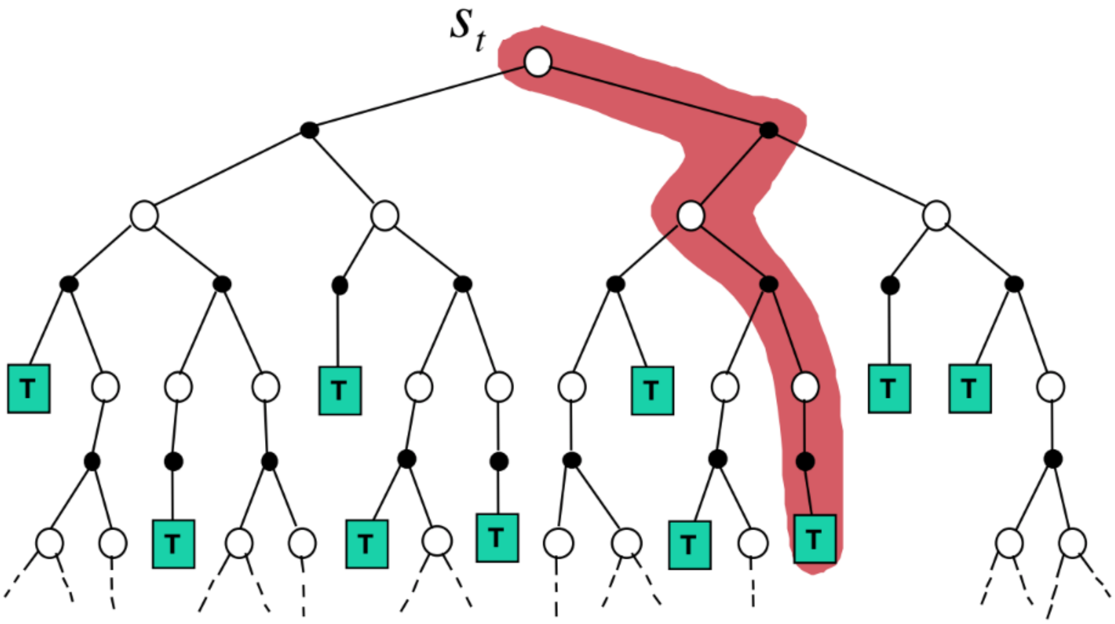
$$\hat{z}_k = \frac{z_k + (1-\alpha)z_{k-1} + (1-\alpha)^2 z_{k-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

# MC vs TD

- Both methods converge to the true value if experience $\to \infty$
- MC requires full episodes.
- MC is a very simple and general idea. It can be applied for any system.
- TD exploits the Markov property.
  - It is more efficient for Markov environments.
  - It might not converge for certain environments (e.g.: non-Markov).

# Monte Carlo Update

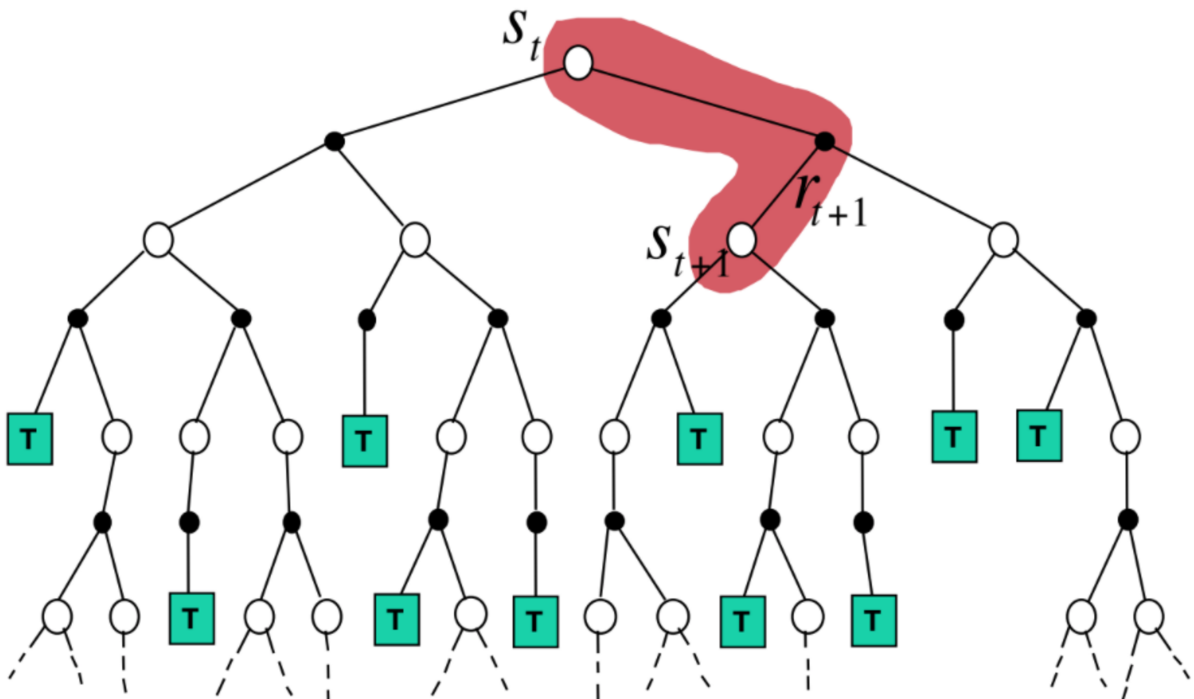
$$V(x_t) \leftarrow V(x_t) + \alpha(U_t - V(x_t))$$




Credit: David Silver

# Temporal Difference Update

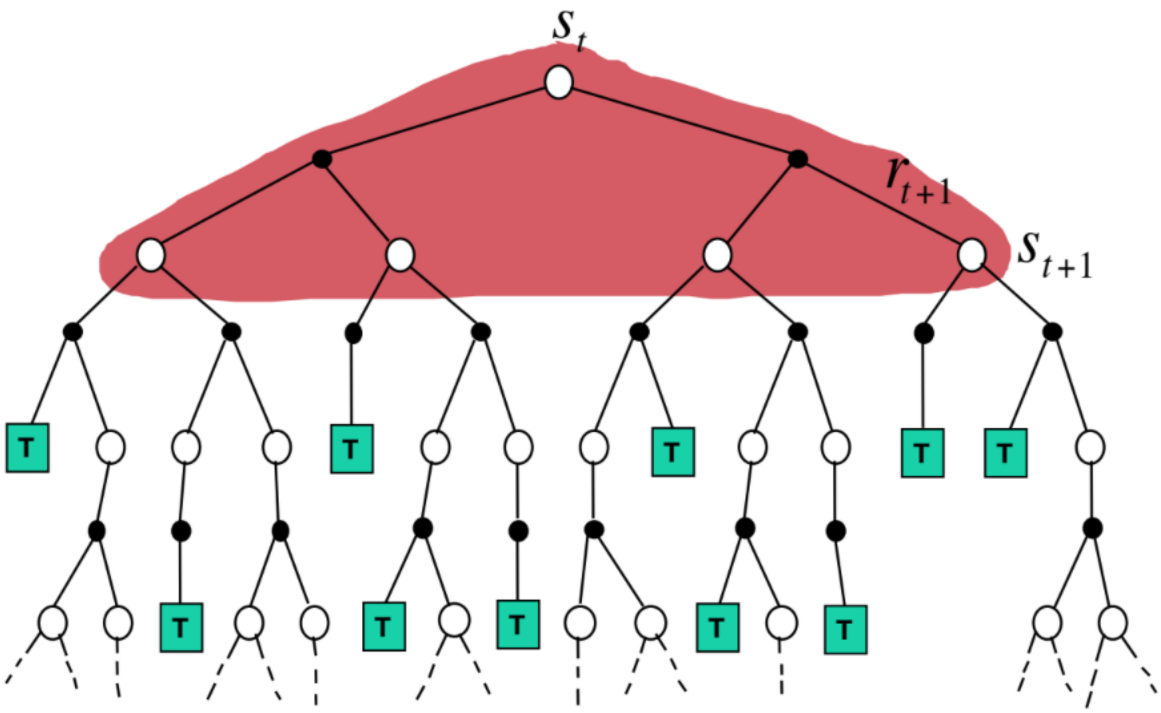
$$V(x_t) \leftarrow V(x_t) + \alpha(R_{t+1} + \gamma V(x_{t+1}) - V(x_t))$$




Credit: David Silver

# Dynamic Programming Update

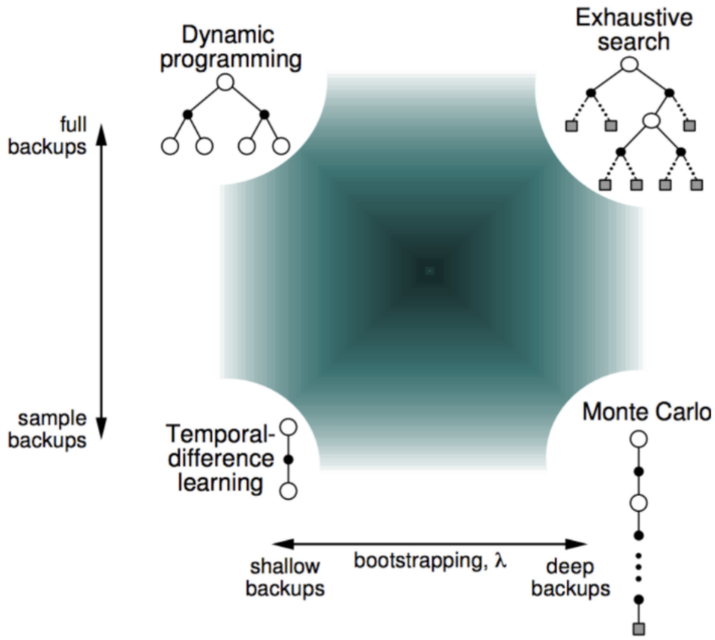
$$V(x_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(x_{t+1})\right]$$




Credit: David Silver

# Unified view of RL

- The TD algorithm that we have seen is called TD(0), because it considers only one step ahead.
- The generalization is called TD($\lambda$), which combines multiple steps ahead.




Credit: David Silver

# Sampling control

- We have seen how to use MC and TD for prediction (policy evaluation).
- We are going to use MC and TD for control (optimal policy/value).

- On-policy learning
  - *Learn on the job*
  - Learn about policy $\pi$ from experience sampled from $\pi$
- Off-policy learning
  - *Look over someone's shoulder*
  - Learn about policy $\pi$ from experience sampled from $\mu$

# Quick Parenthesis: Q-values

- Computing the greedy policy from $V(x)$ requires the MDP model $p(x'|x,a)$ and $R(x,a,x')$.

$$\pi'(x) = \arg\max_a \sum_{x'} p(x'|x,a)\left(R(x,a,x') + \gamma V(x')\right)$$

- Computing the greedy policy from $Q(x,a)$ can be done model-free:

$$\pi'(x) = \arg\max_a Q(x,a)$$

- Solution: we are going to do control based on $Q$-functions.

# SARSA

- Iteratively estimate $Q$ and $\pi$ (similar to policy iteration)
- Policy evaluation (estimate $Q$ given $\pi$):

  Good: Monte Carlo: run multiple full episodes, then update policy.

  Better: Improve policy after each episode.

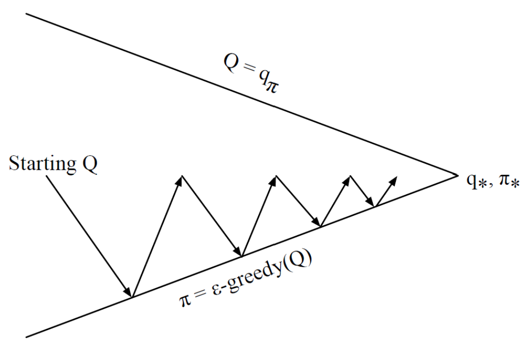
  Best: Do not run full episodes. Use Temporal Differences.

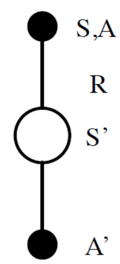
$$Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \alpha(R + \gamma Q(x_{t+1}, a_{t+1}) - Q(x_t, a_t))$$

- Policy improvement (improve $\pi$ given $Q$):

  Exploit: The optimal policy is greedy to the value function...

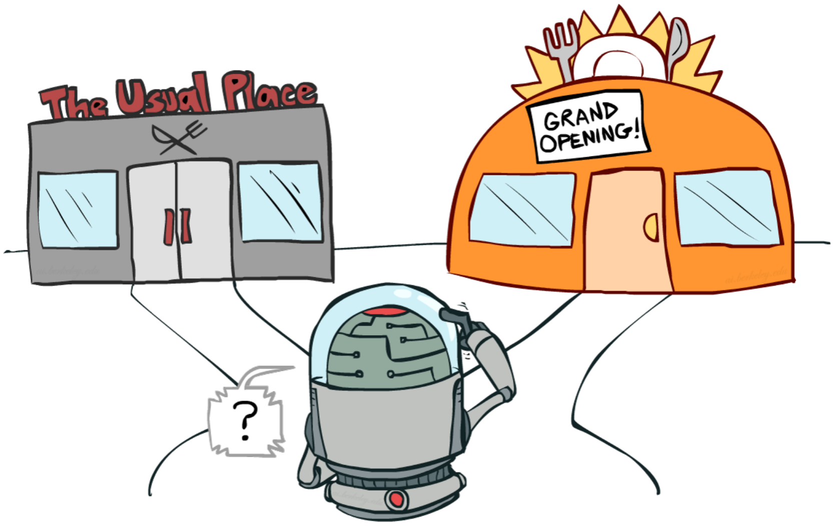
  Explore: ... but remember that we also need to explore!



# Exploration vs Exploitation

- The greedy policy is the best option if we know the model.
- Now, we are learning by experience.
  - The agent must have diverse experiences to learn.
  - But the agent must also find an optimal behavior.



# Regret

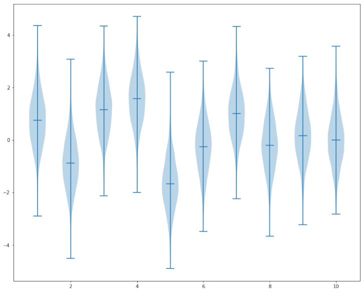
- Regret measures your mistakes. The (expected) reward of your actions, including suboptimal choices, against the (expected) optimal reward.
- Minimizing regret is not only learning to be optimal. It is optimally learning to be optimal.
- Pure random exploration will always find the optimal policy, but it has very high regret.
- A good trade-off of exploration and exploitation can provide optimal regret.

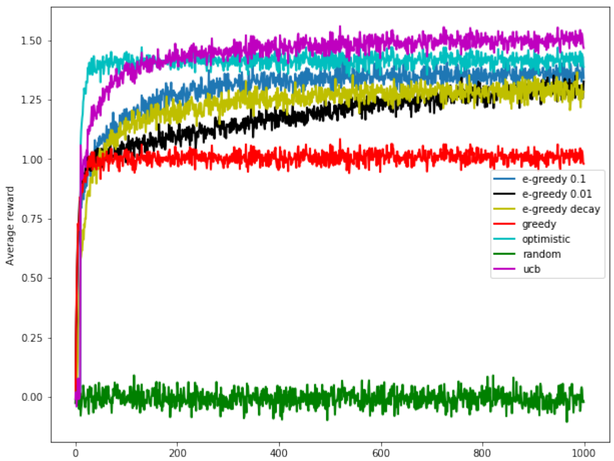
# Exploration-exploitation trade-off principles

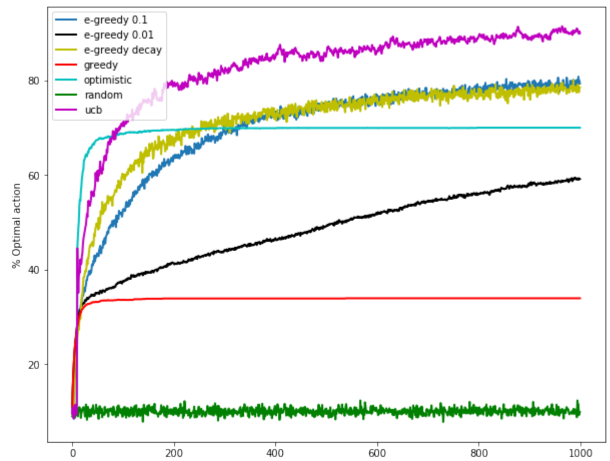
- The limit cases are the *greedy policy* (always exploit) and the *random policy* (always exploit).
  - We have seen that both policies have high regret.
- Random mixing ($\epsilon$-greedy)
  - The simplest approach: flip a coin and choose *greedy* or *random policy* depending on the outcome.
- Optimistic initialization
  - Assume *unknown* = *best*. Try everything at least once.
- Optimism in the Face of Uncertainty
  - In case of uncertainty, assume the best possible outcome.
  - For example, upper confidence bound
  - If you are right: you win!
  - If you are wrong: you learn a lot!

# Exploration-exploitation demo

**Exercise/demo**: `https://drive.google.com/file/d/1xAph2c1P8pPcJudiWBCd9Rrqt4p2EORa/view?usp=sharing`





# DP vs TD algorithms

| | Full backup | TD Backup |
|---|---|---|
| $V^{\pi}(x), Q^{\pi}(x,a)$ | Policy evaluation | TD learning |
| $\pi^*(x,a), V^*(x), Q^*(x,a)$ | Policy iteration | SARSA |
| $V^*(x), Q^*(x,a)$ | Value iteration | ??? |

# DP vs TD algorithms

| | Full backup | TD Backup |
|---|---|---|
| $V^{\pi}(x), Q^{\pi}(x,a)$ | Policy evaluation | TD learning |
| $\pi^{*}(x,a), V^{*}(x), Q^{*}(x,a)$ | Policy iteration | SARSA |
| $V^{*}(x), Q^{*}(x,a)$ | Value iteration | Q-learning |

# Q-learning

- Start with Q-value iteration

$$Q_{k+1}(x_t, a_t) = \sum_{x_{t+1}} p(x_{t+1}|x_t, a_t) \left( R(x_t, a_t, x_{t+1}) + \gamma \max_{a'} Q_k(x_{t+1}, a') \right)$$

- Consider the sample update

$$sample = R(x_t, a_t, x_{t+1}) + \gamma \max_{a'} Q_k(x_{t+1}, a')$$

- Which, using the running average, can be incorporated as:

$$Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \alpha(R_{t+1} + \gamma \max_{a'} Q(x_{t+1}, a') - Q(x_t, a_t))$$

# Q-learning algorithm

**Algorithm 1** Q-learning algorithm

**Input:** Step size $\alpha \in [0, 1]$, policy parameters: e.g., small $\epsilon > 0$.
  Initialize $Q(x, a)$, for all $x \in \mathcal{X}, a \in \mathcal{A}$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$
  **for** each episode **do**:
    Initialize $x_0$
    **for** each step $t$ in episode **do**:
      Choose $a_t$ from $x_t$ using a policy derived from $Q$ (e.g.: $\epsilon$-greedy)
      Take action $a_t$, observe $R_{t+1}$ and $x_{t+1}$.
      $Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \alpha(R_{t+1} + \gamma \max_{a'} Q(x_{t+1}, a') - Q(x_t, a_t))$
      **if** $x_{t+1}$ is terminal **then** Stop episode

# On-policy vs Off-policy

- SARSA and Q-learning require 2 actions for each update.

$$Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \alpha(R_{t+1} + \gamma Q(x_{t+1}, a') - Q(x_t, a_t))$$

- This may come from different sources:
  - The behavior policy is how the agent is acting.
  - The target policy is the policy that the agent is learning.
- On-policy vs off-policy
  - On-policy methods use the same policy for the behavior and target.
  - Off-policy methods use a different policy. They can learn the optimal policy even acting suboptimally!

# On-policy vs Off-policy

- In SARSA, all actions are selected according to the target policy $\pi$, for example: $\epsilon$-greedy.

$$Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \alpha(R_{t+1} + \gamma Q(x_{t+1}, a_{t+1}) - Q(x_t, a_t))$$

- In Q-learning, actions are selected according to the behavior policy $\mu$, for example: $\epsilon$-greedy, but the agent assumes that in the future it will be optimal ($\pi$ is the greedy policy).

$$Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \alpha(R_{t+1} + \gamma \max_{a'} Q(x_{t+1}, a') - Q(x_t, a_t))$$

# Scaling Q-learning

- How many states does smallGrid Pacman have?

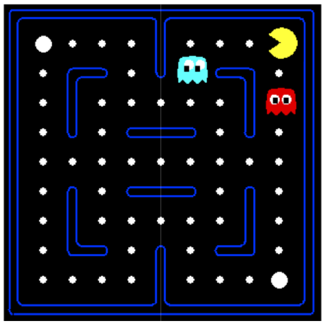
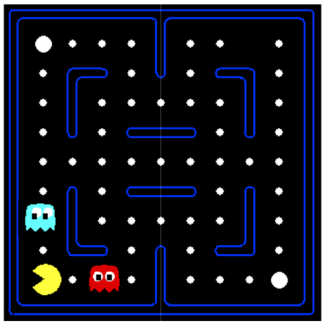
# Scaling Q-learning

- How many states does smallGrid Pacman have?


- Classic Pacman has: $240^5 \cdot 2^{240} = 1.766847110^{72}$ states
- Basic Q-learning keeps a table with a Q-value for every state and action.
- This idea is not able to scale:
  - During training, it is impossible to visit all the states/actions.
  - The table cannot be kept in memory.

# Generalization Q-learning

- We want to learn from few states and generalize to similar states.

| We learn that this is a *bad state* | Basic Q-learning tell us nothing about this state. | Or even this state! |
|---|---|---|
|  |  |  |

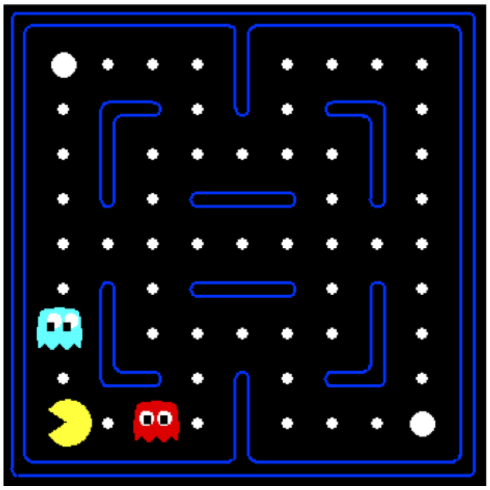
- What do they have in common?

Credit: Dan Klein, Pieter Abbeel

# Approximate Q-learning

- Idea: exploit certain properties (features) of state.
- A feature is a function from states or (state,action) to numbers.
- We can use features to describe other functions

- Examples of features:
  - Distance to a ghost.
  - Distance to a dot.
  - Number of ghost.
  - Is Pacman in a corner/tunnel?
  - $1/(distance\_to\_dot)^2$
  - ...
  - Is it the state in this slide?
  - ...
  - Does this action get me closer to food?



Credit: Dan Klein, Pieter Abbeel

# Linear function approximation

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(x) = w_1 f_1(x) + w_2 f_2(x) + \ldots + w_n f_n(x)$$
$$Q(x, a) = w'_1 f_1(x, a) + w'_2 f_2(x, a) + \ldots + w'_n f_n(x, a)$$

- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but actually be very different in value!

# Approximate Q-learning

- Take a linear Q-value function:

$$Q(x, a) = w_1 f_1(x, a) + w_2 f_2(x, a) + \ldots + w_n f_n(x, a)$$

- Q-learning update is based on estimation error:

$$\textit{error} = \left( R_{t+1} + \gamma \max_{a'} Q(x_{t+1}, a') \right) - Q(x, a)$$

- Basic Q-learning updates each value with the error:

$$Q(x, a) \leftarrow Q(x, a) + \alpha \cdot \textit{error}$$

- Approximate Q-learning updates the weights of active features:

$$w_i \leftarrow w_i + \alpha \cdot \textit{error} \cdot f_i(x, a)$$

# Approximate Q-learning

- Approximate Q-learning updates the weights of active features:

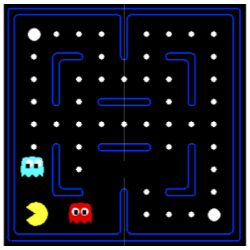
$$w_i \leftarrow w_i + \alpha \cdot \textcolor{red}{error} \cdot f_i(x, a)$$

- Intuitive interpretation:
  - If something *unexpectedly* bad happens, penalize the features that were on $\Rightarrow w_i \downarrow$.
  - If something *unexpectedly* good happens, reward the features that were on $\Rightarrow w_i \uparrow$.
- Formal explanation: This is *online least squares with gradient descent*.
  - If error is Gaussian, then least squares = maximum likelihood.

# Example: Q-learning in Pacman



- Initial value: $Q(x, a) = 4.0 f_{DOT}(x, a) - 1.0 f_{GST}(x, a)$
- Features: $f_i(x, a) = \frac{1}{dist\_pacman\_i}$ $\qquad \alpha = 0.004$



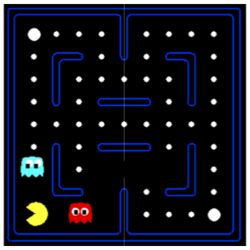
$f_{DOT}(x, N) = 0.5 \quad f_{GST}(x, N) = 1$

$$Q(x_t, N) = 1$$

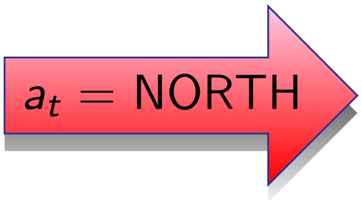
# Example: Q-learning in Pacman

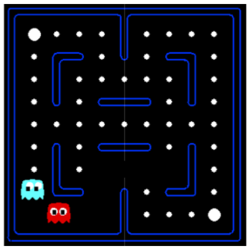


- Initial value: $Q(x, a) = 4.0 f_{DOT}(x, a) - 1.0 f_{GST}(x, a)$
- Features: $f_i(x, a) = \frac{1}{dist\_pacman\_i}$ $\qquad \alpha = 0.004$



$R = -500$



$f_{DOT}(x, N) = 0.5 \quad f_{GST}(x, N) = 1$

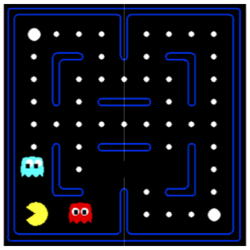
$Q(x_t, N) = 1$

$Q(x_{t+1}, \cdot) = 0$

# Example: Q-learning in Pacman

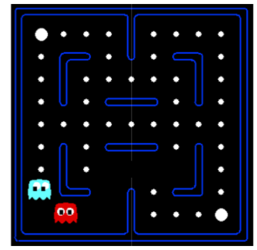


- Initial value: $Q(x,a) = 4.0 f_{DOT}(x,a) - 1.0 f_{GST}(x,a)$
- Features: $f_i(x,a) = \frac{1}{dist\_pacman\_i}$ $\qquad \alpha = 0.004$



$R = -500$

$a_t = \text{NORTH}$



$f_{DOT}(x,N) = 0.5 \quad f_{GST}(x,N) = 1$

$Q(x_t,N) = 1$

$Q(x_{t+1},\cdot) = 0$

$$error = (R + \gamma \max_{a'} Q(x_{t+1},a')) - Q(x_t,a_t) = -501$$

$$w_{DOT} \leftarrow 4.0 + \alpha(-501)0.5 \qquad w_{GST} \leftarrow -1.0 + \alpha(-501)1.0$$

- Final value: $\boxed{Q(x,a) = 3.0 f_{DOT}(x,a) - 3.0 f_{GST}(x,a)}$

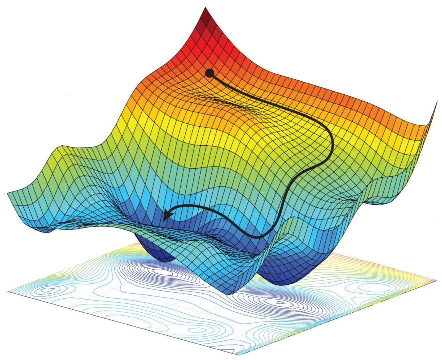
# Online least squares

- Remember gradient descent for logistic regression (see Fundamentals).
- Using *online least squares* we can minimize the error, one point $(x, y)_p$ at a time, by following the gradient $w_m = w_m + \frac{1}{2}\alpha \frac{\partial J_p(w)}{\partial w_m}$.
  - We have the same problems here: learning rate $\alpha$, overfitting, etc.

$$J_p(w) = \left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial J_p(x)}{\partial w_m} = -2\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x)\right) f_m(x)$$



Credit: Alexander Amini, Daniela Rus

# Bibliography


- Richard S. Sutton, Andrew G. Barto. Reinforcement learning: An Introduction (second edition), 2018
  http://incompleteideas.net/book/the-book.html
- David Silver. Advanced Topics in Machine Learning, 2015
  http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
- Dan Klein, Pieter Abbeel. Artificial Intelligence CS188
  http://ai.berkeley.edu