Sequential Learning with LS-SVM for Large-Scale Data Sets

Tobias Jung, University of Mainz, Germany

Daniel Polani, University of Hertfordshire, U.K.

Motivation: Online learning sometimes useful, e.g. for

1. Large-scale data sets
2. Time series prediction
3. Reinforcement learning

Our goal: Efficient online learning with regularization networks

At a glance

Will talk about:

- Regularization networks (e.g. GP regression, Kernel ridge regression, LS-SVM)
- Subset of regressors approximation
- Sparse greedy online selection (supervised)
- Efficient implementation (like recursive least squares + a 'growing' operation)
- Experimental evaluation + recent application to RL

Contribution: Exploiting sparse online approximation we can

- Add a new basis function in $\mathcal{O}(m^2)$ time [normal would be $\mathcal{O}(tm)$]
- Compute the contribution (reduction of error) of one basis function candidate during greedy forward selection in $\mathcal{O}(m^2)$ time [normal would be $\mathcal{O}(tm)$]
- Compute predictive variance with augmentation in $\mathcal{O}(m^2)$ time [normal would be $\mathcal{O}(tm)$]

Regularization networks

(also appear as Kernel Ridge Regression, Gaussian Process Regression, LS-SVM, ...)

Objective:

- **Given:** training data $\{\mathbf{x}_i, y_i\}_{i=1}^t$, inputs $\mathbf{x}_i \in \mathbb{R}^d$, outputs $y_i \in \mathbb{R}$
- **Goal:** reconstruct underlying function f from $\mathcal{H}_k$ (RKHS) by solving

$$\min_{f \in \mathcal{H}_k} \sum_{i=1}^{t} (y_i - f(\mathbf{x}_i))^2 + \lambda \|f\|_{\mathcal{H}_k}^2$$

Solution:

- **Representer theorem:** may assume $f(\cdot) = \sum_i k(\mathbf{x}_i, \cdot) w_i$. Thus solve t-**by-**t problem

$$\min_{w \in \mathbb{R}^t} \|\mathbf{K}\mathbf{w} - \mathbf{y}\|^2 + \lambda \mathbf{w}^T \mathbf{K} \mathbf{w}$$

- **Solution:**

$$\mathbf{w} = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y}$$

where $k(\cdot, \cdot)$ is the kernel function (e.g. polynomials, Gaussian RBF,...), $\mathbf{K}$ is the $t \times t$ kernel matrix $[\mathbf{K}]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ and λ is the regularization parameter.

Caveat: solution costs $\mathcal{O}(t^3)$. Impractical for intermediate to large data sets ...

Subset of regressors approximation

(proposed by Girosi (1990), Wahba (1990),...)

Idea:

- **Choose:** a subset of the data $\{\tilde{\mathbf{x}}\}_{i=1}^{m}$, where $m \ll t$
- **Approximate:** the kernel by (also arises from the Nyström approximation)

$$k(\mathbf{x}, \mathbf{x}') = [\mathbf{k}_m(\mathbf{x})]^T \mathbf{K}_{mm}^{-1} \mathbf{k}_m(\mathbf{x}') \qquad \forall \mathbf{x}, \mathbf{x}'$$

where $\mathbf{k}_m(\cdot) = \big(k(\tilde{\mathbf{x}}_1, \cdot), \ldots, k(\tilde{\mathbf{x}}_m, \cdot)\big)^T$ and $[\mathbf{K}_{mm}]_{ij} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$.

Reduced problem:

- **Represent:** using only the subset: $f(\cdot) = \sum_i^m k(\tilde{\mathbf{x}}_i, \cdot) w_i$
- **Quadratic problem:** solve the m-by-m problem

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

- **Solution:**

$$\mathbf{w} = \big(\mathbf{K}_{tm}^T \mathbf{K}_{tm} + \lambda \mathbf{K}_{mm}\big)^{-1} \mathbf{K}_{tm}^T \mathbf{y}$$

where $[\mathbf{K}_{tm}]_{ij} = k(\mathbf{x}_i, \tilde{\mathbf{x}}_j)$. This costs only $\mathcal{O}(tm^2)$ operations (instead of $\mathcal{O}(t^3)$).

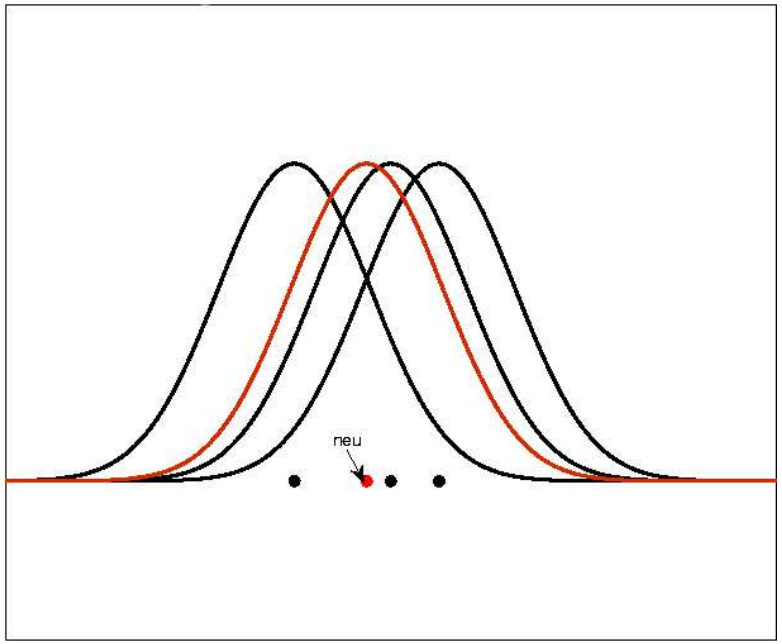
Coming up next: how to select the subset

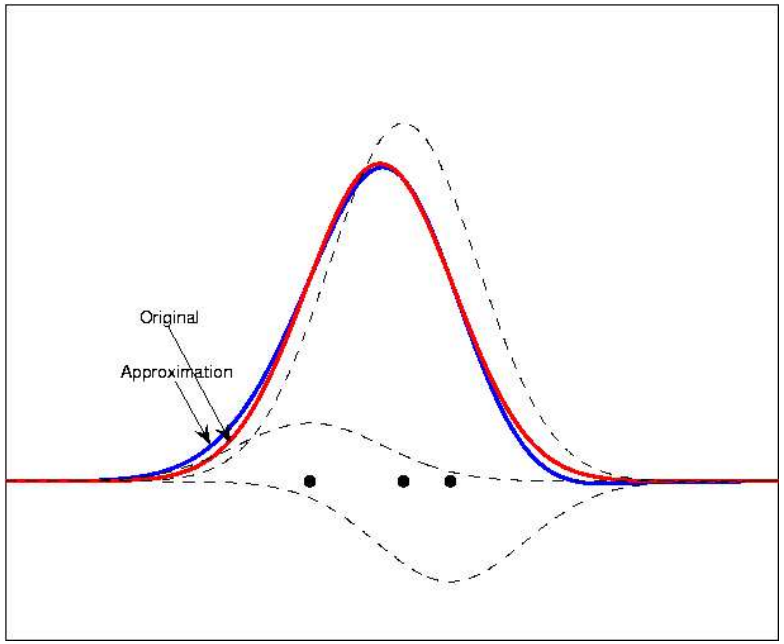
Sparse greedy online approximation

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

Online selection: assume training 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 input data $\mathbf{x}_t$ from the current dictionary:





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

Now: what do we gain by doing this?

Effect of online approximation

Note: **online** means that every time step only the **current** elements in the dictionary are used. Future elements do not retroactively contribute to the approximations in the past!

Effect: whenever we would need to consider all past examples, e.g. in a $t \times m$ model

- when adding a new basis function
- when computing the score of basis function candidates in greedy forward selection
- when computing predictive variance in SR with 'augmentation' (Rasmussen & Q-C., 2005)

we can exploit that adding a new basis function centered on some $\mathbf{x}^*$, that is

$$\mathbf{K}_{t,m+1} = \begin{bmatrix} \mathbf{K}_{tm} & \mathbf{q} \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{t-1,m} & \mathbf{K}_{t-1,m}\mathbf{a}^* \\ \mathbf{k}_m(\mathbf{x}_t)^T & k(\mathbf{x}_t, \mathbf{x}^*) \end{bmatrix},$$

where $\mathbf{a}^* = \mathbf{K}_{mm}^{-1}\mathbf{k}_m(\mathbf{x}^*)$, only costs $\mathcal{O}(m^2)$ operations instead of the usual $\mathcal{O}(tm)$.

Now we can piece together an efficient online implementation...

Efficient online implementation

Outline:

- **Solve:** $\min_{\mathbf{w}} J_{tm}(\mathbf{w}) = \|\mathbf{K}_{tm}\mathbf{w} - \mathbf{y}_t\|^2 + \lambda \mathbf{w}^T \mathbf{K}_{mm}\mathbf{w}$
- **Normal equations:** $\mathbf{w}_{tm} = \left(\mathbf{K}_{tm}^T \mathbf{K}_{tm} + \lambda \mathbf{K}_{mm}\right)^{-1} \mathbf{K}_{tm}^T \mathbf{y}_t$
- **Abbreviations:**
  - Cross-product matrix $\mathbf{P}_{tm} := \left(\mathbf{K}_{tm}^T \mathbf{K}_{tm} + \lambda \mathbf{K}_{mm}\right)$
  - Regularized costs $\xi_{tm} := J_{tm}(\mathbf{w}_{tm})$
- **Goal:** update $\{\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm}\}$ when new data arrives

Sequential learning: at time $t+1$ (with m basis functions currently selected into the dictionary)

- Observe the new input-output pair $(\mathbf{x}_{t+1}, y_{t+1})$
- Propagate $\{\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm}\}$ forward in time by the operations
  - **Normal step:** $\{\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm}\} \longrightarrow \{\mathbf{P}_{t+1,m}^{-1}, \mathbf{w}_{t+1,m}, \xi_{t+1,m}\}$
  - **Growing step:** $\{\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm}\} \longrightarrow \{\mathbf{P}_{t,m+1}^{-1}, \mathbf{w}_{t,m+1}, \xi_{t,m+1}\}$
  - **Pruning step:** $\{\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm}\} \longrightarrow \{\mathbf{P}_{t,m\setminus i}^{-1}, \mathbf{w}_{t,m\setminus i}, \xi_{t,m\setminus i}\}$

Normal step: $\{\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm}\} \longrightarrow \{\mathbf{P}_{t+1,m}^{-1}, \mathbf{w}_{t+1,m}, \xi_{t+1,m}\}$

Easy: just the usual recursive least squares update: $\mathbf{P}_{t+1,m} = \mathbf{P}_{tm} + \mathbf{k}_{t+1}\mathbf{k}_{t+1}^T$, thus

$$\mathbf{P}_{t+1,m}^{-1} = \mathbf{P}_{tm}^{-1} - \frac{\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}\mathbf{k}_{t+1}^T\mathbf{P}_{tm}^{-1}}{\Delta}, \qquad \mathbf{w}_{t+1,m} = \mathbf{w}_{tm} + \frac{\varrho}{\Delta}\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}$$

$$\xi_{t+1,m} = \xi_{tm} + \frac{\varrho^2}{\Delta}$$

where

- $\mathbf{k}_{t+1} := \mathbf{k}_m(\mathbf{x}_{t+1})$ ($m \times 1$ vector)
- $\Delta = 1 + \mathbf{k}_{t+1}^T\mathbf{P}_{tm}^{-1}\mathbf{k}_{t+1}$ (scalar)
- $\varrho = y_{t+1} - \mathbf{k}_{t+1}^T\mathbf{w}_{tm}$ (scalar)

Operation count is $\mathcal{O}(m^2)$.

Growing step: $\{\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm}\} \longrightarrow \{\mathbf{P}_{t,m+1}^{-1}, \mathbf{w}_{t,m+1}, \xi_{t,m+1}\}$

How to add a basis function: centered on $\mathbf{x}_{t+1}$

$$\mathbf{P}_{t,m+1}^{-1} = \begin{bmatrix} \mathbf{P}_{tm}^{-1} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} + \frac{1}{\Delta_b} \begin{bmatrix} -\mathbf{w}_b \\ 1 \end{bmatrix} \begin{bmatrix} -\mathbf{w}_b \\ 1 \end{bmatrix}^T$$

where

- $\mathbf{w}_b = \mathbf{a}_{t+1} + \frac{\delta}{\Delta} \mathbf{P}_{t-1,m}^{-1} \mathbf{k}_{t+1}, \qquad \Delta_b = \frac{\delta^2}{\Delta} + \lambda\delta$
- $\delta = k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) - \mathbf{k}_{t+1}^T \mathbf{a}_{t+1}$

can be obtained in $\mathcal{O}(m)$ operations (caching and reusing terms).

(Remaining updates of $\mathbf{w}_{t,m+1}, \xi_{t,m+1}$ are described in our paper...)

When to add a basis function: Two-part criterion (supervised)

- **Novelty:** $\delta = k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) - \mathbf{k}_{t+1}^T \mathbf{a}_{t+1}$ (i.e. the reconstruction error from sparse greedy approximation)
- **Usefulness:** $\delta^2 \varrho^2 / (\Delta_b \Delta^2)$ (reduction of costs when we would add it to the basis)

Operation count is $\mathcal{O}(m^2)$.

Pruning step: $\{\mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm}\} \longrightarrow \{\mathbf{P}_{t,m\backslash i}^{-1}, \mathbf{w}_{t,m\backslash i}, \xi_{t,m\backslash i}\}$

How to delete the i-th basis function: swap columns/rows i and m. Then delete the m-th one:

$$\begin{bmatrix} \mathbf{P}_{t,m-1}^{-1} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix} = \mathbf{P}_{tm}^{-1} - \frac{1}{\mathbf{P}_{tm}^{-1}(m,m)} \begin{bmatrix} \mathbf{P}_{tm}^{-1}(1:m-1,m) \\ -1 \end{bmatrix} \begin{bmatrix} \mathbf{P}_{tm}^{-1}(1:m-1,m) \\ -1 \end{bmatrix}^T$$

(Remaining updates of $\mathbf{w}_{t,m\backslash i}, \xi_{t,m\backslash i}$ are described in our paper...)

When to delete a basis function: Compute the score (i.e. the increase of costs when basis function i would be removed)

$$\varepsilon_i = \frac{\mathbf{w}_{tm}(i)^2}{\mathbf{P}_{tm}^{-1}(i,i)} \qquad i = 1, \ldots, m$$

for every basis function i and delete the one with the smallest score. The computation of this criterion is very cheap.

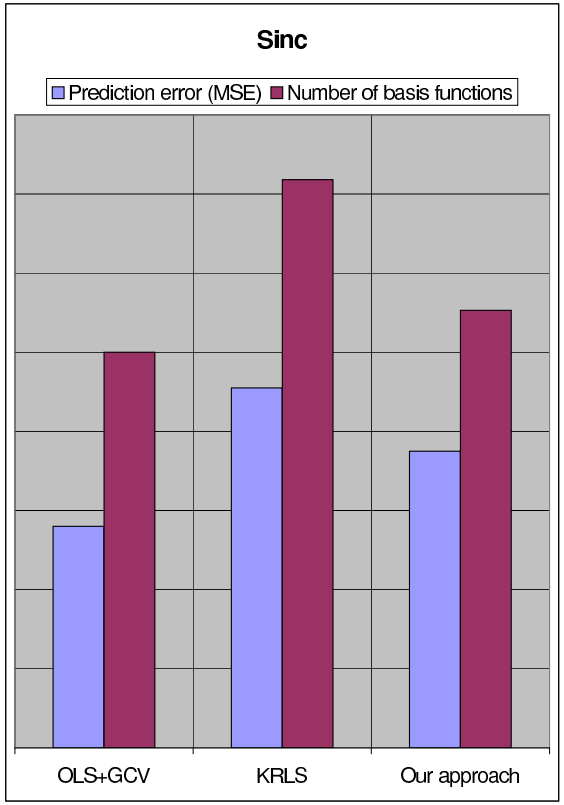
Operation count is $\mathcal{O}(m^2)$.

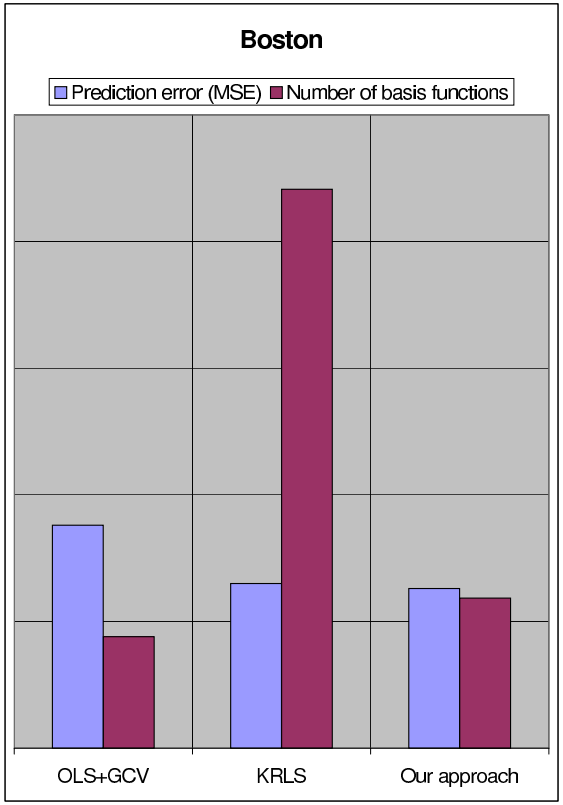
Experiments I

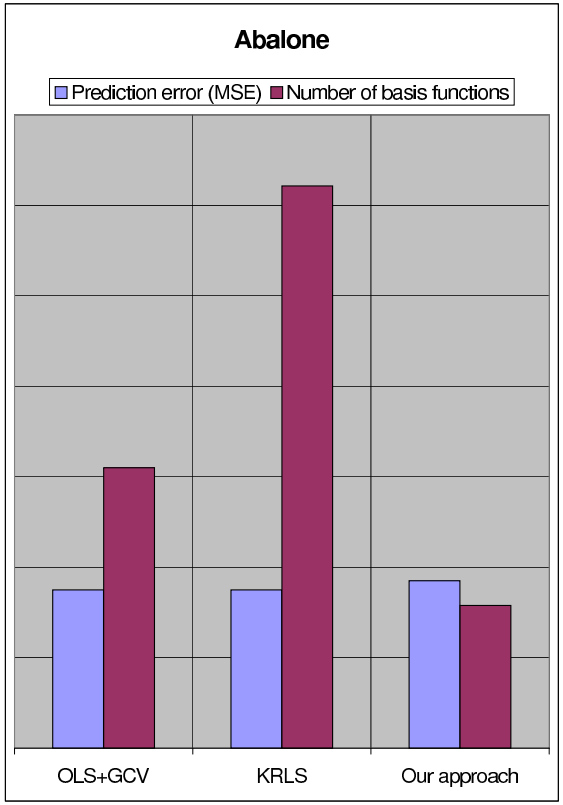
Compare subset selection criteria: Prediction performance vs. number of selected basis functions

Candidates:

- Orthogonal least squares + kernel based dictionary + GCV stopping criterion
- Kernel recursive least squares with unsupervised basis selection (Engel et al. 2004)
- Our approach with supervised basis selection

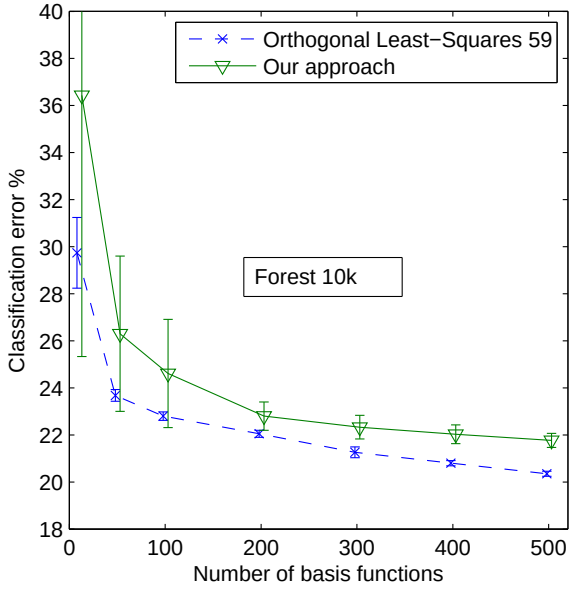


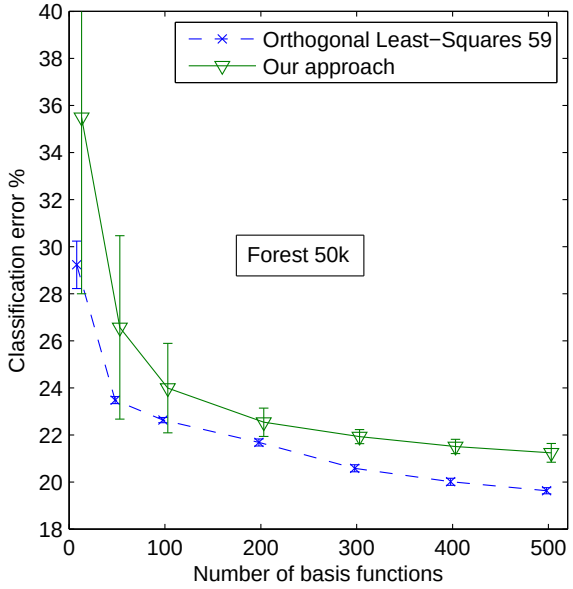


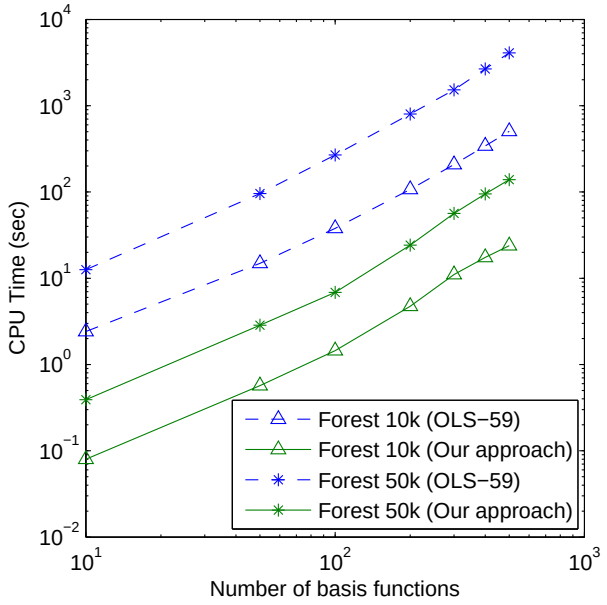


Experiments II: Forest data set

Large-scale real world benchmark: Our approach vs. OLS-59



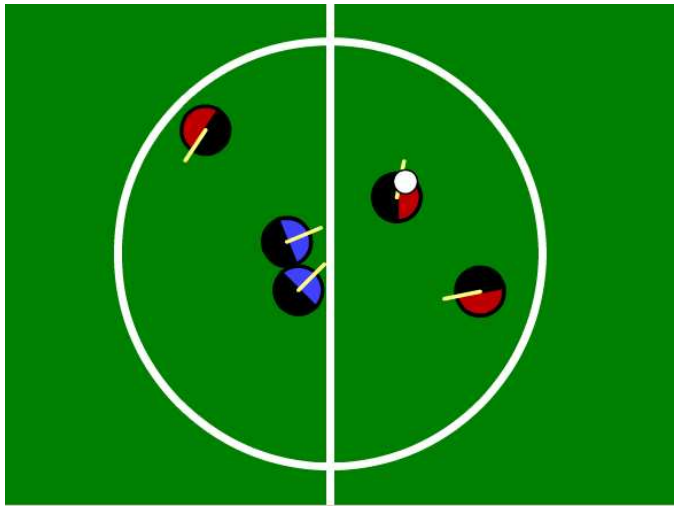




Recent work: application to approximate policy evaluation in **reinforcement learning...**

RoboCup-Keepaway (Stone et al. 2005)

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

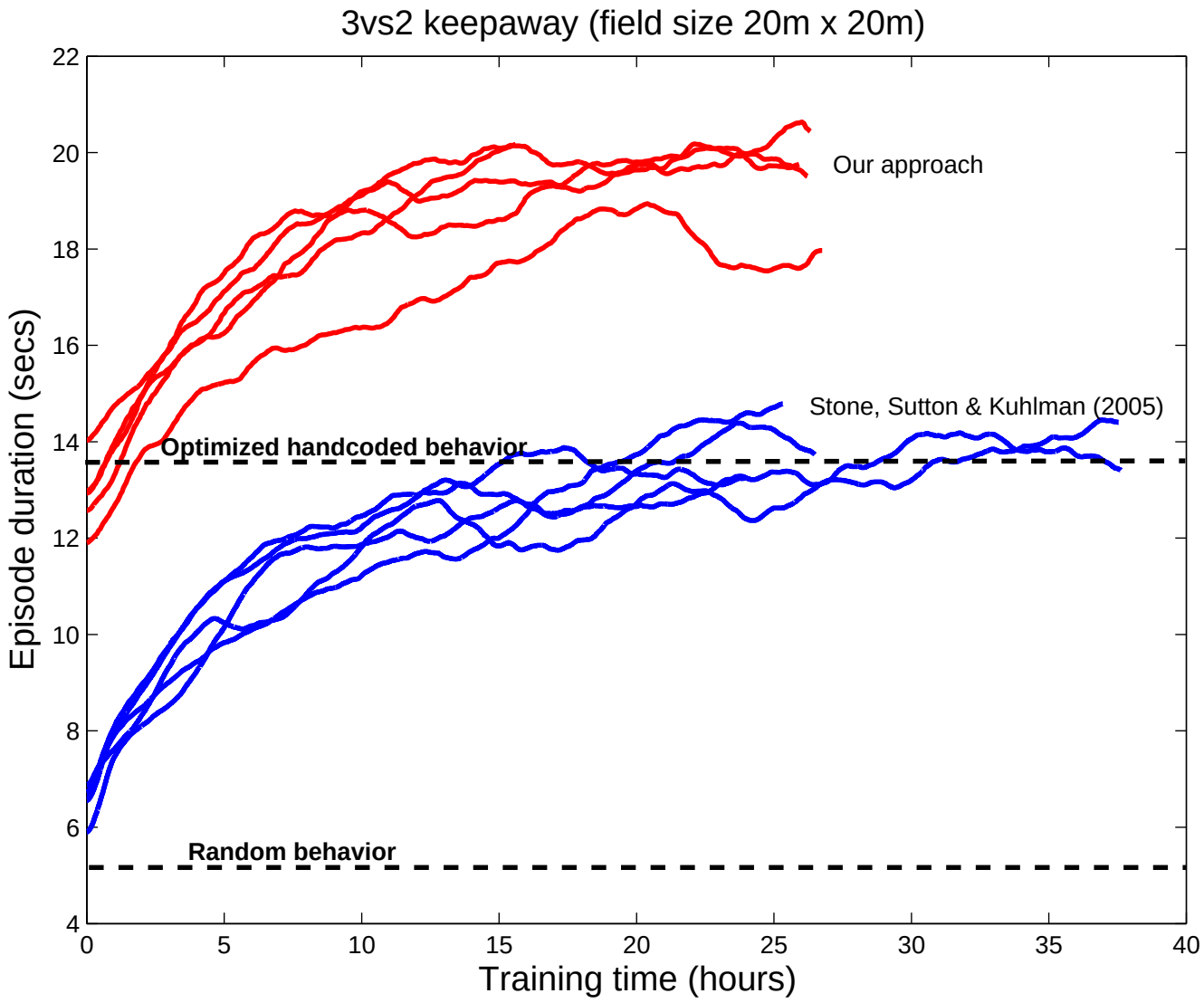


Challenges:

- **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: learning optimal control

Compare: Our approach vs. the textbook approach sarsa(λ) + tilecoding



Summary

Talked about:

- **Topic:** Online learning with regularization networks
- **Methods:**
  - Subset of regressors approximation
  - Online greedy selection of relevant basis functions (supervised)
  - Efficient recursive implementation: $\mathcal{O}(m^2)$ per step (independent of the total number of data)
- **Results:**
  - Performance is at least as good as the related KRLS algorithm ...
  - ... but chooses a (sometimes much) smaller subset of basis functions
- **Applications:**
  - Large-scale data sets
  - Approximate policy evaluation in real-time reinforcement learning