# Reinforcement Learning

Michèle Sebag ; TP : Diviyan Kalainathan, Laurent Cetinsoy
TAO, CNRS − INRIA − Université Paris-Sud

Jan. 15th, 2018

*Credit for slides: R. Sutton, F. Stulp*

# Where we are

**MDP** Main Building block

**General settings**

| | Model-based | Model-free |
|---|---|---|
| Finite | Dynamic Programming | Discrete RL |
| Infinite | (optimal control) | **Continuous RL** |

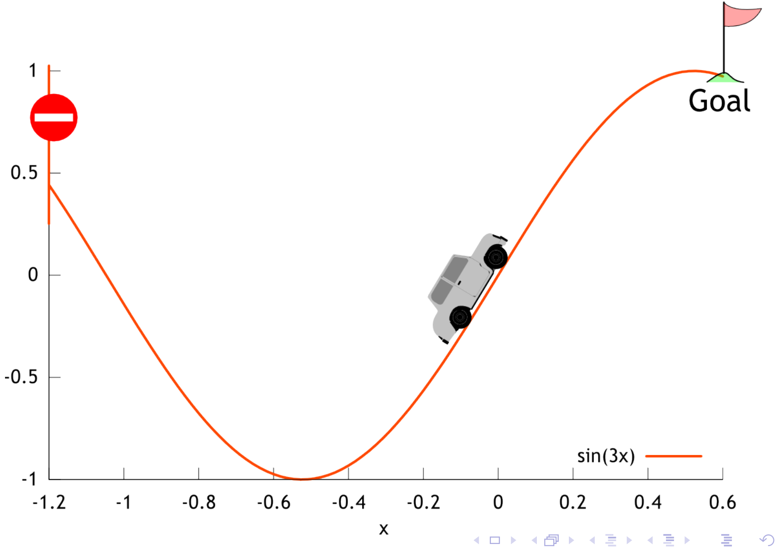
This course: Function approximation

# Why function approximation ?

**Exploration needed**: in each state, try every action.

**Impossible**

- In continuous state space



# Why function approximation ?

**Exploration needed**: in each state, try every action.
**Impossible**

- In large finite state space


**More** *Playing Atari with Deep Reinforcement Learning*, Mnih et al., 2015.
https://www.cs.toronto.edu/ vmnih/docs/dqn.pdf

# A learning problem (1/2)

## Notations

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

## Goal

$$\text{Build } V : \mathcal{S} \mapsto \mathbb{R}$$

## Remind: Supervised Machine Learning

$$\mathcal{E} = \{(\mathbf{x}_i, y_i), \mathbf{x}_i \in \mathcal{X} \text{ (instance space) }, y_i \in \mathcal{Y} \text{ (label space) }, i = 1 \ldots n\}$$

- Classification: $\mathcal{Y} = \{-1, 1\}$ or $\{1, \ldots k\}$
- **Regression** $\mathcal{Y} = \mathbb{R}$

# A learning problem (2/2)

**Assume we have the training set**

$$\mathcal{E} = \{(s_i, V^*(s_i)), i = 1 \dots n\}$$

**Then**

- Find a hypothesis space $\mathcal{H}$
- Find an optimization criterion $\mathcal{L}$ (data fitting )
- Solve the optimization problem

$$\hat{V}^* = \underset{V \in \mathcal{H}}{arg\ opt} [\mathcal{L}(V)]$$

# A learning problem (2/2)

**Assume we have the training set**

$$\mathcal{E} = \{(s_i, V^*(s_i)), i = 1 \ldots n\}$$

**Then**

- Find a hypothesis space $\mathcal{H}$
- Find an optimization criterion $\mathcal{L}$ (data fitting + **regularization**)
- Solve the optimization problem

$$\hat{V}^* = \underset{V \in \mathcal{H}}{arg\ opt} [\mathcal{L}(V)]$$

# Not a standard learning problem (1/2)

**Standard supervised ML criteria**

$$\mathcal{L}(V) = \sum_{i=1}^{n} (V^*(s_i) - V(s_i))^2 + \mathcal{R}(V)$$

Minimize the average error.

**But**
In RL, one error is enough to lose the game... to fall down from the cliff... to kill the robot...

# Not a standard learning problem (1/2)

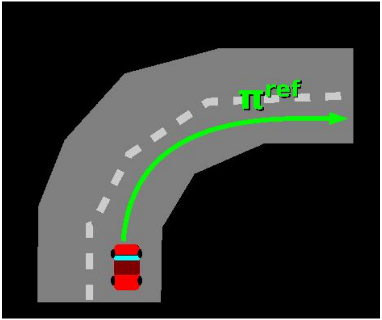
## Standard supervised ML criteria

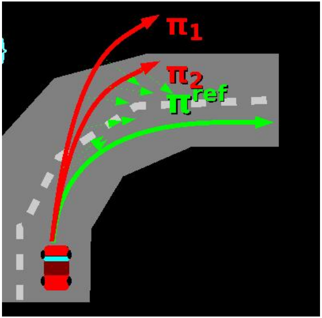
$$\mathcal{L}(V) = \sum_i (V^*(s_i) - V(s_i))^2 + \mathcal{R}(V)$$

Minimize the average error with respect to independent identically distributed $s_i$.

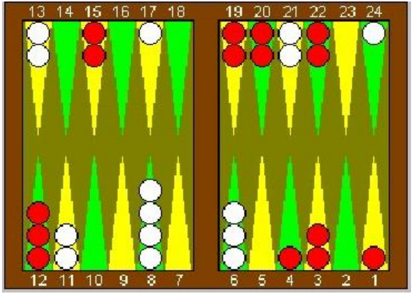
## But

A wrong move, or the transition error can send you off the road... and then the error might be cumulative.



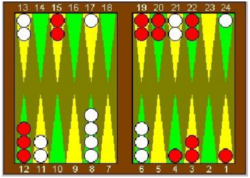
# Optimizing a pseudo-value: TD-Gammon, 1




**The game of Backgammon** Gerald Tesauro, 89-95

- State: vector of handcrafted features (e.g., number of White or Black checkers at each location) $\mathcal{S} \subset \mathbb{R}^D$
- Data: set of games
- A game: sequence of states $x_1, \ldots x_T$

## TD-Gammon, 2. Where does the value come from ?




**Assumptions**

$$y_0 = \quad .5 \qquad \text{value of initial state}$$

$$y_T = \begin{cases} 1 & \text{if } x_T \text{ is a winning state} \\ 0 & \text{if } x_T \text{ is a losing state} \end{cases}$$

**And for other states ?**

Value is supposed to be continuous

**Search space $\mathcal{H}$** Neural Nets $W$, weight vector in $\mathbb{R}^d$

**Learning criterion**

$$\text{Minimize } \left(V(x_T) - y_T\right)^2 + \sum_{\ell} \left(V(x_\ell) - V(x_{\ell+1})\right)^2$$

**Learning procedure: weight update**

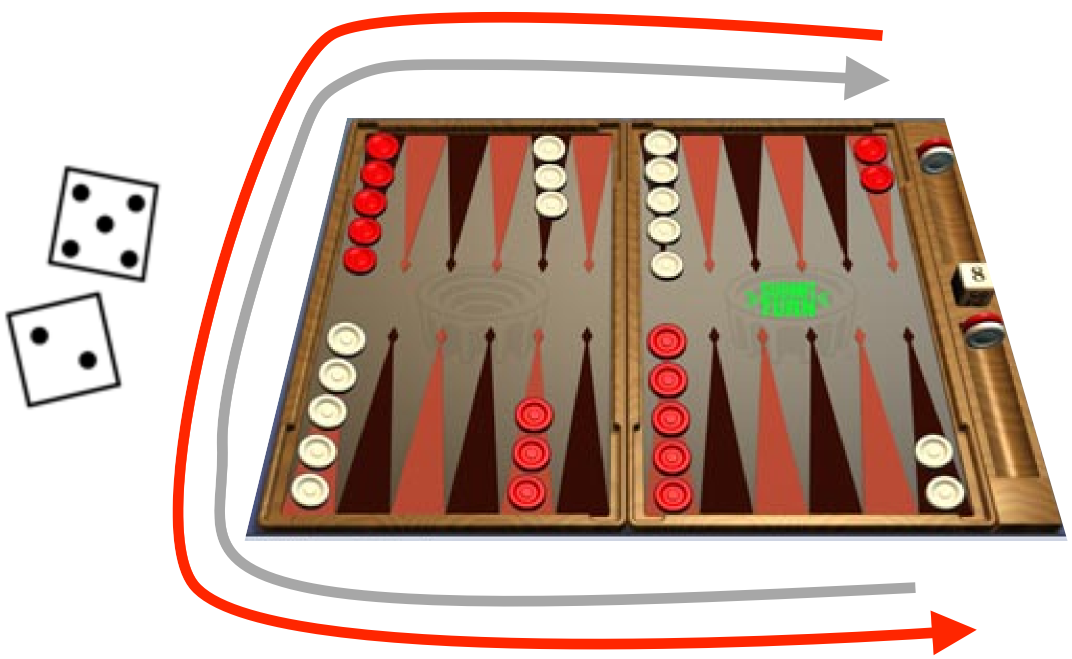
$$\Delta w = \alpha \left(V(x_{\ell+1}) - V(x_\ell)\right) \sum_{k=1}^{\ell} \lambda^{\ell-k} \nabla_w V(x_k)$$

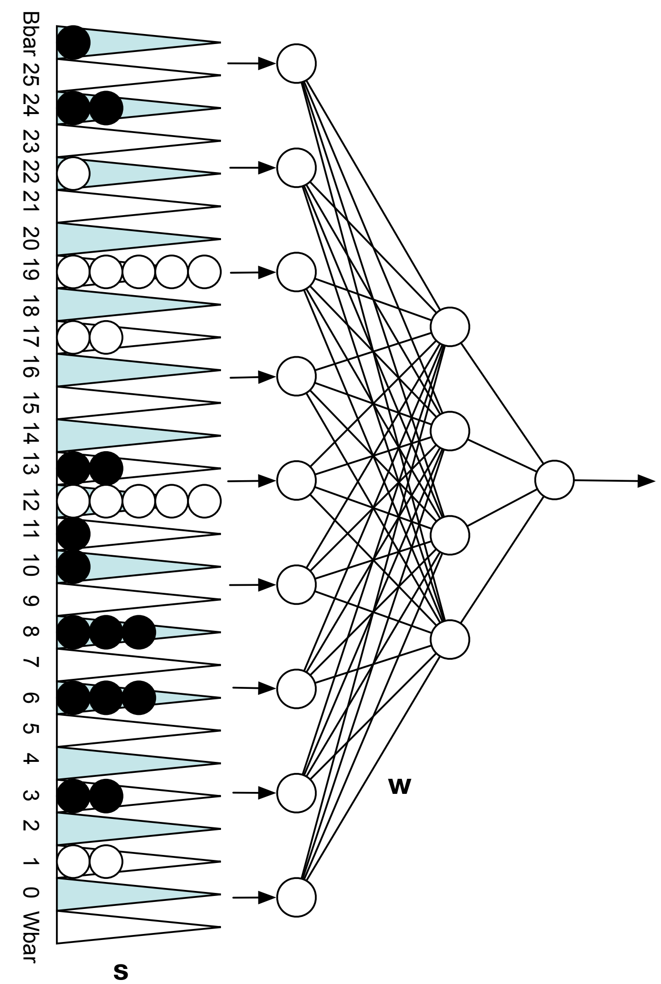
**Learning by Self-play**: Iteratively, 200,000 games

- Play using $V_i$ as value function
- Use games to retrain weight vector $W_i$
- Increment $i$

# Example: TD-Gammon

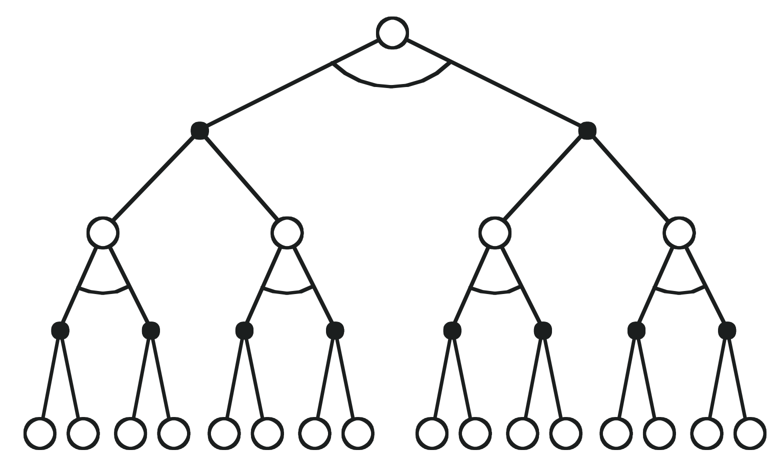
Tesauro, 1992-1995






estimated state value
(≈ prob of winning)

Action selection
by a shallow search



Start with a random Network

Play millions of games against itself

Learn a value function from this simulated experience

Six weeks later it's the best player of backgammon in the world

Originally used expert handcrafted features, later repeated with raw board positions

# Finding a representation

## Using basis functions

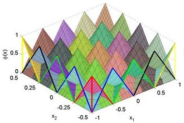
$$\phi_1 \dots \phi_K : \mathcal{S} \mapsto \mathbb{R}$$

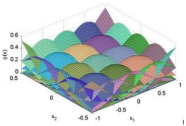
- Usually $\phi$ are normalized,

$$\sum_{i=1}^{K} \phi(s) = 1$$

Fuzzy memberships

Radius-basis functions

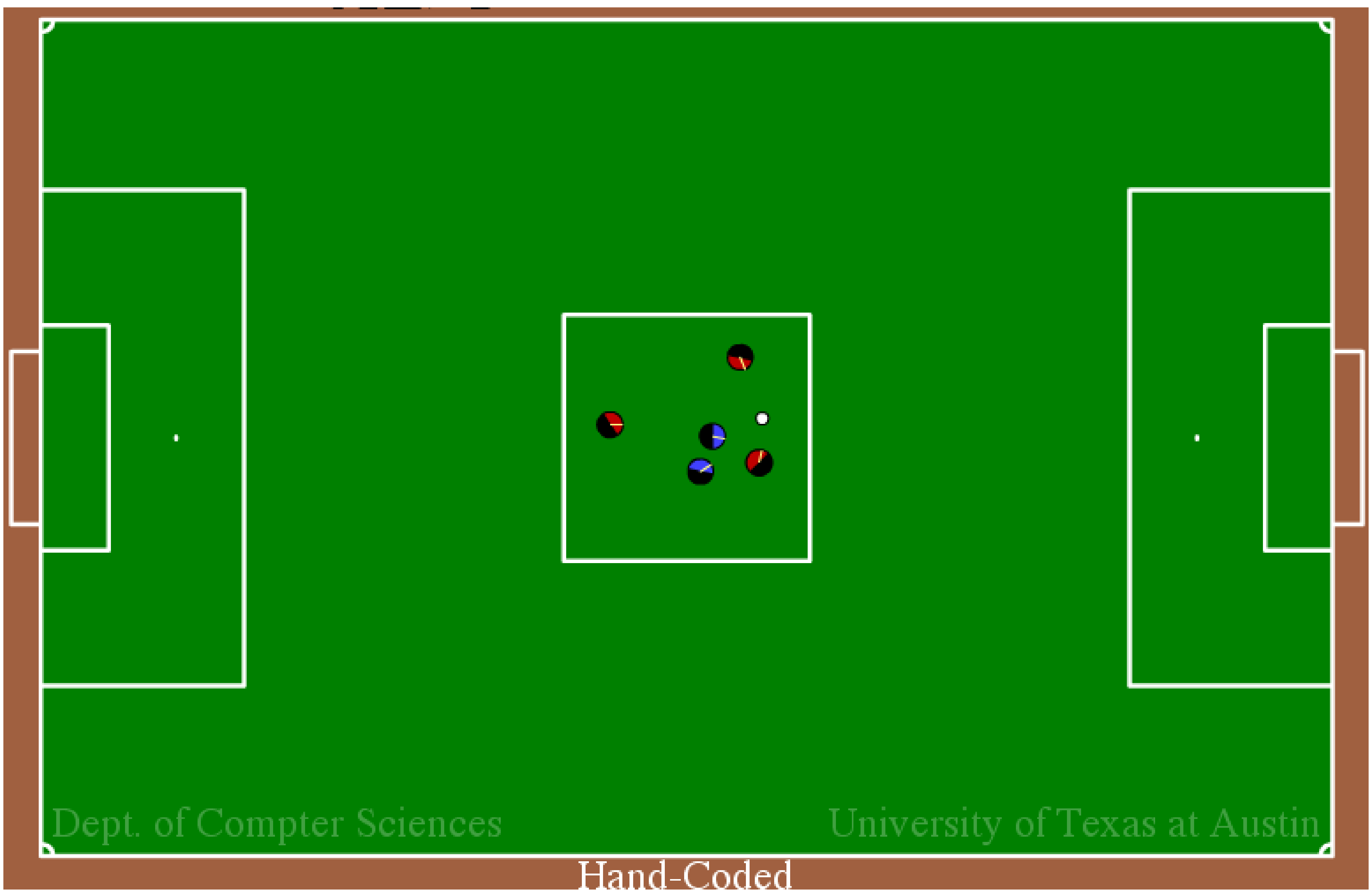


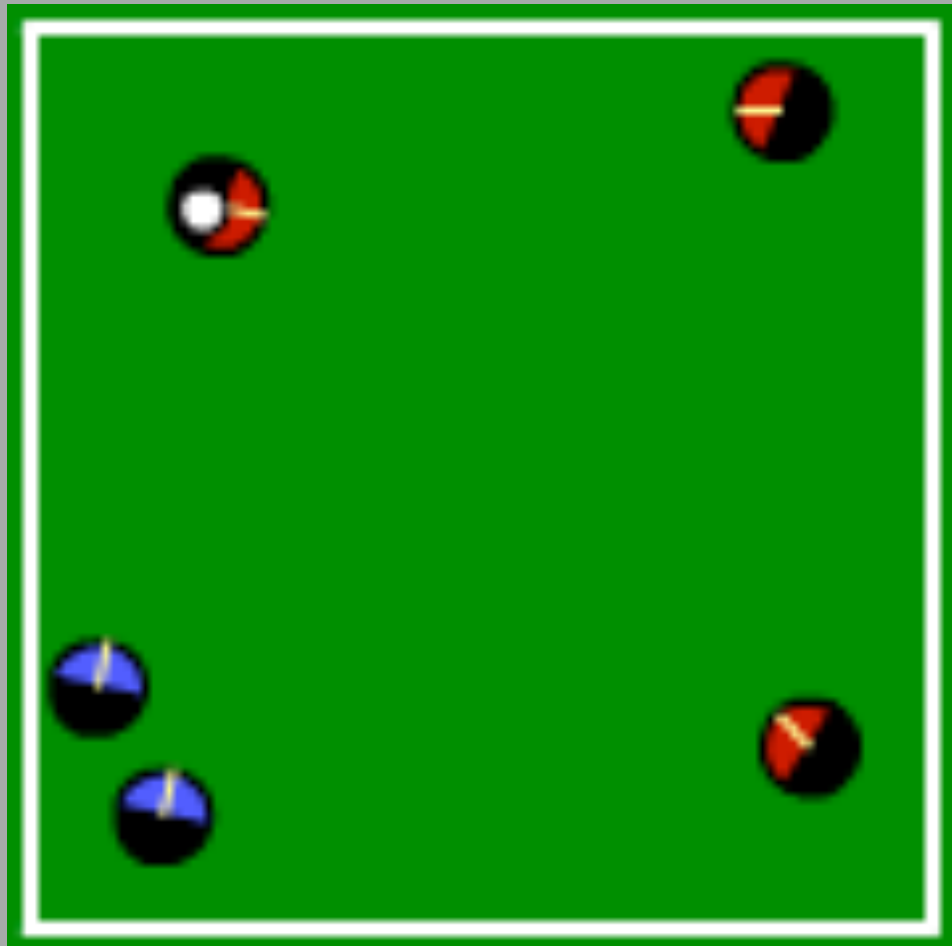


And then, back to Dynamic Programming.

# RoboCup soccer keepaway

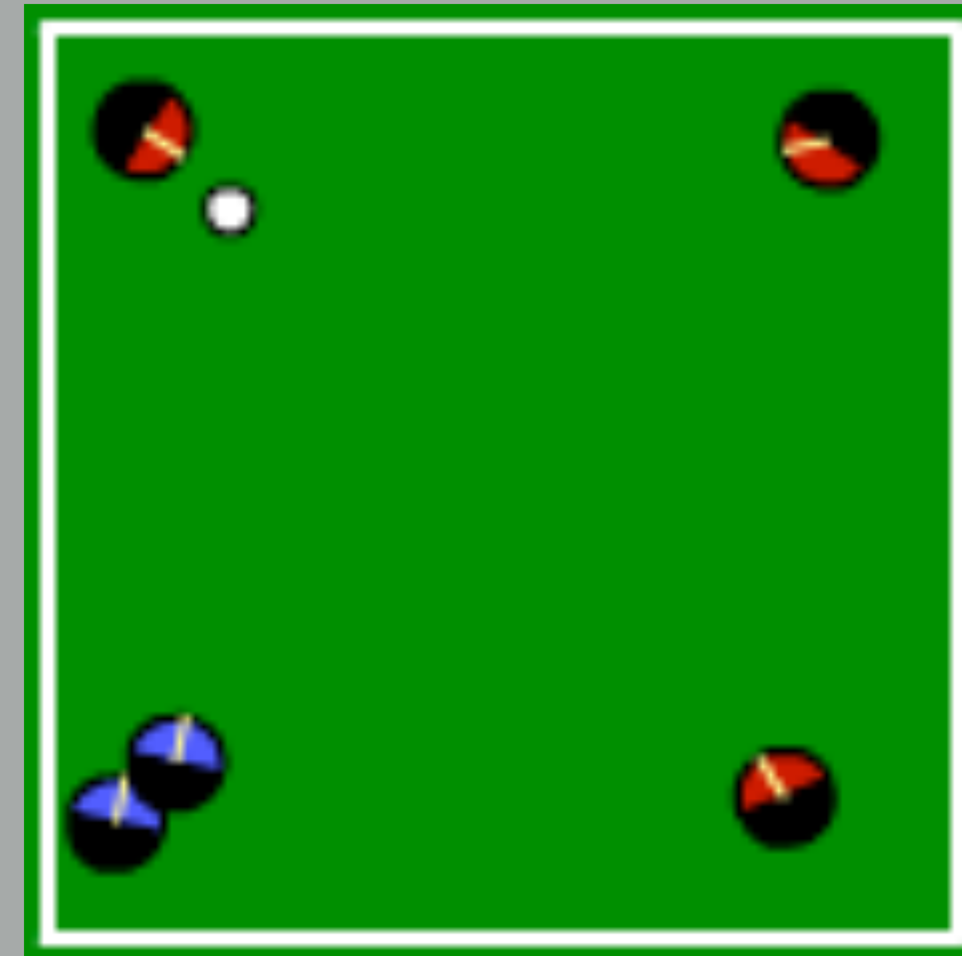
Stone, Sutton & Kuhlmann, 2005





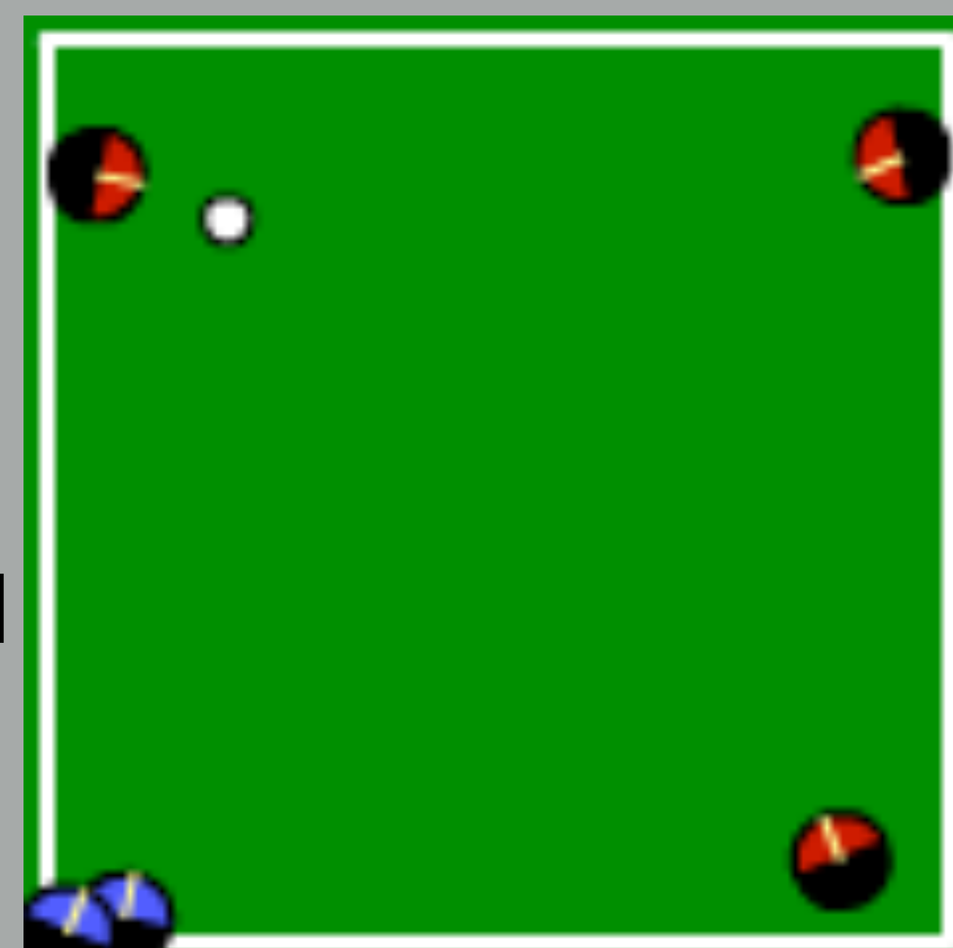
Random

Learned



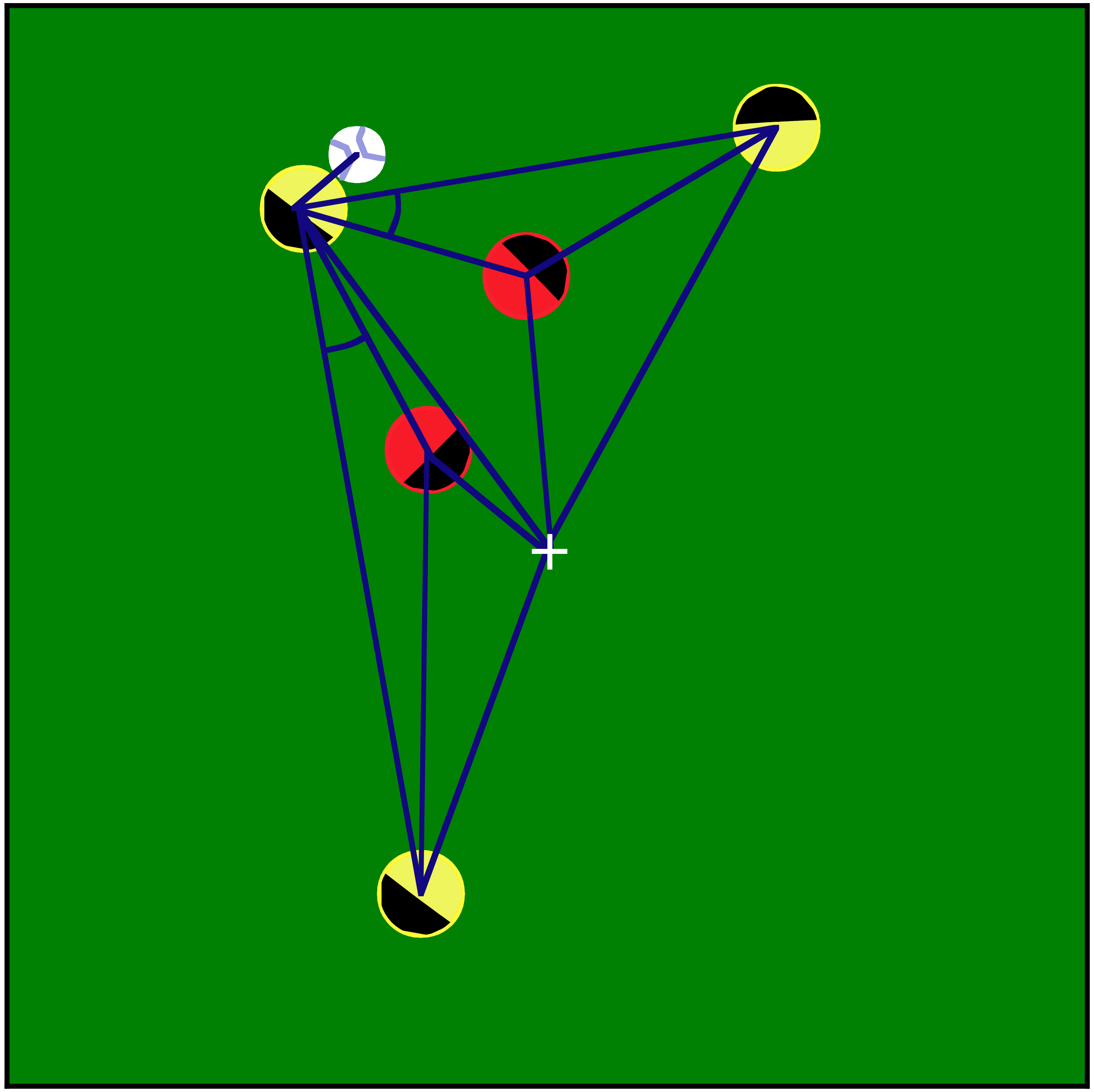
Hand-coded

Hold



Stone, Sutton & Kuhlmann, 2005

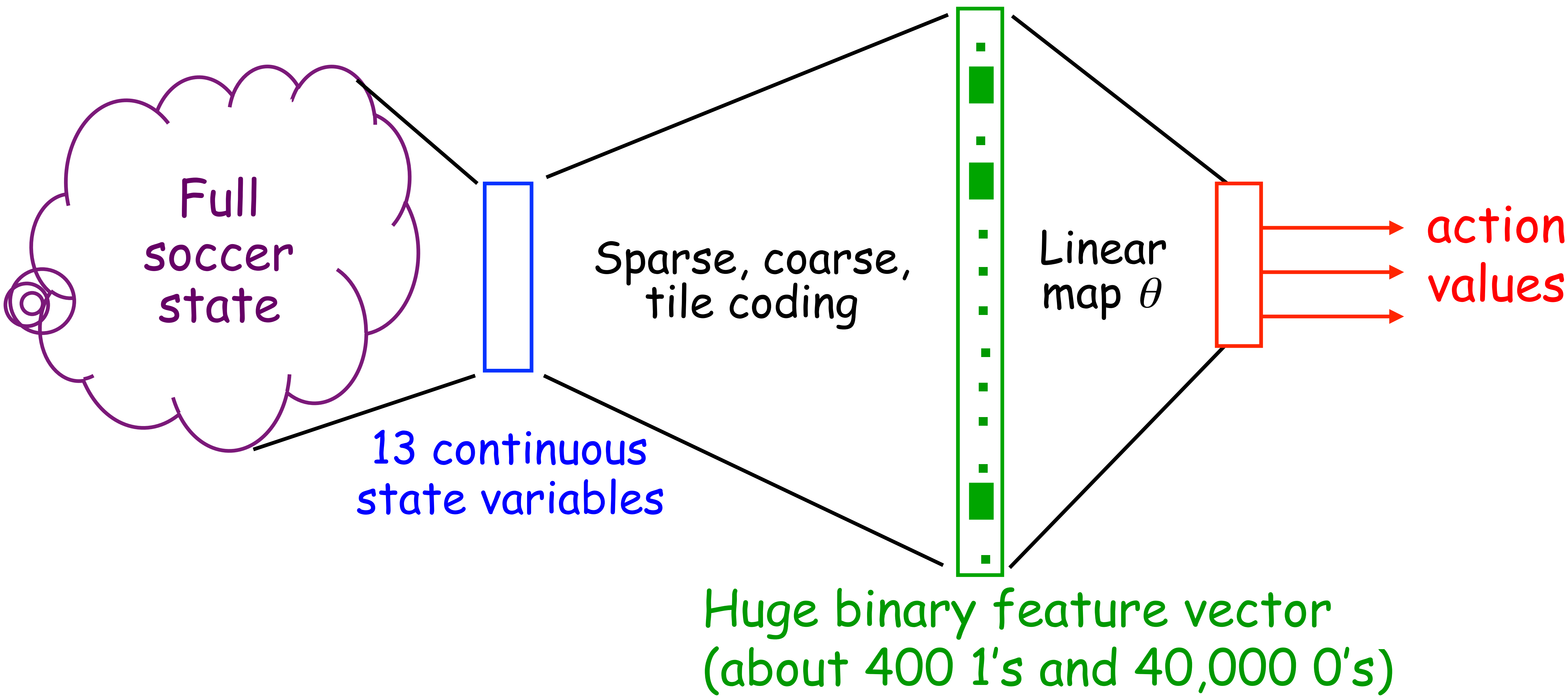
# How is the state encoded?
# In 13 continuous state variables



11 distances among the players, ball, and the center of the field

2 angles to takers along passing lanes

# The Feature-Construction Pipeline



# Parametric action-value function

**Find**

$$v(s, \theta) \approx V^*(s)$$

$$q(s, a, \theta) \approx Q^*(s, a)$$

**Search spaces**

- Linear approximation: (many) handcrafted features, and then find linear weights
- NN approximation **Deep Reinforcement Learning**

**What matters**

- **Linear** Learning complexity required to scale up to large problems
- **Self-play** to acquire examples in critical regions
- Online learning; dealing with **non-stationary** target value function

# Mean-square error, 1

## Optimization problem

$$\mathcal{L}(\theta) = \sum_{s \in \mathcal{S}} \left(v(s,\theta) - V^*(s)\right)^2$$

Any difficulties with this formulation ?

# Mean-square error, 1

## Optimization problem

$$\mathcal{L}(\theta) = \sum_{s \in \mathcal{S}} \mathbf{P(s)} \left( v(s, \theta) - V^*(s) \right)^2$$

# Mean-square error, 1

## Optimization problem

$$\mathcal{L}(\theta) = \sum_{s \in \mathcal{S}} \mathbf{P(s)} \left(v(s,\theta) - V^*(s)\right)^2$$

## Why using distribution $P$ ?

- $v(s,\theta)$ is an approximation: it has to make errors
- Not all errors are equally harmful: harmful errors must weight more.
- $P$ might reflect a uniform distribution;
  or the distribution associated to the current policy $\pi$ (on-policy learning);
  or to another policy used to acquire data (off-policy learning)
- Most generally, a new point $(s_t, V_t(s_t))$ is drawn and $\theta_t$ is updated using stochastic gradient.

# Mean-square error, 2

$$
\begin{aligned}
\theta_{t+1} &= \theta_t - \frac{1}{2}\alpha\nabla_{\theta_t}\left(V_t(s_t) - v(s,\theta_t)\right)^2 \\
&= \theta_t + \alpha\left(V_t(s_t) - v(s,\theta_t)\right).\nabla_{\theta_t} v(s,\theta_t)
\end{aligned}
$$

## Requirements

- $v(s,\theta_t)$ must be an unbiased estimate of the desired $V_t(s_t)$.
- not the case in general (except for Monte-Carlo); but practical.
- The approximation of the value function must allow for optimization, to define the policy by greedification:

$$\hat{\pi}(s) = \underset{a\in\mathcal{A}}{argmax}\left(\hat{q}(s,a,\theta^*)\right)$$

# Learning Criteria

## Notations

- For state $s$, push value toward backed-up value $v$ $\qquad s \mapsto v$

## Backed-up value

### Dynamic programming

$$s \mapsto \mathbb{E}\left[r(s) + \gamma V(s')\right]$$

### Monte-Carlo

$$s \mapsto r(s) + \sum_{t=1}^{T} \gamma^t r_t$$

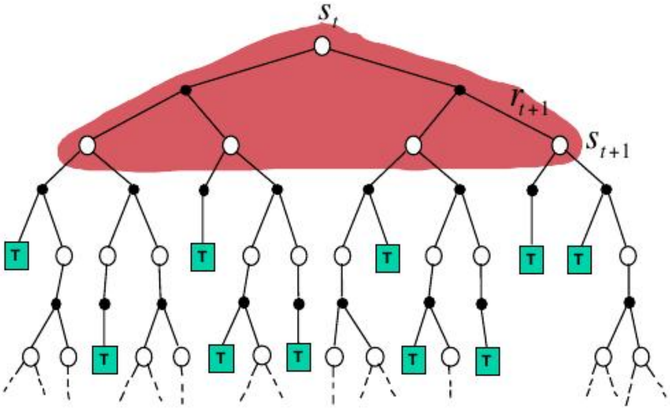
### TD(0)

$$s_t \mapsto r(s_t) + \gamma V(s_{t+1})$$

# Learning Criteria

$$s \mapsto \mathbb{E}\left[r(s) + \gamma V(s')\right]$$

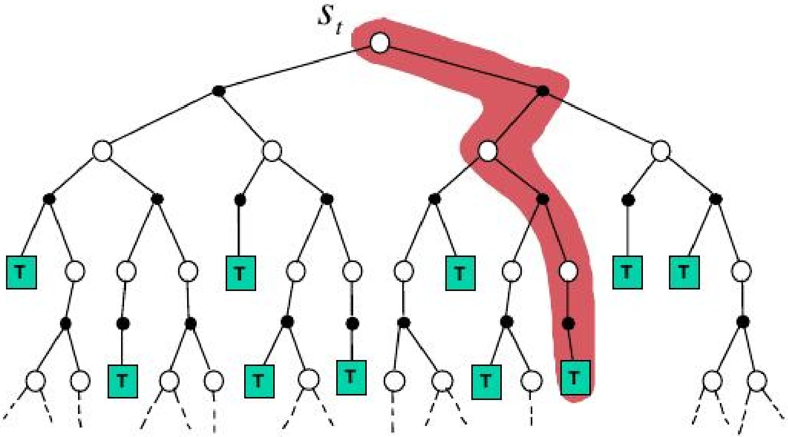
## Dynamic programming



# Learning Criteria

$$s \mapsto r(s) + \sum_{t=1}^{T} \gamma^t r_t$$

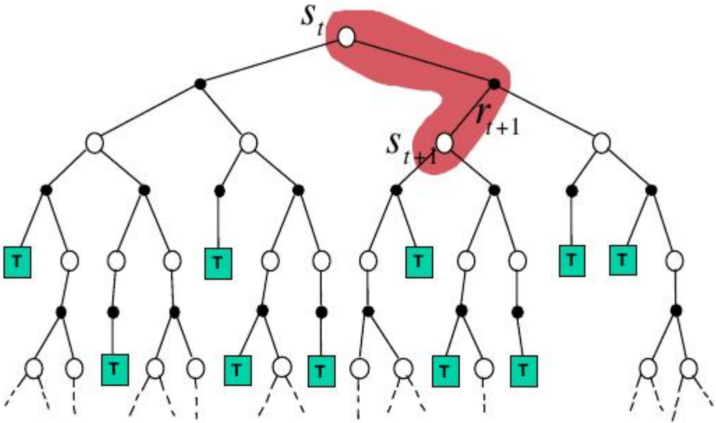
## Monte-Carlo



# Learning Criteria

$$s_t \mapsto r(s_t) + \gamma V(s_{t+1})$$

## Temporal Difference



# Semi-gradient Q-learning

Watkins 89

**Loss function**

Bellman optimality equation

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left(\underbrace{R_{t+1} + \gamma \max_{a\in\mathcal{A}} q(S_{t+1}, a, \theta)}_{target\ \ value} - q(S_t, A_t, \theta)\right)^2\right]$$

- target depends on $\theta$, let us ignore this and
- only take the derivative wrt $q(S_t, A_t, \theta)$:

$$\Delta\theta_t = \left(R_{t+1} + \gamma \max_{a\in\mathcal{A}} q(S_{t+1}, a, \theta_t) - q(S_t, A_t, \theta_t)\right) . \frac{\partial q(S_t, A_t, \theta_t)}{\partial \theta_t}$$

# Semi-gradient SARSA

Sutton 89, Rummery 94

**Loss function**

Bellman expectation equation

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left(\underbrace{R_{t+1} + \gamma q(S_{t+1}, A_{t+1}, \theta)}_{target \;\; value} - q(S_t, A_t, \theta)\right)^2\right]$$

- again target depends on $\theta$ and we ignore this,
- taking the derivative wrt $q(S_t, A_t, \theta)$:

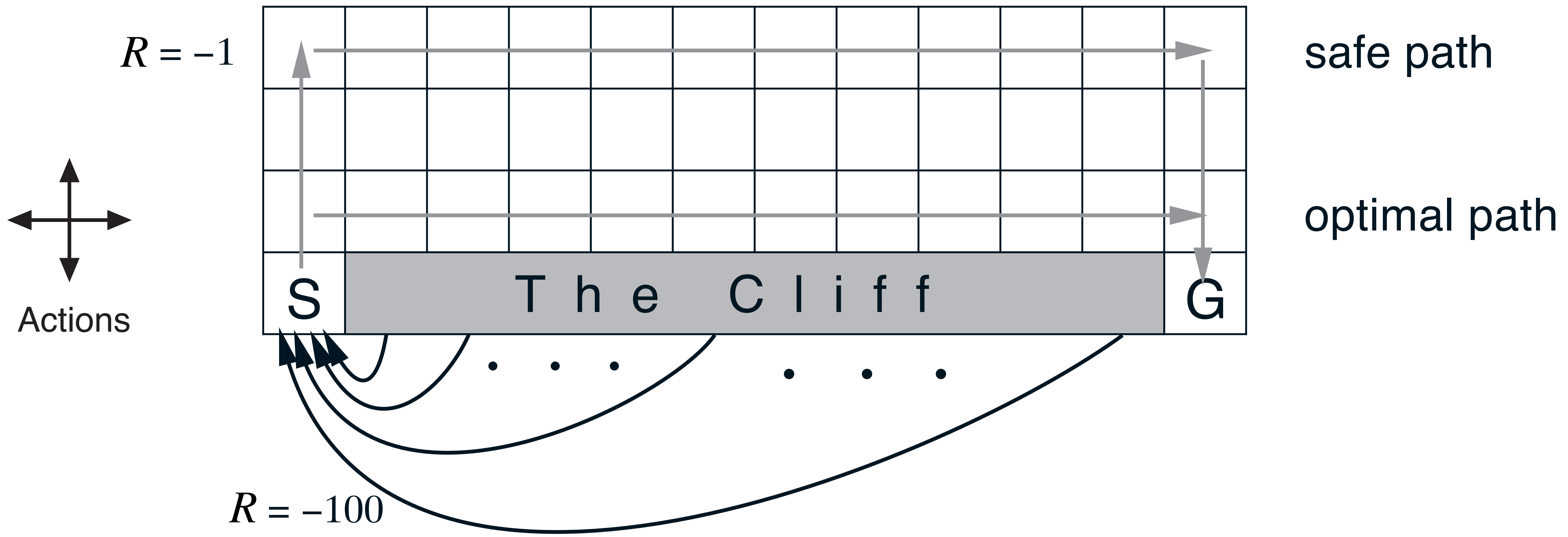
$$\Delta\theta_t = (R_{t+1} + \gamma q(S_{t+1}, A_{t+1}, \theta_t) - q(S_t, A_t, \theta_t)) \cdot \frac{\partial q(S_t, A_t, \theta_t)}{\partial \theta_t}$$

**Remark**

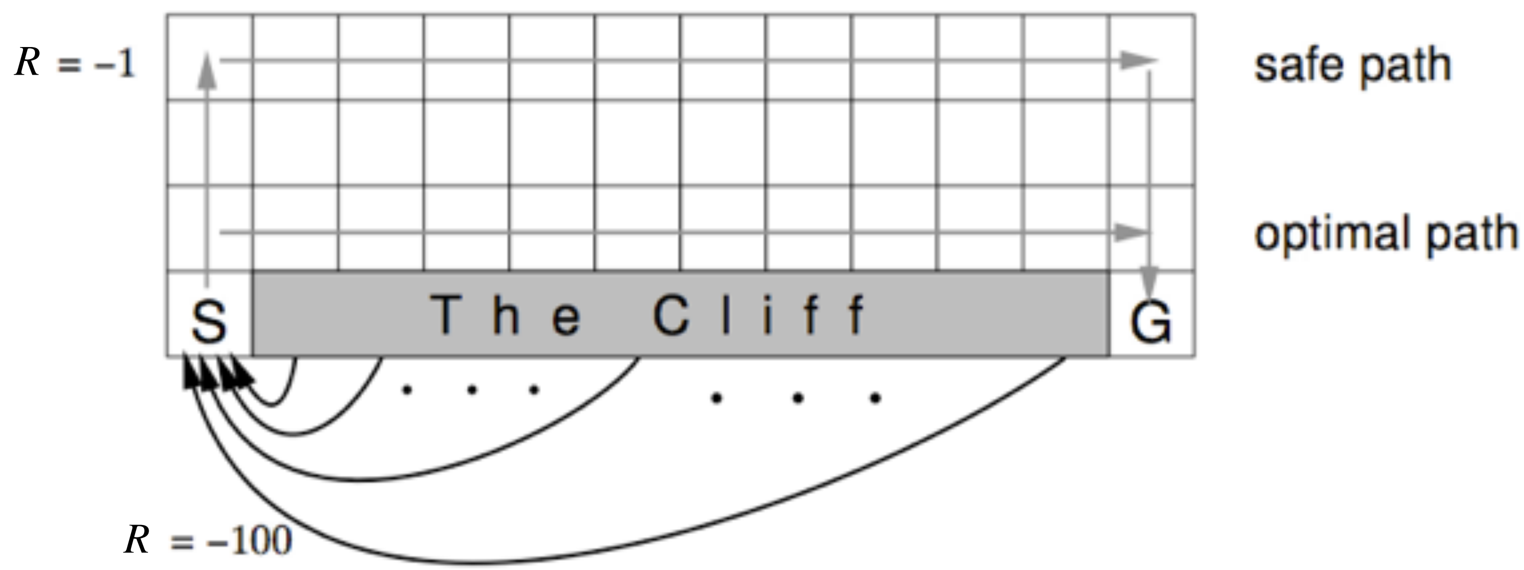
- This is an on-policy algorithm: it approximates $q^\pi$ not $Q^*$.
- Therefore $\pi$ should incorporate some exploration (be $\epsilon$-greedy)

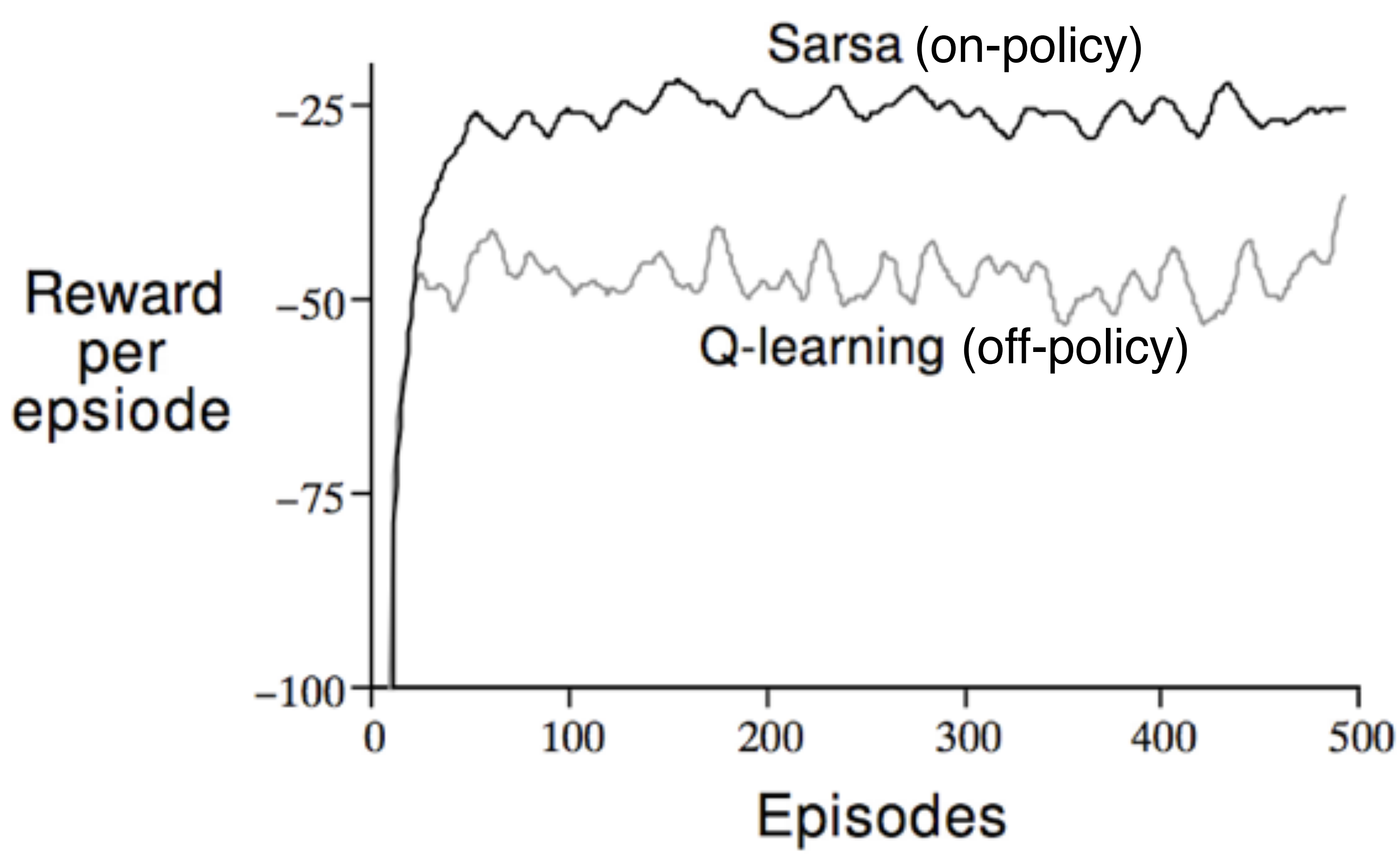
https://www.youtube.com/watch?v=ggqnxyjaKe4: on-policy performs better but finds poorer policies. (next slide).

# Cliff-walking example (on-policy vs off-policy)



# Cliff-walking example (on-policy vs off-policy)




both algorithms
are ε−greedy
ε = 0.1

# Fitted Q iteration

Ernst et al. 2005

**Principle** iterating over the time horizon

- Given a set of four-tuples $(s, a, r, s')$
- First iteration:

$$\hat{q}_1(s, a) \approx r(s, a)$$

- iteration $N$:

$$\hat{q}_n(s_t, a_t) \approx r(s_t, a_t) + \gamma \max_{a \in \mathcal{A}} \hat{q}_{n-1}(s_{t+1}, a)$$

- Successive calls to the supervised learning algorithm are independent: possible to adapt the resolution/complexity depending on the iteration and the available sample.

**Search space**: Decision trees

- Non parametric; flexible
- Scalability wrt high-dimensional spaces
- Robustness wrt irrelevant features, noise, outliers.

# Trees in Fitted Q iteration

## Decision tree

Quinlan 89; Breiman 86

- Select cutting feature and cutting threshold to maximize the average variance reduction of the output variable
- Select hyper-parameter (min number of examples in a leaf) by cross-validation

## Bagged trees

Breiman 96

- $M$ times — $M$ hyper-parameter
- Bootstrap the training set
- Grow a decision tree from the bootstrapped data

## KD-tree

- In each node at depth $d$: cutting feature is $i$-th feature if $d < \#$ features
- cutting threshold: median of the $f_i$ value in the training set
- (does it change among iterations ?)

## Random Forests

Breiman 01; Geurts 04

- Like Bagged trees, except
- Sample a number $K$ of (cutting feature, cutting threshold), return the best one

# Trees in Fitted Q iteration, 2

Note $l$ a leaf in a tree

$$q(s,a) = \sum_{trees} \sum_{l} k(s,a,l) v(l)$$

with

$$k(s,a,l) = \frac{1_{(s,a) \in l}}{\sum_i 1_{(s_i,a_i) \in l}}$$

**Property**

$$||\hat{q}_n(s,a)||_\infty \leq B + \gamma ||\hat{q}_{n-1}(s,a)||_\infty$$

with $\hat{q}_0(s,a) = 0$.
Therefore

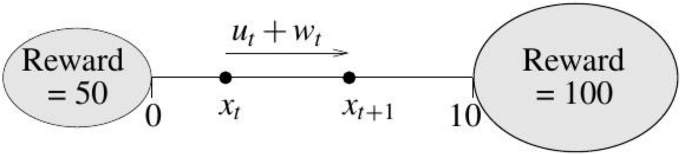
$$||\hat{q}_n(s,a)||_\infty \leq \frac{B}{1-\gamma}$$

with $B$ a bound on the reward.

# The RiverSwim

Ernst et al, 05

## The problem




11 states $(0, 1, \ldots 10)$
2 actions, right or left
rewards on terminal states 0 or 10.

## The results

1. Bellman residuals wrt number $\#\mathcal{F}$ of 4-tuples.

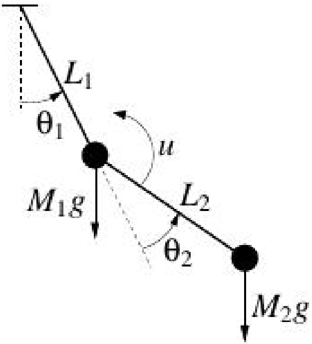
| Tree-based method | $\#\mathcal{F}$ | | |
|---|---|---|---|
| | 720 | 2010 | 6251 |
| Pruned CART Tree | 2.62 | 1.96 | 1.29 |
| Pruned Kd-Tree | 1.94 | 1.31 | 0.76 |
| Pruned Tree Bagging | 1.61 | 0.79 | 0.67 |
| Pruned Extra-Trees | 1.29 | 0.60 | 0.49 |
| Pruned Tot. Rand. Trees | 1.55 | 0.72 | 0.59 |

2. But the score is about the same for all methods

# The Acrobot

Ernst et al, 05

## The problem



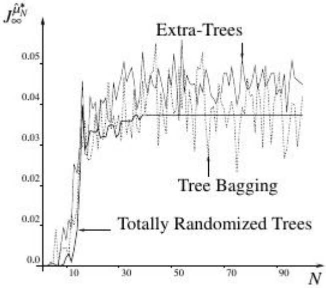

state in $\mathbb{R}^4$: $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)$
action: torque $u$ = -5 or 5
reward: distance to up-equilibrium position, if $< 1$ (then terminates)

# The Acrobot

## The results

$\#\mathcal{F} \approx 150{,}000$ tuples

1. The return




2. Comparative performances

| Tree-based method | Policy which generates $\mathcal{F}$ | |
|---|---|---|
| | ε-greedy | Random |
| Pruned CART Tree | 0.0006 | 0. |
| Kd-Tree (Best $n_{min}$) | 0.0004 | 0. |
| Tree Bagging | 0.0417 | 0.0047 |
| Extra-Trees | 0.0447 | 0.0107 |
| Totally Rand. Trees | 0.0371 | 0.0071 |

When

Learning or Optimizing the Value ?

Approximating Value (gradient)

Approximating Value (decision tree)

Summary

# Function approximation for RL: Summary

**Goal**

Learn an approximation $\hat{v}$ of the value function; define $\hat{\pi}$ from $\hat{v}$

**Ingredients**

- Data
- Learning criterion
- Learning procedure

# Function approximation for RL: Summary

**Goal**

Learn an approximation $\hat{v}$ of the value function; define $\hat{\pi}$ from $\hat{v}$

**Ingredients**

- Data **off-line; online**
- Learning criterion
- Learning procedure

# Function approximation for RL: Summary

**Goal**

Learn an approximation $\hat{v}$ of the value function; define $\hat{\pi}$ from $\hat{v}$

**Ingredients**

- Data **off-line; online**
- Learning criterion **data fitting; Bellman residual**
- Learning procedure

# Function approximation for RL: Summary

**Goal**

Learn an approximation $\hat{v}$ of the value function; define $\hat{\pi}$ from $\hat{v}$

**Ingredients**

- Data **off-line; online**
- Learning criterion **data fitting; Bellman residual**
- Learning procedure **knn; decision trees; gradient (linear or NN)**

# Function approximation for RL: Summary, 2

## Comments

1. Required to scale up
2. Pitfalls:
   - Sufficient representation needed (if large representation, robust learning required, e.g. decision trees)
   - Self-play / replay mandatory
   - A further stage of optimization is required to define $\hat{\pi}$

   - Pathologies: gradient can blow up (see Fig. 8.13, Sutton Barto)

# Function approximation for RL: Summary, 2

## Comments

1. Required to scale up
2. Pitfalls:
   - Sufficient representation needed (if large representation, robust learning required, e.g. decision trees)
   - Self-play / replay mandatory
   - A further stage of optimization is required to define $\hat{\pi}$

   - Pathologies: gradient can blow up (see Fig. 8.13, Sutton Barto)

## After all

- **Value is a means for building a policy**
- **Can we build the policy directly ? next course**