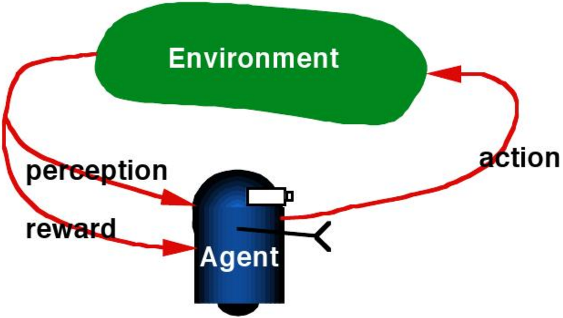
L3 Apprentissage

Michèle Sebag – Benjamin Monmège
LRI – LSV

22 mai 2013

Overview

Reinforcement Learning



Generalities

- An agent, spatially and temporally situated
- Stochastic and uncertain environment
- Goal: select an action in each time step,
- ... in order maximize expected cumulative reward over a time horizon

What is learned ?

$$\text{A policy} = \text{strategy} = \{ \text{state} \mapsto \text{action} \}$$

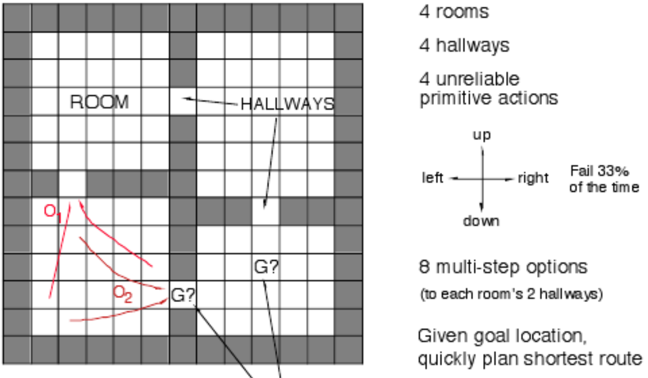
Reinforcement Learning

Context

An unknown world.

Some actions, in some states, bear rewards with some delay [with some probability]

Goal : **find policy (state $\rightarrow$ action) maximizing the expected reward**



Reinforcement Learning, example

World You are in state 34.
Your immediate reward is 3. You have 3 actions

Robot I'll take action 2

World You are in state 77
Your immediate reward is -7. You have 2 actions

Robot I'll take action 1

World You are in state 34 (again)

Markov Decision Property: actions/rewards only depend on the current state.

Reinforcement Learning

Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will – others things being equal – be more firmly connected with the situation, so that when it recurs, they will more likely to recur;

those which are accompanied or closely followed by discomfort to the animal will – others things being equal – have their connection with the situation weakened, so that when it recurs, they will less likely to recur;

the greater the satisfaction or discomfort, the greater the strengthening or weakening of the link.
Thorndike, 1911.

Formal background

Notations

- State space $\mathcal{S}$
- Action space $\mathcal{A}$
- Transition model $p(s, a, s') \mapsto [0, 1]$
- Reward $r(s)$

Goal

- Find policy $\pi : \mathcal{S} \mapsto \mathcal{A}$

$$\text{Maximize } E[\pi] = \text{ Expected cumulative reward}$$

(detail later)

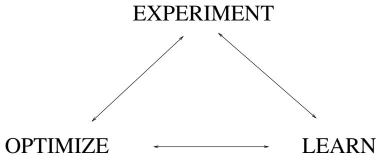
Applications

- Robotics
  Navigation, football, walk,
- Games
  Backgammon, Othello, Tetris, Go, ...
- Control
  Helicopter, elevators, telecom, smart grids, manufacturing, ...
- Operation research
  Transport, scheduling, ...
- Other Computer Human Interfaces, ...

Position of the problem

3 interleaved tasks

- Learn a world model (p, r)
- Decide/select (the best) action
- Explore the world



Sources

- Sutton & Barto, Reinforcement Learning, MIT Press, 1998
- `http://www.eecs.umich.edu/~baveja/NIPS05RLTutorial/`

Particular case

If the transition model is known

Reinforcement learning $\rightarrow$ Optimal control

What's hard

Curse of dimensionality

- State: features *size*, *texture*, *color*, … …
  $|\mathcal{S}|$ exponential wrt number of features
- Not all features are always relevant

Example:

see	swann	white	—
	swann	black	take a video
	bear	—	flee

What's hard

Curse of dimensionality

- State: features *size*, *texture*, *color*, … …
  $|\mathcal{S}|$ exponential wrt number of features
- Not all features are always relevant

Example:

see	swann	white	–
	swann	black	take a video
	bear	–	flee

Time horizon – Bounded rationality

- T.h. is infinite: eternity. NEVER
- Finite, unknown: reach the goal asap
- Finite: reach the goal in T time steps
- Bounded rationality: find as fast as possible a decent policy (finding an approximation of the goal).

Overview

Formalisation

Notations

- State space $\mathcal{S}$
- Action space $\mathcal{A}$
- Transition model
  - deterministic: $s' = t(s, a)$
  - probabilistic: $P^a_{s,s'} = p(s, a, s') \in [0, 1]$.
- Reward $r(s)$ **bounded**
- Time horizon H (finite or infinite)

Goal

- Find policy (strategy) $\pi : \mathcal{S} \mapsto \mathcal{A}$
- which maximizes (discounted) cumulative reward from now to timestep H

$$\sum_t r(s_t)$$

Formalisation

Notations

- State space $\mathcal{S}$
- Action space $\mathcal{A}$
- Transition model
  - deterministic: $s' = t(s, a)$
  - probabilistic: $P^a_{s,s'} = p(s, a, s') \in [0, 1]$.
- Reward $r(s)$ **bounded**
- Time horizon H (finite or infinite)

Goal

- Find policy (strategy) $\pi : \mathcal{S} \mapsto \mathcal{A}$
- which maximizes (discounted) cumulative reward from now to timestep H

$$\sum_{t=1}^{H} \gamma^t r(s_t) \qquad \gamma < 1$$

Formalisation

Notations

- State space $\mathcal{S}$
- Action space $\mathcal{A}$
- Transition model
  - deterministic: $s' = t(s, a)$
  - probabilistic: $P^a_{s,s'} = p(s, a, s') \in [0, 1]$.
- Reward $r(s)$ **bounded**
- Time horizon H (finite or infinite)

Goal

- Find policy (strategy) $\pi : \mathcal{S} \mapsto \mathcal{A}$
- which maximizes (discounted) cumulative reward from now to timestep H

$$\mathbb{E}_{s_0,\pi}[\sum_{t=1}^{\infty} \gamma^t r(s_t)]$$

Markov Decision Process

But can we define $P^a_{ss'}$ and $r(s)$?

- YES, if all necessary information is in s
- NO, otherwise
  - If state is partially observable

Goal: arrive in the third branch

  - If environment (reward and transition distribution) is changing
    Reward for *first* photo of an object by the satellite

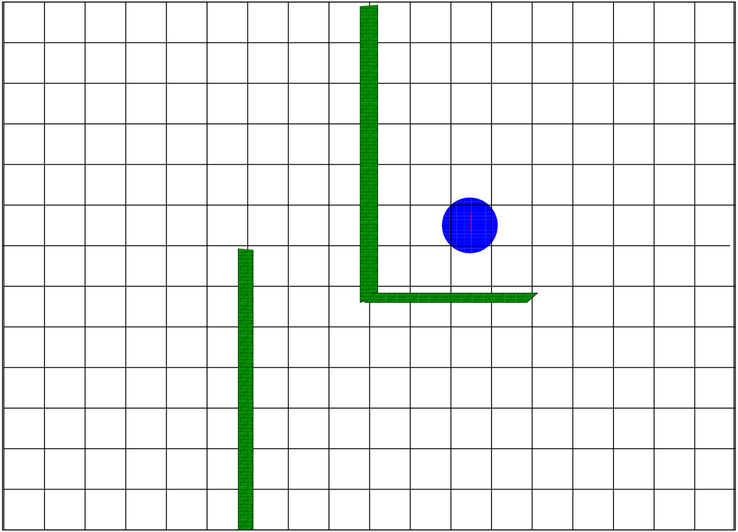
The Markov assumption

$$P(s_{h+1} | s_0 \; a_0 \; s_1 \; a_1 \ldots s_h \; a_h) = P(s_{h+1} | s_h \; a_h)$$

Everything you need to know is the current (state, action).

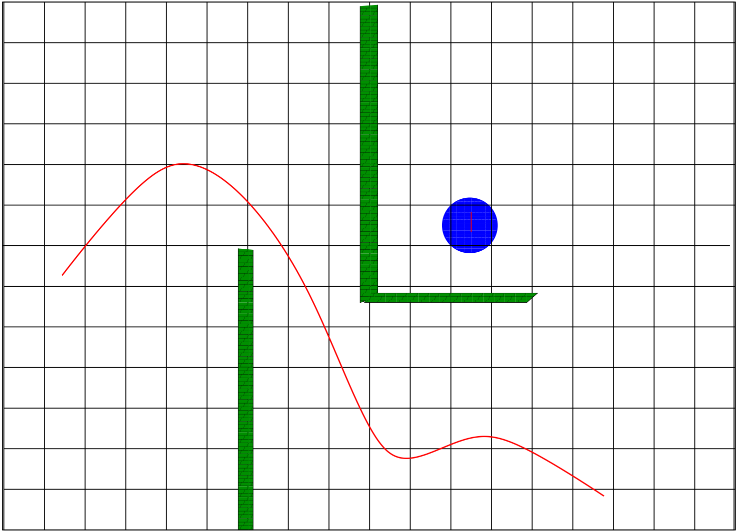
Find the treasure

Single reward: on the treasure.

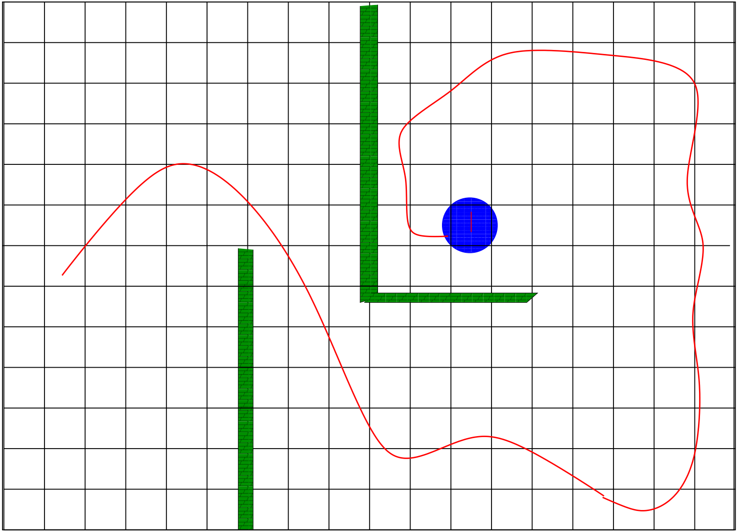


Wandering robot

Nothing happens...

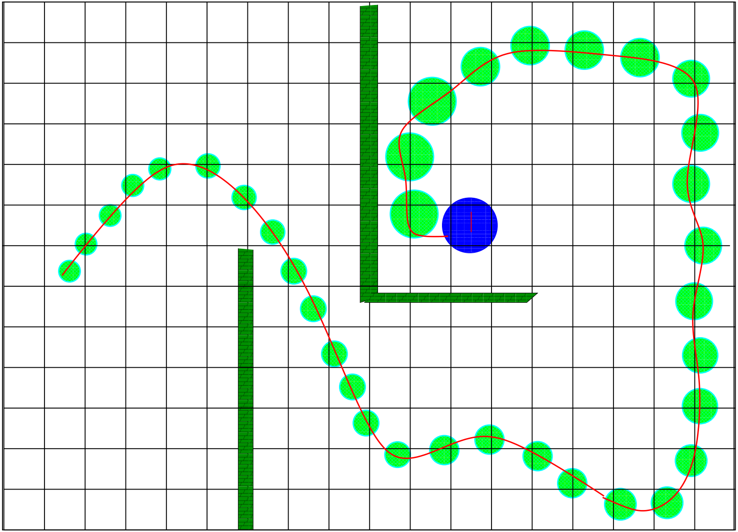


The robot finds it



Robot updates its value function

$V(s,a) ==$ "distance" to the treasure *on the trajectory*.

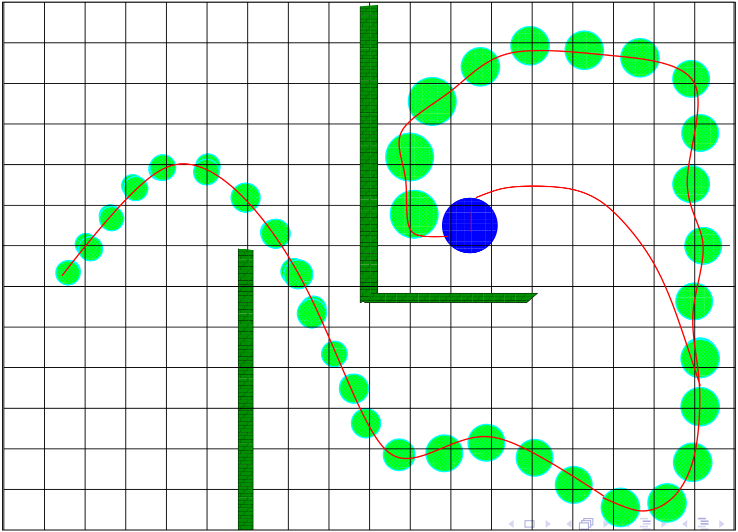


Reinforcement learning

* Robot most often selects $a = \arg\max V(s, a)$
* and sometimes explores (selects another action).

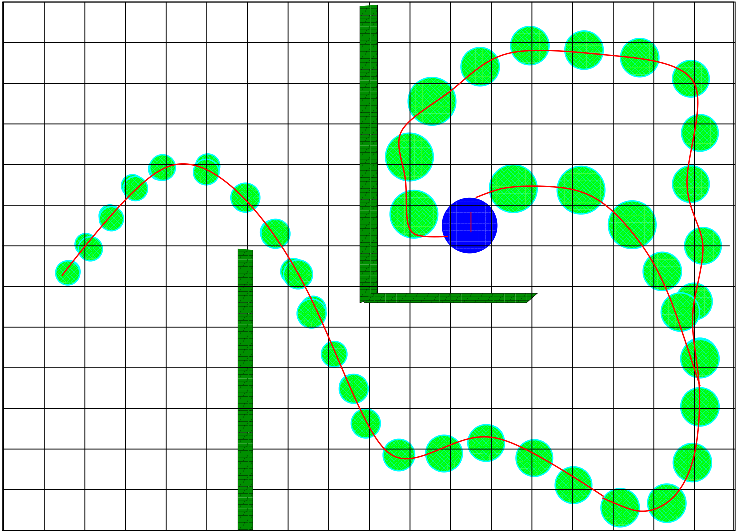
Reinforcement learning

* Robot most often selects $a = \arg\max V(s, a)$
* and sometimes explores (selects another action).
* Lucky exploration: finds the treasure again



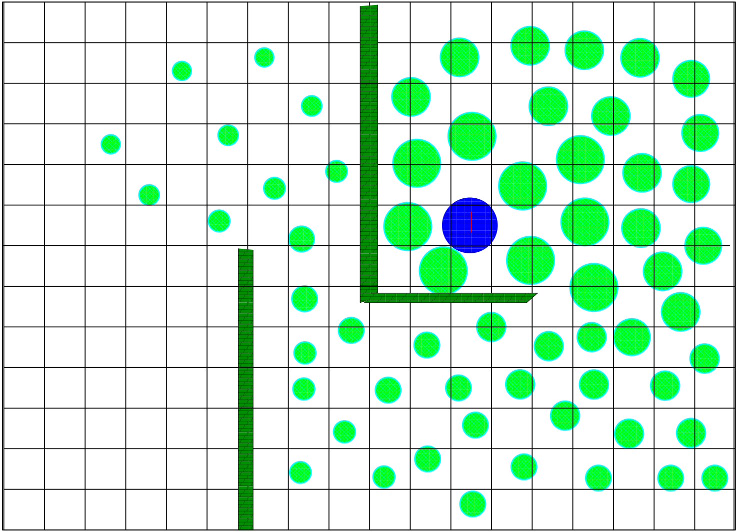
Updates the value function

* Value function tells how far you are from the treasure *given the known trajectories*.



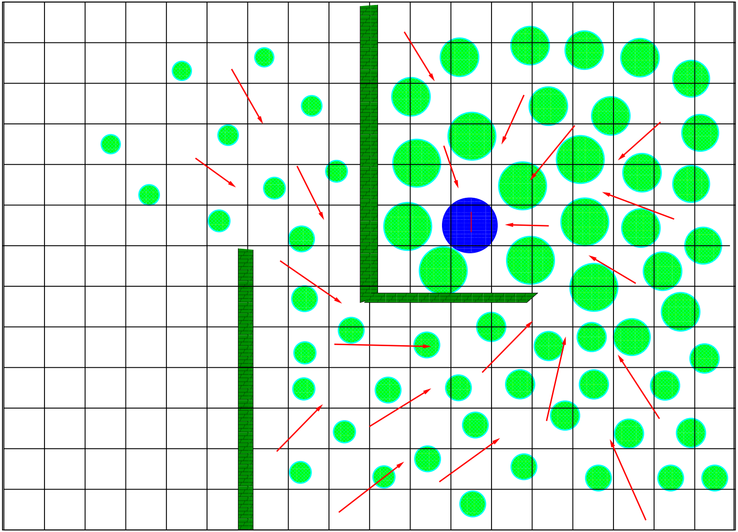
Finally

* Value function tells how far you are from the treasure



Finally

Let's be greedy: selects the action maximizing the value function



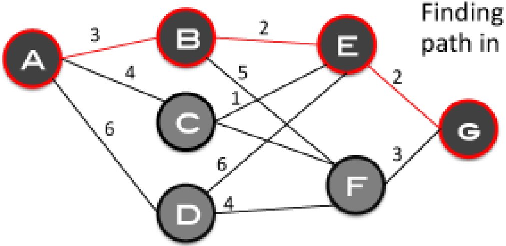
Underlying: Dynamic programming

Principle

- Recursively decompose the problem in subproblems
- Solve and propagate

An example

$$\ell(\text{shortest path } (A,B)) < \ell(sp(A,C)) + \ell(sp(C,B))$$



Finding the shortest path in a graph

Approaches

- Value function
  - Value iteration
  - Policy iteration
- Temporal differences
- Q-learning
- Direct policy search
  optimization in the π space — Stochastic optimization

Policy and value function 1/3

Finite horizon, deterministic transition

$$V_\pi(s_0) = r(s_0) + \sum_{h=1}^{H} r(s_h)$$

where $s_{h+1} = t(s_h, a_h = \pi(s_h))$

Policy and value function 1/3

Finite horizon, deterministic transition

$$V_\pi(s_0) = r(s_0) + \sum_{h=1}^{H} r(s_h)$$

where $s_{h+1} = t(s_h, a_h = \pi(s_h))$

Finite horizon, stochastic transition

$$V_\pi(s_0) = r(s_0) + \sum_{h=1}^{H} \mathbf{p(s_{h-1}, a_{h-1} = \pi(s_{h-1}), s_h)} r(s_h)$$

where $s_{h+1} = s$ with proba $p(s_h, a_h = \pi(s_h), s)$

Policy and value function, 2/3

Finite horizon, stochastic transition

$$V_\pi(s_0) = r(s_0) + \sum_{h=1}^{H} \mathbf{p}(\mathbf{s_{h-1}}, \mathbf{a_{h-1}} = \pi(\mathbf{s_{h-1}}), \mathbf{s_h}) r(s_h)$$

where $s_{h+1} = s$ with proba $p(s_h, a_h = \pi(s_h), s)$

Infinite horizon, stochastic transition

$$V_\pi(s_0) = r(s_0) + \sum_{h=1}^{H} \gamma^{\mathbf{h}} \mathbf{p}(\mathbf{s_{h-1}}, \mathbf{a_{h-1}} = \pi(\mathbf{s_{h-1}}), \mathbf{s_h}) r(s_h)$$

with discount factor γ, $0 < \gamma < 1$

Remark

$\gamma < 1 \rightarrow V < \infty$

γ small $\rightarrow$ myopic agent.

Value function and Q-value function

Value function

$$V : S \mapsto \mathbb{R}$$

$V_\pi(s)$: utility of state s when following policy π

Improving π by using V_π requires to know the transition model:

$$\pi(s) \rightarrow \text{ arg max } P^a_{ss'} V_\pi(s')$$

Q function

$$Q : (S \times A) \mapsto \mathbb{R}$$

$Q_\pi(s, a)$: utility of selecting action a in state s when following policy π

Improving π by using Q_π is straightforward:

$$\pi(s) \rightarrow \text{ arg max } Q_\pi(s, a)$$

Optimal policies

From value function to a better policy

$$\pi(s) = argmax_a\{P^a_{ss'}V_\pi(s')\}$$

From policies to optimal value function

$$V^*(s) = max_\pi V_\pi(s)$$

From value function to optimal policy

$$\pi^*(s) = argmax_a\{P^a_{ss'}V^*(s')\}$$

Linear and dynamic programming

If transition model and reward function are known

Step 1

$$\pi(s) := \arg\max_a \left\{ \sum_{s'} P^a_{s,s'} \left(r(s') + \gamma V(s') \right) \right\}$$

Step 2

$$V(s) := \sum_{s'} P^{a=\pi(s)}_{s,s'} \left(r(s') + \gamma V(s') \right)$$

Properties

Converges eventually toward the optimum if all states, actions are considered.

Value iteration

Bellman equation

Iterate

$$V_{k+1}(s) := \max_a \left\{ \sum_{s'} P^a_{s,s'} \left(r(s') + \gamma V_k(s') \right) \right\}$$

Stop when

$$max_s |V_{k+1}(s) - V_k(s)| < \epsilon$$

Initialisation

- arbitrary
- educated is better — see Inverse Reinforcement Learning

Policy iteration

Principle

- Modify π — step 1
- Update V until convergence — step 2

Getting faster

- Don't wait until V has converged before modifying π.

Discussion

Policy and value iteration

- Must wait until the end of the episode
- Episodes might be long

Can we update V on the fly ?

- I have estimates of how long it takes to go to RER, to catch the train, to arrive at Cité-U
- Something happens on the way (bump into a friend, chat, delay, miss the train,...)
- I can update my estimates of when I'll be home...

TD(0)

1. Initialize V and π
2. Loop on episode
   2.1 Initialize s
   2.2 Repeat

$$\begin{aligned}
&\text{Select action } a = \pi(s) \\
&\text{Observe } s' \text{ and reward } r \\
&V(s) \leftarrow V(s) + \alpha(\underbrace{r + \gamma V(s')}_{R} - V(s)) \\
&s \leftarrow s'
\end{aligned}$$

   2.3 Until s' terminal state

Discussion

Update on the spot ?

- Might be brittle
- Instead one can consider several steps

$$R = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})$$

Find an intermediate between

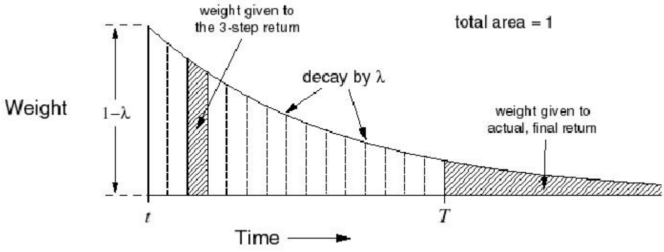
- Policy iteration

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots$$

- TD(0)

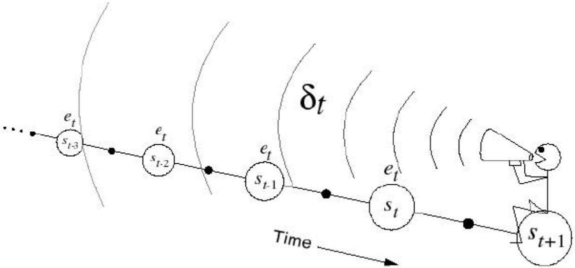
$$R_t = r_{t+1} + \gamma V_t(s_{t+1})$$

TD(λ), intuition



$$R_t^{\lambda} = \underbrace{(1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)}}_{} + \underbrace{\lambda^{T-t-1} R_t}_{}$$

TD(λ), intuition, followed



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

TD(λ)

1. Initialize V and π
2. Loop on episode
   2.1 Initialize s
   2.2 Repeat

$$a = \pi(s)$$
Observe s' and reward r
$$\delta \leftarrow r + V(s') - V(s)$$
$$e(s) \leftarrow e(s) + 1$$
For all s"
$$V(s'') \leftarrow V(s") + \alpha\delta e(s'')$$
$$e(s'') \leftarrow \gamma\lambda e(s'')$$
$$s \leftarrow s'$$

   2.3 Until s' terminal state

Q-learning

Principle: Iterate

- During an episode (from initial state until reaching a final state)
- At some point explore and choose another action;
- If it improves, update $Q(s, a)$:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \times \left[\overbrace{\underbrace{r(s_{t+1})}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})}_{\text{max future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right]$$

Equivalent to

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)(1 - \alpha) + \alpha[r(s_{t+1}) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})]$$

Partial summary

Notations

- State space $\mathcal{S}$
- Action space $\mathcal{A}$
- Transition model
  - deterministic: $s' = t(s, a)$
  - probabilistic: $P^a_{s,s'} = p(s, a, s') \in [0, 1]$.
- Reward $r(s)$ **bounded**
- Time horizon H (finite or infinite)

Policy $\pi \leftrightarrow$ Value function $V(s)$ (ou $Q(s, a)$)

1. Update V Iterate [until convergence]
2. Modify π

Reinforcement Learning, 2

Strengths

- Optimality guarantees (converge to global optimum)...

Weaknesses

- ...if each state is visited often, and each action is tried in each state
- Number of states: exponential wrt number of features

Behavioral cloning

Sammut, Bain 95

Input

- Traces (s_t, a_t) of expert

Supervised learning

- Learn $\hat{h}(s_t) = a_t$

Limitations

- Expert's mistakes
- Mistakes of $\hat{h}$: unbounded consequences

Inverse Reinforcement Learning

Abbeel, Ng, 2004

Input

- Traces (s_t, a_t) of expert

Supervised learning

- Learn V t.q. $V(s_t, a_t) > V(s_t, a')$

Limitations

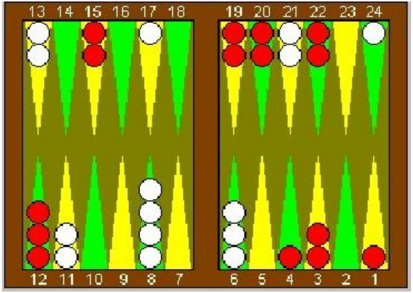
- Expert's mistakes
- Requires appropriate representation

More ?

http://videolectures.net/ecmlpkdd2012_abbeel_learning_robotics/

Overview

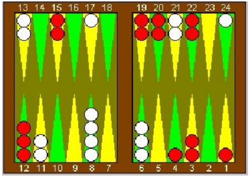
Dynamic programming & Learning



Backgammon Gerald Tesauro, 89-95

- State: raw description of a game (number of White or Black checkers at each location) $\mathbb{R}^D$
- Data: set of games
- A game: sequence of states $x_1, \ldots x_T$; value on last y_T: wins or loses

Dynamic programming & Learning



Learning

- Learned: $F : \mathbb{R}^D \mapsto [0, 1]$ s.t.

$$\text{Minimize } |F(x_T) - y_T|; \qquad |F(x_\ell) - F(x_{\ell+1})|$$

- Search space: F is a neural net $\equiv w$ $\mathbb{R}^d$
- Learning rule 200,000 games

$$\Delta w = \alpha(F(x_{\ell+1}) - F(x_\ell)) \sum_{k=1}^{\ell} \lambda^{\ell-k} \nabla w F(x_k)$$

Preference-based Value Learning

Cheng et al. 2011

Motivation

- Value depends on (numerical) reward functions
- ...adjusted by trial and errors... (what is the cost of an injury ?)

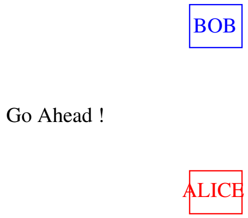
Proposed approach

- In state s, trigger action $a \in \mathcal{A}$, then apply policy π roll-out
- Compare trajectories: $(s, a, s_1, a_1, \ldots); (s, a', s'_1, a'_1, \ldots)$
- Use preference learning: define $a <_{s,\pi} a'$

Direct Value Learning

Murphy's law

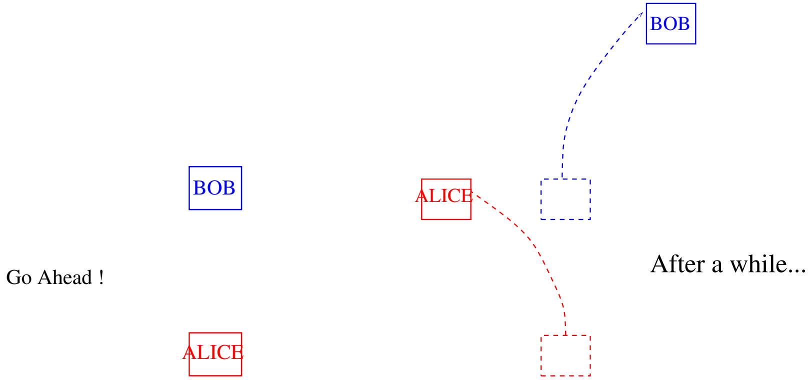
When in good situation, will be degraded



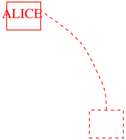
Direct Value Learning

Murphy's law

When in good situation, will be degraded

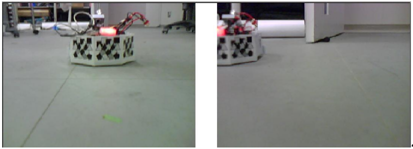


Direct Value Learning, 2

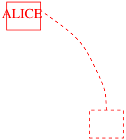


Consider Alice's trajectory

$$s_0 \succ s_1 \ldots \succ s_T$$

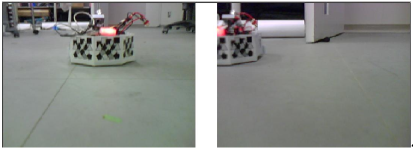
s_0  s_T

Direct Value Learning, 2



Consider Alice's trajectory

$$s_0 \succ s_1 \ldots \succ s_T$$

s_0  s_T

Preference-based Value Learning

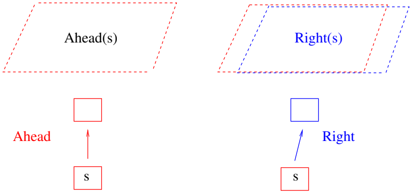
$$V(s) = \langle w^*, s \rangle$$

s.t.

$$w^* = argmin||w||^2 \text{ s.t. } \langle w, s_t \rangle > \langle w, s_{t+1} \rangle + 1$$

Approximate transition model

Given s and $Ahead(s)$, one can estimate $Right(s)$

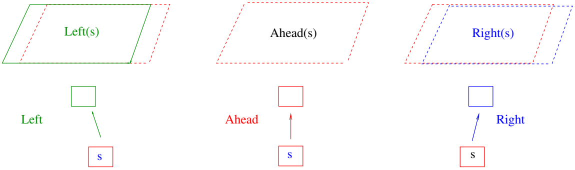


$$Right(s) \approx \text{ toric translation } \{Ahead(s)\}$$

DiVa controller

At time t, the best action at time $t-1$ can be estimated

$$a^*_{t-1} = argmax\{V^*(action(s_t)),\ \text{action} \in \mathcal{A}\}$$



Continuity assumption

$$\pi(s_t) = a^*_{t-1}$$

Experimental setting

Context

- Pandaboard, dual-core ARM Cortex-A9 OMAP4430,
- each core running at 1 GHz
- 1 GB DDR2 RAM.
- USB camera with resolution (320$\times$240), and color depth of monochrome 8bit.

Train/test

- Train: 11 runs, 64 time steps, Alice located behind Bob, both with a Go Ahead controller.
- Test: Bob equipped with a Braitenberg controller, Alice with a DiVa controller.

Goal of experiments

Compare and assess

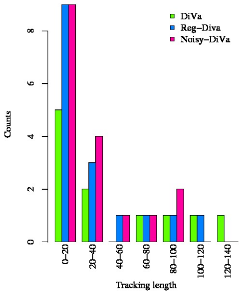
- DiVa
- Noisy-DiVa (irrelevant states)

- Regression-DiVa
  Learn V^* using regression instead of ranking.

Approximate transition model

Approximation guarantees ?

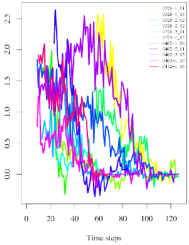
How long does Alice follow Bob ?

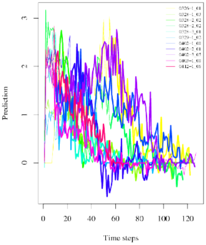


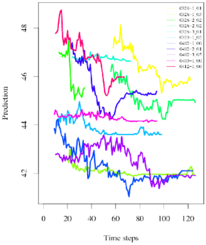
2 frames per second

The value function

Setting: Leave one out

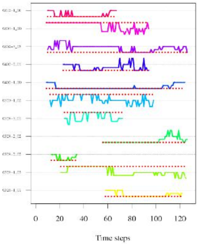




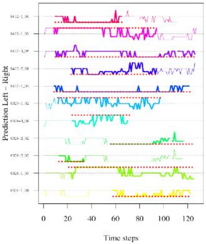


DiVa Noisy-DiVa Regression-DiVa

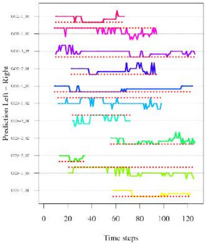
DiVa controller on training data (Leave one out)



DiVa

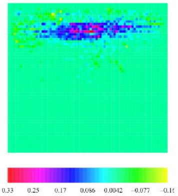


Noisy-DiVa

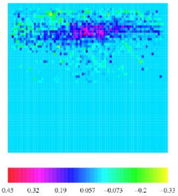


Regression-DiVa

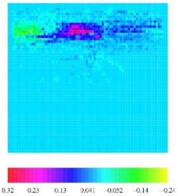
Value weights: sensitivity to toric translation



DiVa

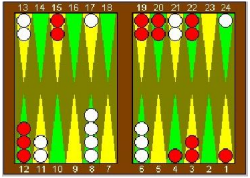


Noisy-DiVa



Regression-DiVa

Discussion



DiVa versus TD-Gammon Tesauro 02

- Scarce data while TD-Gammon used self-play
- DiVa uses ranking
- TD-Gammon sets the value of end state (win/loss) + min total variation

Perspectives

1. Dimensionality reduction — Riedmiller 12
2. Mid-size action spaces
   estimate the best rotation
3. Application to robot docking