# Least Squares SVM for Least Squares TD Learning

Tobias Jung, University of Mainz, Germany

Daniel Polani, University of Hertfordshire, U.K.

# Contents

**Will talk about:**

- **Background:** Approximate dynamic programming, approximate policy iteration
- **Methods:**
  - Least squares policy evaluation (LS-TD)
  - Kernel-based learning (LS-SVM)
  - Subset of regressors approximation
- **Experiments:** Toy examples + a bigger 'real-world' problem

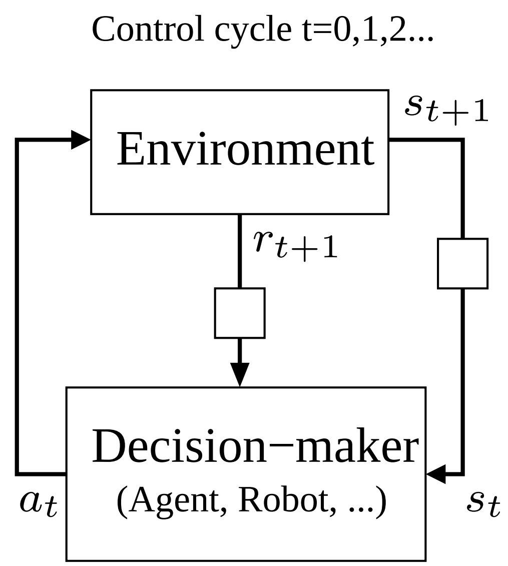
**Motivation:** combine the best of two worlds:

- Least squares TD learning (much faster convergence than plain TD($\lambda$))
- Least squares SVM (superior to fixed parameterized function approximators)

# Dynamic Programming I

A Markov decision process basically consists of

- **States** $\mathcal{S} = \{s_1, \ldots, s_N\}$
- **Actions** $\mathcal{A} = \{a_1, \ldots, a_M\}$
- **Rewards** $R(s'|s,a)$
- **Transition probabilities** $P(s'|s,a)$ (Markov)




**Objective:** choose actions to maximize performance

**Hitch:** usually *delayed reward*

$\Longrightarrow$ choose actions to maximize **long-term reward**

# Dynamic Programming II

**Criterion:** expected infinite-horizon discounted sum of rewards

**How do we get there?**

- **Policy:** $\pi : \mathcal{S} \longrightarrow \mathcal{A}$ (deterministic,stationary)
- **Value function:** ($0 < \gamma < 1$ discount factor)

$$V^{\pi}(s) = E^{\pi}\left\{\sum_{t \geq 0} \gamma^t R(s_{t+1}|s_t, a_t) \mid s_0 = s, a_t = \pi(s_t)\right\} \quad \forall s$$

- **Bellman says:** $V^{\pi}$ obeys fixed-point relation $V^{\pi} = \mathcal{T}_{\pi} V^{\pi}$, where

$$\left(\mathcal{T}_{\pi} V\right)(s) = \sum_{s'} P(s'|s, \pi(s)) \left[R(s'|s, \pi(s)) + \gamma V(s')\right]$$

**Goal:** seek optimal policy $\pi^* = \operatorname{argmax}_{\pi} V^{\pi}$, i.e. a policy $\pi^*$ satisfying $V^{\pi^*}(s) \geq V^{\pi}(s)$, $\forall s, \forall \pi$

# Dynamic Programming III

**Now:** many ways to obtain $\pi^*$, e.g. methods based on **policy iteration:**

**Policy iteration:** choose initial policy $\pi_0$. Iterate for $k = 0, 1, \ldots$

- **Policy evaluation:** compute $V^{\pi_k}$
- **Policy improvement:** derive greedy policy $\pi_{k+1}$ from $V^{\pi_k}$:

$$\pi_{k+1}(s) = \operatorname*{argmax}_a \left\{ \sum_{s'} P(s'|s,a) \left[ R(s'|s,a) + \gamma V^{\pi_k}(s') \right] \right\}, \forall s$$

**Problems:**

- Model $P(s'|s,a),\ R(s'|s,a)$ must be known $\longrightarrow$ simulation
- Number of states large or states $\in \mathbb{R}^d$ $\longrightarrow$ function approximation

$\Longrightarrow$ Approximate policy iteration, approximate policy evaluation (with least squares)

# Approximate dynamic programming

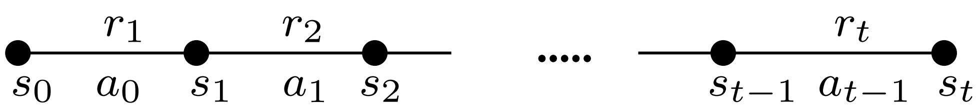
**Assume:** value function is linearly parameterized

$$\boxed{\tilde{V}(s) = [\boldsymbol{\phi}_m(s)]^T \mathbf{w}}$$

| | | |
|---|---|---|
| where | $\boldsymbol{\phi}_m(s) = [\phi_1(s), \ldots, \phi_s]$ | a $m \times 1$ feature vector |
| | $\mathbf{w}$ | a $m \times 1$ weight vector |
| | $\phi_i(s) : \mathcal{S} \to \mathbb{R}$ | basis function |

**Approximate policy evaluation:** choose initial policy $\pi_0$. Iterate for $k = 0, 1, 2, \ldots$

- Observe a long trajectory under fixed $\pi_k$ (e.g. agent interacts with environment)

  where $s_i \sim P(s_i | s_{i-1}, a_{i-1}), r_i = R(s_i | s_{i-1}, a_{i-1}), a_i = \pi(s_i)$

- Estimate $\tilde{V}^{\pi_k}$ using the trajectory (**approximate policy evaluation**)

- Derive $\pi_{k+1}$ as greedy policy from $\pi_k$

# Approximate policy evaluation with least squares

**Want:** to determine $\mathbf{w}$ in $V^{\pi_k}(s) = [\boldsymbol{\phi}_m(s)]^T \mathbf{w}$ from trajectory $s_0, s_1, \ldots, s_t$ and rewards $r_1, \ldots, r_t$.

- **Bellman residual minimization approach:** determine weights $\mathbf{w}$ by

$$\mathbf{w} = \operatorname*{argmin}_{\hat{\mathbf{w}}} \left\{ \sum_{i=0}^{t-1} [\boldsymbol{\phi}_m(s_i)]^T \hat{\mathbf{w}} - \sum_{s'} P(s'|s_i, \pi_k(s_i)) \left[ R(s'|s_i, \pi_k(s_i)) + \gamma [\boldsymbol{\phi}_m(s_{i+1})]^T \hat{\mathbf{w}} \right] \right\}^2$$

  For **deterministic transitions** we can use the observed trajectory:

$$\boxed{\mathbf{w} = \operatorname*{argmin}_{\hat{\mathbf{w}}} \left\{ \sum_{i=0}^{t-1} [\boldsymbol{\phi}_m(s_i)]^T \hat{\mathbf{w}} - \left[ r_i + \gamma [\boldsymbol{\phi}_m(s_{i+1})]^T \hat{\mathbf{w}} \right] \right\}^2 .}$$

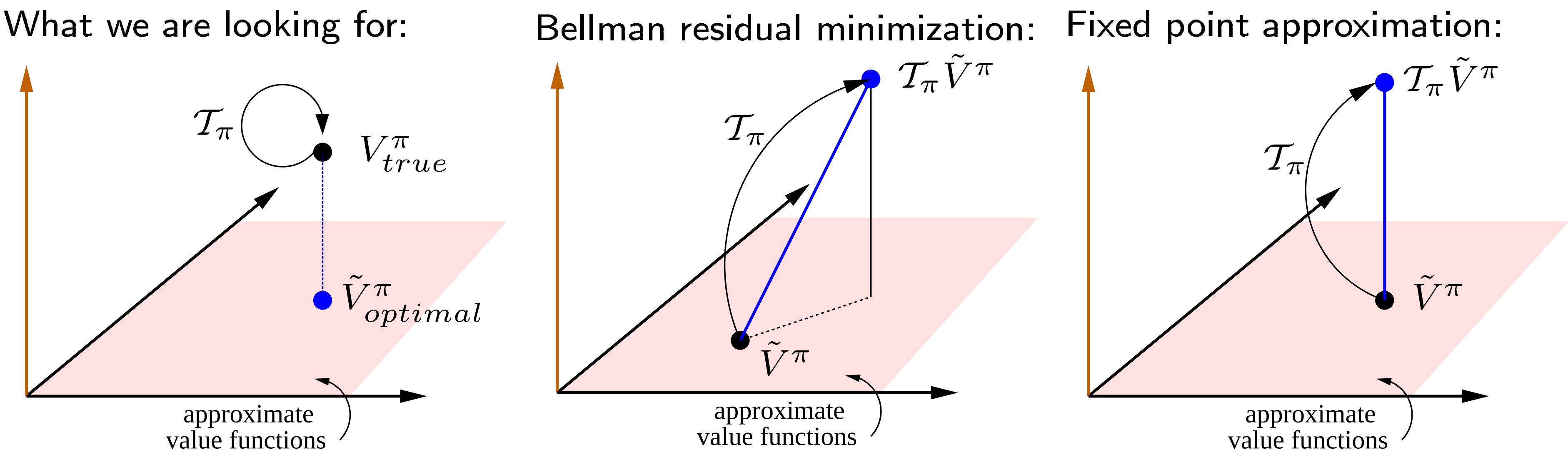
  (For **stochastic transitions** this is not possible $\longrightarrow$ need 'doubled' samples)

- **Fixed-point approximation approach (LSTD):** determine weights $\mathbf{w}$ by solving

$$\mathbf{w} = \operatorname*{argmin}_{\hat{\mathbf{w}}} \left\{ \sum_{i=0}^{t-1} [\boldsymbol{\phi}_m(s_i)]^T \hat{\mathbf{w}} - \left[ r_i + \gamma [\boldsymbol{\phi}_m(s_{i+1})]^T \mathbf{w} \right] \right\}^2 .$$

  (For now we will ignore LSTD(0), but see recent work at end.)

**Least squares in approximate policy evaluation:**




**At a glance:**

- **Bellman residual minimization approach:** $\left\|\hat{V} - \mathcal{T}_\pi \hat{V}\right\|^2 \to \min_{\hat{V}}$
- **Fixed point approximation approach (LSTD):** $V = \operatorname{argmin}_{\hat{V}} \left\|\hat{V} - \mathcal{T}_\pi V\right\|^2$

Next: consider LS-SVMs to solve the Bellman residual minim problem ...

# Recall how LS-SVMs work ... (the general case)

**Recall:** how LS-SVM (regularization networks, kernel ridge regression, Gaussian process regression) are applied to function approximation.

**The short story:** (using **subset of regressors** approximation)

- Given: some $t$ training data $\{\mathbf{x}_i, y_i\}_{i=1}^{t}$
- Choose: a kernel function $k$ that generates RKHS $\mathcal{H}_k$, the function space of possible solutions (e.g. polynomials, Gaussian RBFs, ...)
- Select: a **subset** $\{\tilde{\mathbf{x}}_i\}_{i=1}^{m}$ of the training data, where $m \ll t$
- Represent: the solution by $f(\cdot) = \sum_{i=1}^{m} k(\tilde{\mathbf{x}}_i, \cdot) w_i$
- Solve: the quadratic t-by-m problem to obtain the weights $\mathbf{w}$

$$\boxed{\min_{\mathbf{w} \in \mathbb{R}^m} \; \|\mathbf{K}_{tm}\mathbf{w} - \mathbf{y}\|^2 + \lambda \mathbf{w}^T \mathbf{K}_{mm} \mathbf{w}}$$

| | | |
|---|---|---|
| where | $[\mathbf{K}_{tm}]_{ij} = k(\mathbf{x}_i, \tilde{\mathbf{x}}_j)$ | a $t \times m$ matrix |
| | $[\mathbf{K}_{mm}]_{ij} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$ | a $m \times m$ matrix |
| | $\lambda$ | a regularization parameter |

Now: application to Bellman residuals ...

# Bellman with LS-SVM

## Application to Bellman residuals:

- Training data: observed trajectory $s_0, s_1, s_2, \ldots, s_t$ along with rewards $r_1, r_2, \ldots, r_t$
- Represent value function by $\boxed{V^\pi(\cdot) = \sum_{i=1}^m k(\tilde{s}_i, \cdot) w_i}$ where $\{\tilde{s}_i\}_{i=1}^m$ is a subset
- Solve the quadratic t-by-m problem corresponding to Bellman

$$\boxed{\min_{\mathbf{w} \in \mathbb{R}^m} \ \|\mathbf{H}_{tm}\mathbf{w} - \mathbf{r}\|^2 + \lambda \mathbf{w}^T \mathbf{K}_{mm} \mathbf{w}}$$

where

$$\mathbf{k}_m(\cdot) = \begin{bmatrix} k(\tilde{s}_1, \cdot) \\ \vdots \\ k(\tilde{s}_m, \cdot) \end{bmatrix}, \quad \mathbf{H}_{tm} = \begin{bmatrix} [\mathbf{k}_m(s_0) - \gamma \mathbf{k}_m(s_1)]^T \\ \vdots \\ [\mathbf{k}_m(s_{t-1}) - \gamma \mathbf{k}_m(s_t)]^T \end{bmatrix}, \quad \mathbf{r} = \begin{bmatrix} r_1 \\ \vdots \\ r_t \end{bmatrix}$$

- Obtain generalized normal equations: $\boxed{\mathbf{w} = \left(\mathbf{H}_{tm}^T \mathbf{H}_{tm} + \lambda \mathbf{K}_{mm}\right)^{-1} \mathbf{H}_{tm}^T \mathbf{r}}$

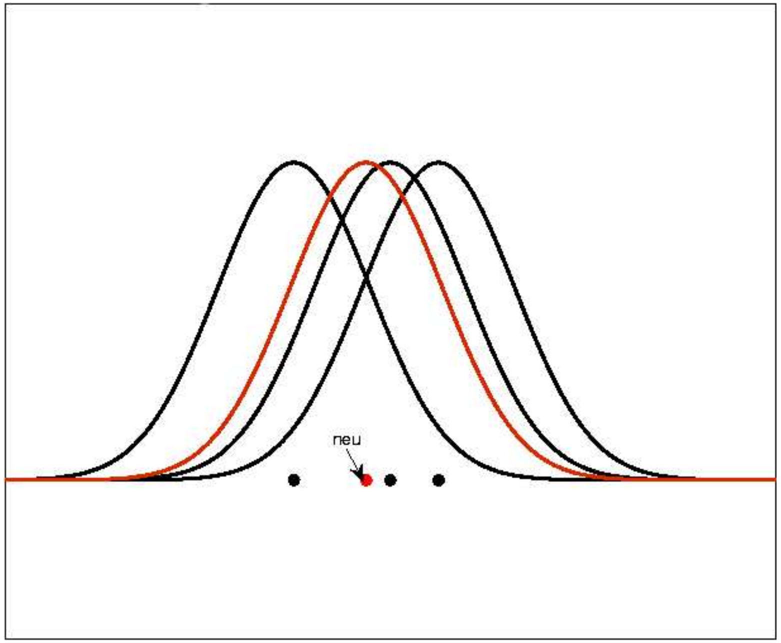
Coming up next: how to select the relevant subset $\{\tilde{s}_i\}_{i=1}^m$ ...

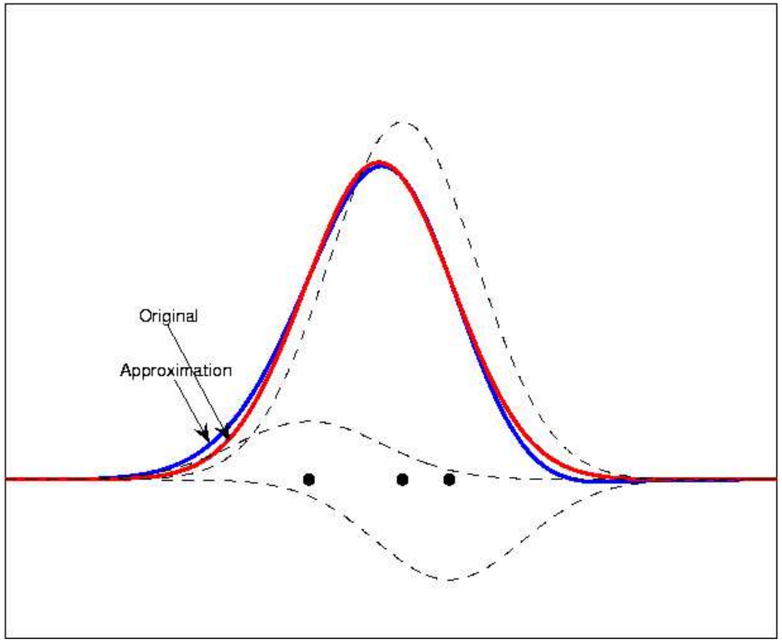
# Sparse greedy online approximation

proposed by Csato & Opper (2002), Engel et al. (2003):

**Online selection:** assume data becomes available **sequentially** at $t = 1, 2, \ldots$

- Start with an empty subset ('dictionary' of basis functions)
- At time $t$ try to approximate the new data $s_t$ from the current dictionary:




- Criterion: if $k(s_t, s_t) - [\mathbf{k}_m(s_t)]^T \mathbf{K}_{mm}^{-1} \mathbf{k}_m(s_t) > \texttt{TOL}$ then $s_t$ is added to subset
- Overall costs: $\mathcal{O}(m^2)$, where $m$ is the current size of subset

Now: putting everything together ...

# Putting it together ...

**A recursive implementation:** at time t observe $s_{t-1} \longrightarrow s_t$ under reward $r_t$

- **Goal:** update $\mathbf{w}_{tm} = \left[ \begin{bmatrix} \mathbf{H}_{t-1,m} \\ \mathbf{h}_t{}^T \end{bmatrix}^T \begin{bmatrix} \mathbf{H}_{t-1,m} \\ \mathbf{h}_t{}^T \end{bmatrix} + \lambda \mathbf{K}_{mm} \right]^{-1} \begin{bmatrix} \mathbf{H}_{t-1,m} \\ \mathbf{h}_t{}^T \end{bmatrix}^T \begin{bmatrix} \mathbf{r}_{t-1} \\ r_t \end{bmatrix}$
- **Definition:** $\mathbf{P}_{t-1,m} = (\mathbf{H}_{t-1,m}^T \mathbf{H}_{t-1,m} + \lambda \mathbf{K}_{mm})^{-1}$, $\mathbf{s}_{t-1,m} = \mathbf{H}_{t-1,m}^T \mathbf{r}_{t-1}$
- **Rank-1 updates** via:
  - Route 1: Formula of Sherman-Morrison-Woodbury ('recursive least squares')
  - Route 2: Update of Cholesky factorization: $\mathbf{P}_{t-1,m} = \mathbf{\Phi}_{t-1,m}^{1/2} \mathbf{\Phi}_{t-1,m}^{T/2}$

**Case 1:** Current example is **represented well** by current dictionary

- Update $\{\mathbf{P}_{t-1,m}, \mathbf{s}_{t-1,m}, \mathbf{w}_{t-1,m}\} \longrightarrow \{\mathbf{P}_{tm}, \mathbf{s}_{tm}, \mathbf{w}_{tm}\}$
- Costs $\mathcal{O}(m^2)$

**Case 2:** Current example is **not represented well** by current dictionary

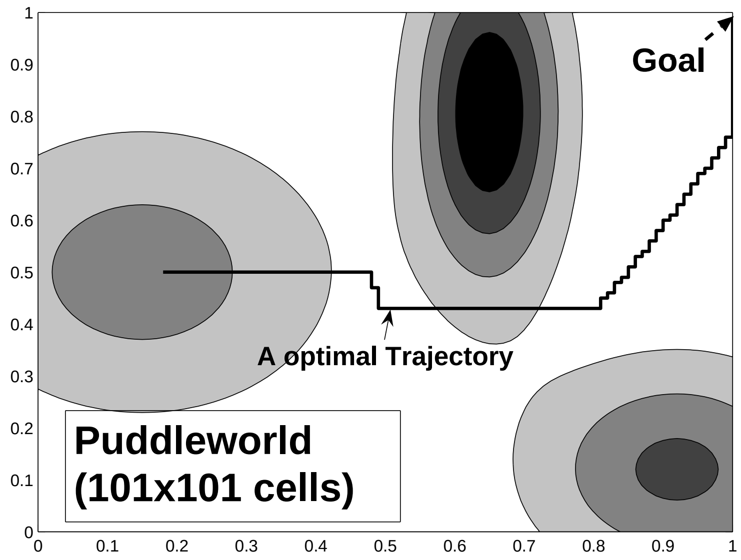
- Add $s_t$ as new basis function to the dictionary
- Update $\{\mathbf{P}_{t-1,m}, \mathbf{s}_{t-1,m}, \mathbf{w}_{t-1,m}\} \longrightarrow \{\mathbf{P}_{t,m+1}, \mathbf{s}_{t,m+1}, \mathbf{w}_{t,m+1}\}$
- Costs $\mathcal{O}(m^2)$

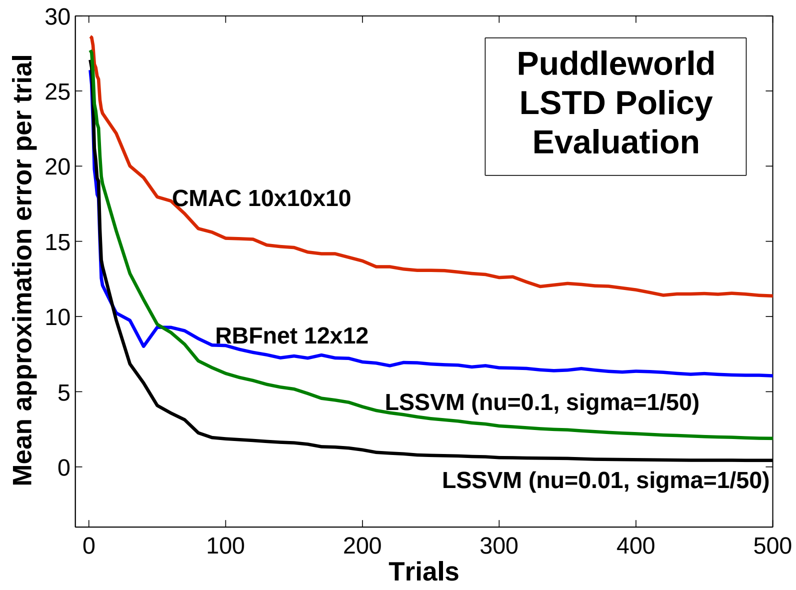
Finally: some toy examples ...

# Example: Policy Evaluation

**Task:** policy-evaluation for a data set of 50,000 observed transitions under an optimal policy

**Comparing:** Tilecoding/CMAC(10x10x10) vs. our approach (RBF-kernel) vs. fixed RBF-net(12x12)



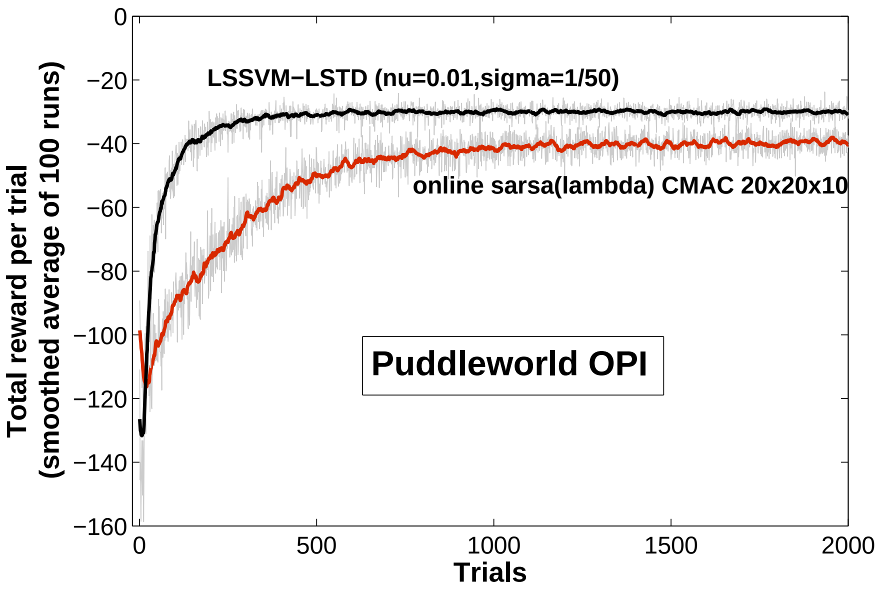

**Required resources:** CMAC (1000 weights), fixed RBF-net (144 weights), our approach (122/202 weights)

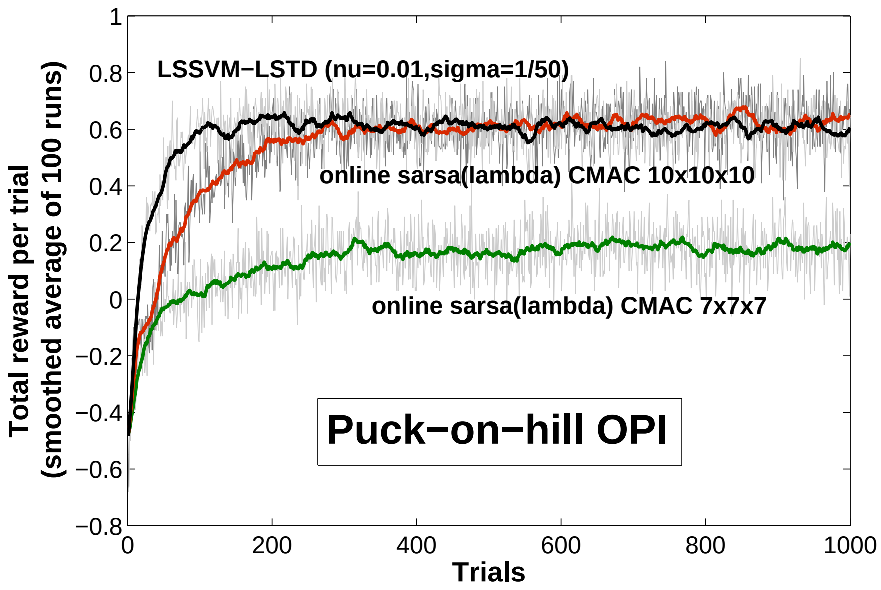
What about optimal control? ⟶ Optimistic policy iteration ...

# Example: Optimistic Policy Iteration

**Episodic tasks:** **puddleworld** and **puck-on-hill**

**Comparing:** textbook sarsa($\lambda$) + tilecoding vs. our approach




Now for some recent results with a 'real-world' problem (not in paper)

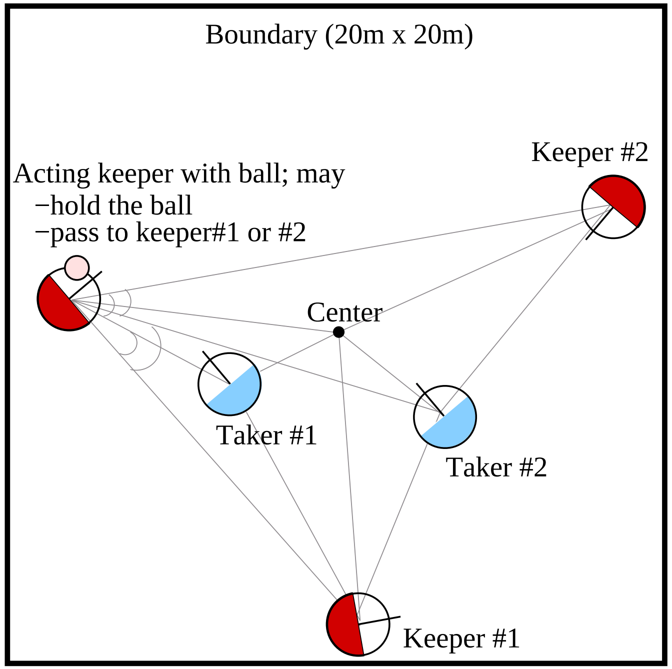
# Recent work

**Current work and improvements:**

- Allow **stochastic transitions**: fixed-point approximation LSTD($\lambda$) instead of Bellman residuals
- Allow **model-free learning**: consider augmented state-action values (Q-function instead of V-function)
- Consider a **supervised** criterion during basis selection $\Longrightarrow$ reduces size of dictionary by 10%–40%

# RoboCup-Keepaway (Stone et al. 2005)

**Goal:** **learn** how to maximize the time the keepers control the ball (reinforcement learning)

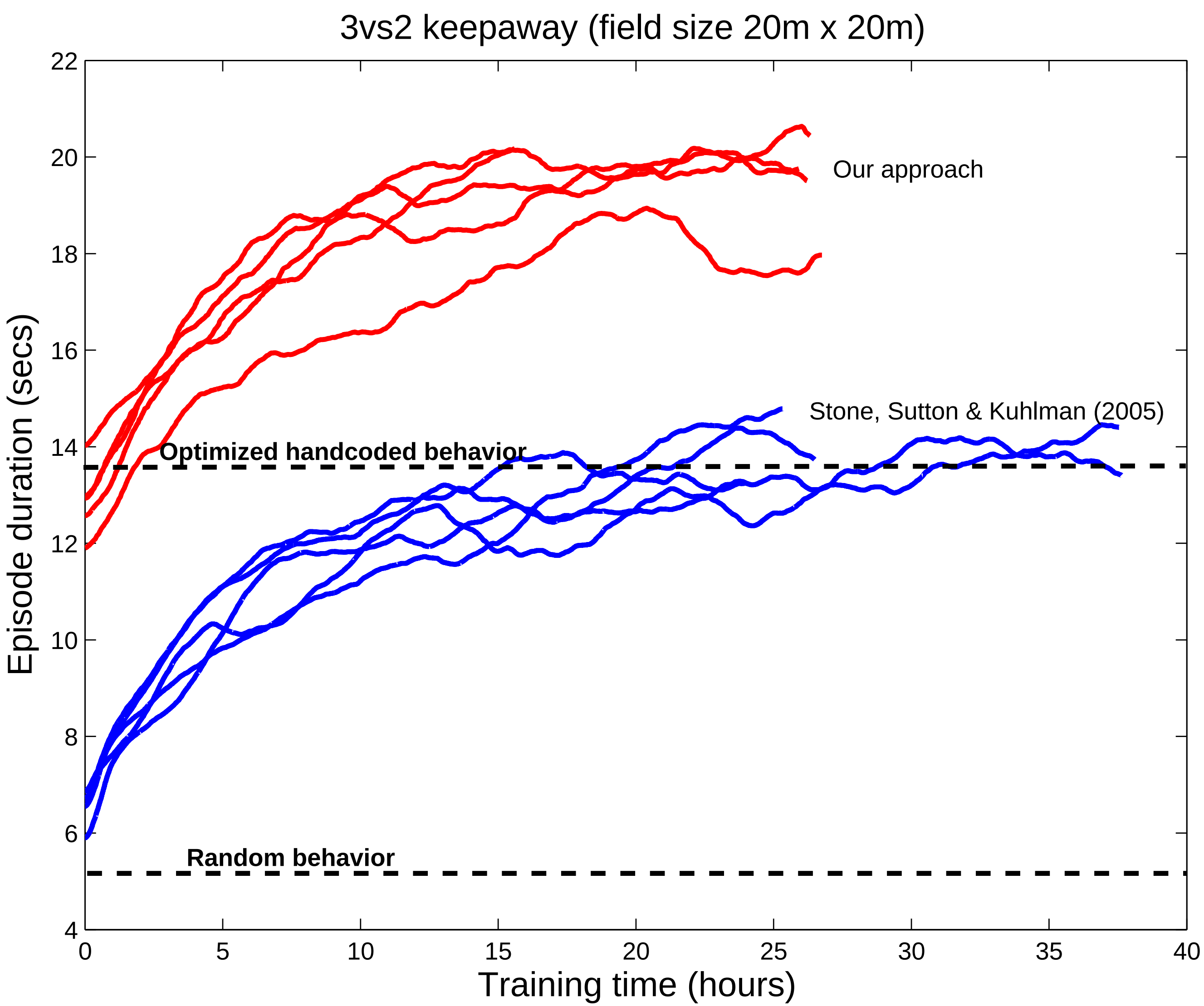



**Challenges:**

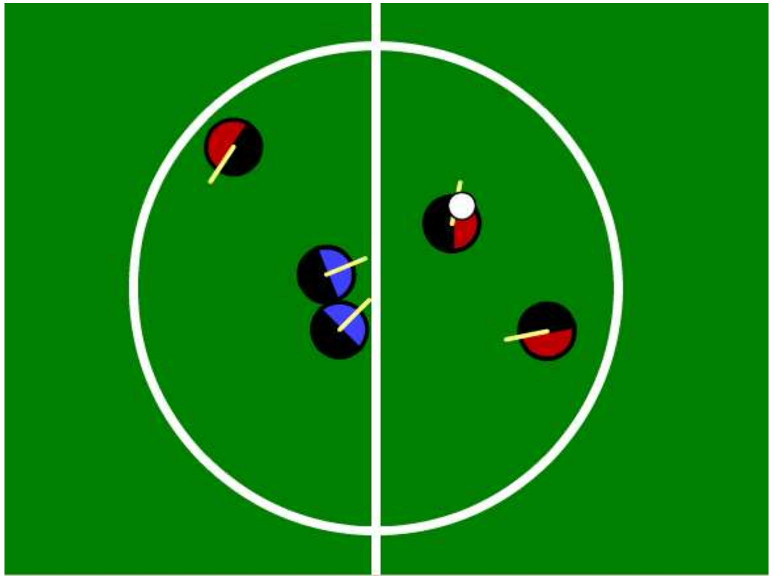
- **dimensionality** of the state space (13 dimensions)
- **stochastic transitions** (noisy perceptions and actions, multiple fully autonomous agents need to cooperate)
- **real-time learning** (uses 'official' soccer server)

# Results: on-line learning

**Compare:** textbook approach sarsa($\lambda$) + tilecoding vs. our approach



## Demonstration?



Show flash videos: random behavior (5 secs average) versus learned behavior (20 secs average) ??

# Summary

**Talked about:**

- **Topic:** approximate dynamic programming
- **Idea:** combine LS-SVM (superior generalization) with LS-TD (very fast convergence)
- **Methods:**
  - approximate policy evaluation with least squares methods
    - deterministic + stochastic
    - model-based + model-free
  - LS-SVM + subset of regressors
  - online selection of subset (unsupervised + supervised)
- **Results:**
  - beats standard tilecoding in LSTD and OPI (for toy problems)
  - dramatical improvements in 3vs2 keepaway