Kernelizing LSPE(λ)

Tobias Jung, University of Mainz, Germany

Daniel Polani, University of Hertfordshire, U.K.

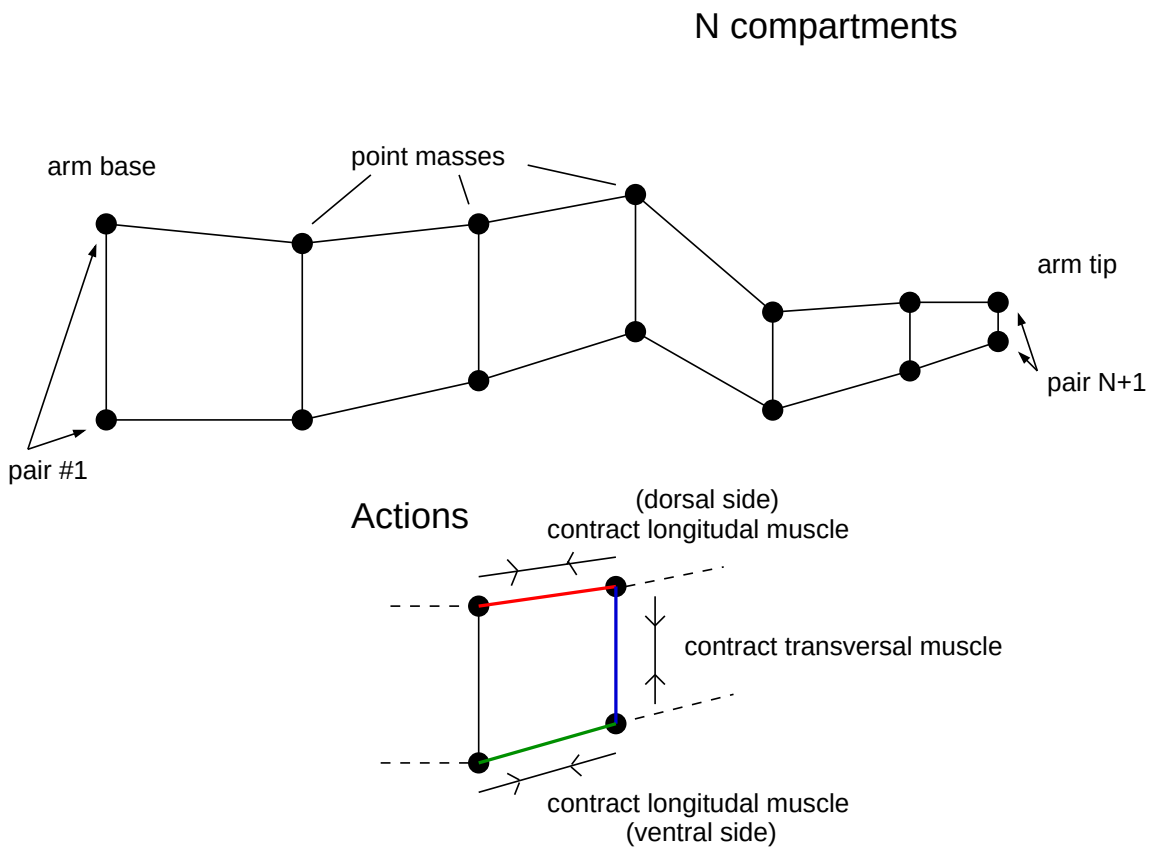
Motto: APE/API with kernel-based function approximation to tackle high-dimensional control tasks heretofore impossible to solve with traditional RL approaches (e.g. Sarsa+Tilecoding)

High-dimensional control tasks such as ...

Octopus (Engel et al. 2005)

(ICML06 RL benchmarking page: http://www.cs.mcgill.ca/dprecup/workshops/ICML06/octopus.html)

Goal: **learning** to control an octopus arm (reaching-task)



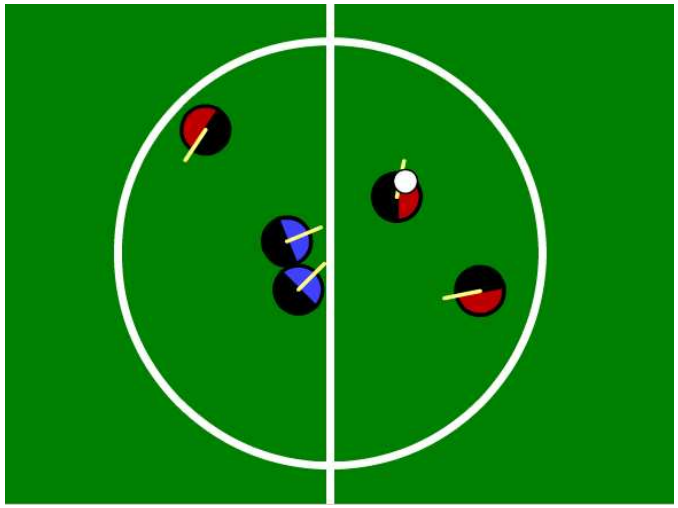
Challenges:

- **high-dimensional** state space $\subset \mathbb{R}^{(2N+2)\times 4}$ (e.g. N=8 compartments $\Longrightarrow \mathbb{R}^{72}$)
- **high-dimensional** and **continous action space** $\subset \mathbb{R}^{N\times 3}$
- ... here: discretized into 7 activation patterns

RoboCup-Keepaway (Stone et al. 2005)

(Available from: `http://www.cs.utexas.edu/users/AustinVilla/sim/keepaway/`)

Goal: **learn** how to maximize the time the keepers control the ball (3 vs. 2)



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)

Contents of this talk

Our wish list:

- high dimensions
- sequential/online learning
- fast, efficient, reliable
- linearly parameterized value function ($\Rightarrow$ LSTD/LSPE policy evaluation)

Our approach:

- API/OPI framework
- APE through LSPE/LSTD with kernel-based function approximation
- Regularization networks/kernel ridge regression/Gaussian process regression ...
- To counter the $\mathcal{O}(n^3)$ scaling of kernel-based learning
  - subset of regressors approximation
  - online selection of relevant basis elements

Result: An efficient, recursive online implementation with automatic supervised selection of relevant basis functions and per-step complexity $\mathcal{O}(m^2)$.

Basics

Policy evaluation: (Model-free!)

$$Q^\pi(s,a) = E_{s'|s,a}\{R(s'|s,a) + \gamma Q^\pi(s', \pi(s'))\}$$

where

- $\mathcal{S} \subset \mathbb{R}^d$ state space, $\mathcal{A}$ action space (discrete)
- $\pi : \mathcal{S} \to \mathcal{A}$ policy
- $P(s'|s,a)$ tranisition probabilities
- $R(s'|s,a)$ reward (1-step return)
- $\gamma \in (0,1)$ discount factor

Approximate policy evaluation

- **1. Function approximation:** represent Q^π through linearly parameterized approximation

$$\tilde{Q}(\mathbf{x};\mathbf{w}) = \sum_{i=1}^{m} w_i \varphi_i(\mathbf{x})$$

where

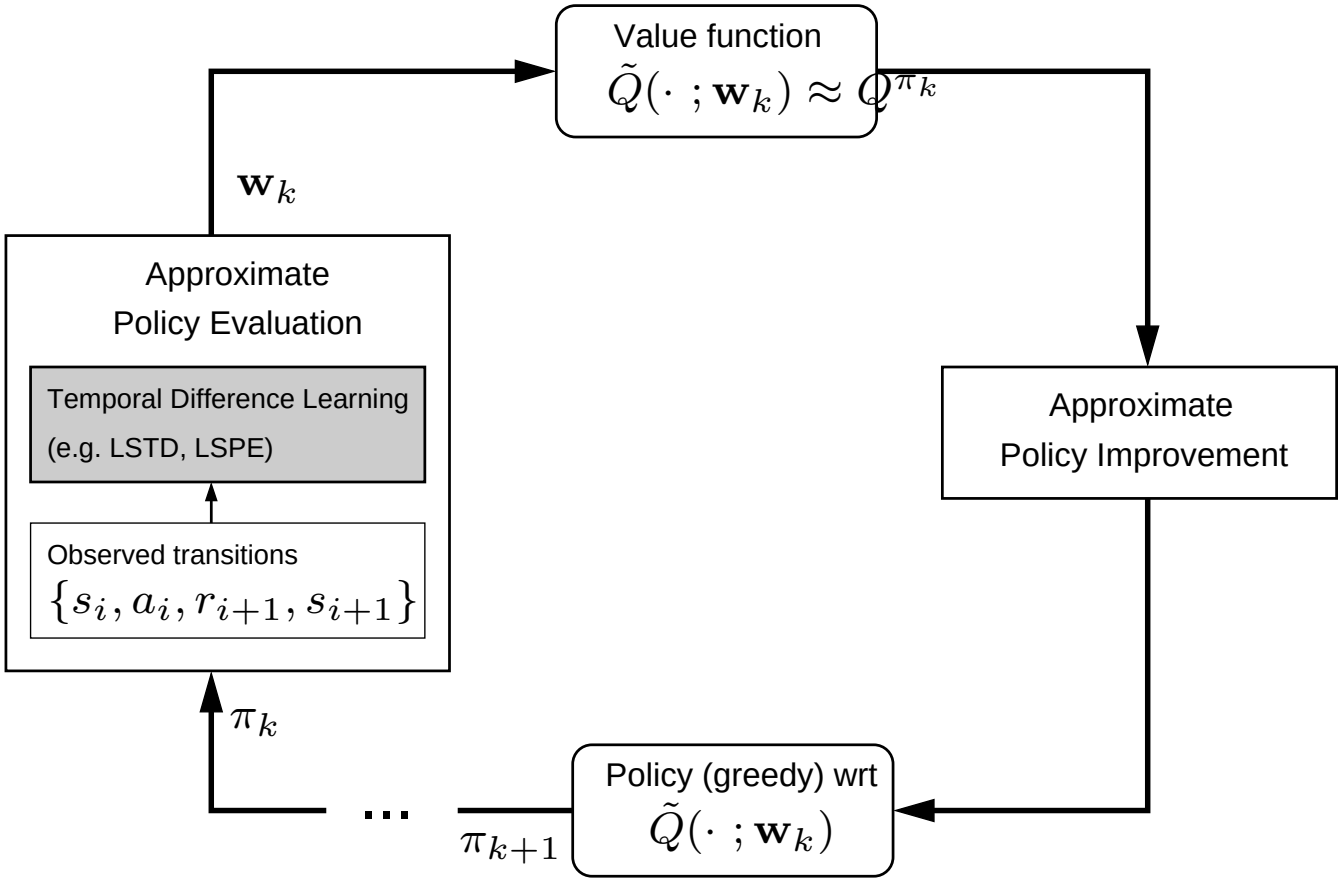
  - $\mathbf{x} := (s,a) \in \mathbb{R}^d \times \mathcal{A}$ staten action tuples
  - $\varphi_i : \mathbb{R}^d \times \mathcal{A} \to \mathbb{R}$ basis functions $(i = 1,\ldots,m)$
  - $\mathbf{w} = \left(w_1,\ldots,w_m\right)^\mathsf{T}$ weights (need to be determined)

- **2. Samples:** approximate $P(\cdot\,|s,a), R(\cdot\,|s,a)$ from sample transitions (simulation)

$$s_t \xrightarrow[r_{t+1}]{\pi_k(s_t)} s_{t+1} \xrightarrow[r_{t+2}]{\pi_k(s_{t+1})} s_{t+2} \longrightarrow \cdots$$

Of course, we're not just interested in merely obtaining Q^π ...

Approximate policy iteration



At the end: everything boils down to the questions

1. By what method do we choose the parametrization of $\tilde{Q}$ and carry out regression?
2. By what method do we learn the weight vector $\mathbf{w}$, given sample transitions?

Motivation

1. **Why kernel-based function approximation?**
   - Avoids curse of dimensionality by representing solution through the data
   - Eliminates explicit node/basis function selection (at least in theory)
   - Instead, just choose kernel/covariance function. E.g. the Gaussian

$$k(\mathbf{x}, \mathbf{y}) = \exp\{h^{-1} \|\mathbf{x} - \mathbf{y}\|^2\}$$

2. **Why least-squares based policy evaluation?**
   - LSPE(λ)/LSTD(λ) shares convergence with TD(λ)
   - But converges considerably faster (far fewer samples necessary)
   - Closed solution vs. stochastic gradient-descent

3. **LSPE vs. LSTD**
   - LSPE is incremental and can work with optimistic policy iteration
   - LSTD is batch $\Rightarrow$ no OPI
   - (Here, we will consider LSPE, but our algorithm also translates to LSTD)

So, how does LSPE work? ...

LSPE(λ) algorithm (Nedić & Bertsekas 2003)

APE: obtain $\tilde{Q}(\mathbf{x};\mathbf{w}^*) = \sum_{i=1}^{m} w_i^* \varphi_i(\mathbf{x})$ as an approximation for $Q^\pi(\mathbf{x})$

- **Initialize** $\mathbf{w}_0 = 0$, choose learning rate $\eta \in [0,1]$, generate $\mathbf{x}_0 = (s_0, \pi(s_0))$
- **For** $t = 0, 1, \ldots$ observe transition to $\mathbf{x}_{t+1}, r^*$ (simulating π)
  - $$\hat{\mathbf{w}}_{t+1} := \operatorname*{argmin}_{\mathbf{w}} \|\mathbf{\Phi}_{t+1}\mathbf{w} - \mathbf{\Phi}_{t+1}\mathbf{w}_t - \mathbf{\Lambda}_{t+1}(\mathbf{r}_{t+1} - \bar{\mathbf{\Phi}}_{t+1}\mathbf{w}_t)\|^2$$

    (this is just a standard least-squares problem!)
  - $\mathbf{w}_{t+1} := \mathbf{w}_t + \eta(\hat{\mathbf{w}}_{t+1} - \mathbf{w}_t)$
- **Convergence:** $\mathbf{w}_t \to \mathbf{w}^*$ the same as LSTD(λ),TD(λ)

Notation:

- Featurevector: $\boldsymbol{\varphi}_{t+1} := \boldsymbol{\varphi}(\mathbf{x}_{t+1}) = \big(\varphi_1(\mathbf{x}_{t+1}), \ldots, \varphi_m(\mathbf{x}_{t+1})\big)^\mathsf{T}$
- Recursion: (every new observation adds a new row)

$$\mathbf{\Phi}_{t+1} = \begin{bmatrix} \mathbf{\Phi}_t \\ \boldsymbol{\varphi}_t^\mathsf{T} \end{bmatrix} \quad \bar{\mathbf{\Phi}}_{t+1} = \begin{bmatrix} \bar{\mathbf{\Phi}}_t \\ \boldsymbol{\varphi}_t^\mathsf{T} - \gamma\boldsymbol{\varphi}_{t+1}^\mathsf{T} \end{bmatrix} \quad r_{t+1} = \begin{bmatrix} \mathbf{r}_t \\ r^* \end{bmatrix} \quad \mathbf{\Lambda}_{t+1} = \ldots$$

Now we will kernelize LSPE(λ) to do away with basis functions

Kernelization I: Primal

- Starting from primal form (dropping index $t+1$ from matrices)

$$\hat{\mathbf{w}}_{t+1} := \operatorname*{argmin}_{\mathbf{w}} \|\mathbf{\Phi w} - \mathbf{\Phi w}_t - \mathbf{\Lambda}(\mathbf{r} - \bar{\mathbf{\Phi}}\mathbf{w}_t)\|^2 + \underbrace{\sigma^2\|\mathbf{w} - \mathbf{w}_t\|^2}_{\text{weight regularizer as in RR}}$$

  Computing derivative wrt $\mathbf{w}$ and equating with zero

$$\Longrightarrow \hat{\mathbf{w}}_{t+1} = \mathbf{w}_t + (\mathbf{\Phi}^\mathsf{T}\mathbf{\Phi} + \sigma^2\mathbf{I})^{-1}\mathbf{\Phi}^\mathsf{T}\mathbf{\Lambda}(\mathbf{r} - \bar{\mathbf{\Phi}}\mathbf{w}_t)$$

- Apply Sherman-Morrison-Woodbury

$$(\mathbf{\Phi}^\mathsf{T}\mathbf{\Phi} + \sigma^2\mathbf{I})^{-1}\mathbf{\Phi}^\mathsf{T} = \mathbf{\Phi}^\mathsf{T}(\mathbf{\Phi}\mathbf{\Phi}^\mathsf{T} + \sigma^2\mathbf{I})^{-1}$$

  to obtain

$$\hat{\mathbf{w}}_{t+1} = \mathbf{w}_t + \mathbf{\Phi}^\mathsf{T}(\mathbf{\Phi}\mathbf{\Phi}^\mathsf{T} + \sigma^2\mathbf{I})^{-1}\mathbf{\Lambda}(\mathbf{r} - \bar{\mathbf{\Phi}}\mathbf{w}_t)$$

- $\Longrightarrow \forall t$ all solutions $\hat{\mathbf{w}}_t$ lie in column space of $\mathbf{\Phi}^\mathsf{T}$, i.e. my be expressed as

$$\hat{\mathbf{w}}_t = \mathbf{\Phi}^\mathsf{T}\boldsymbol{\alpha} = \sum_{i=1}^{t} \varphi(\mathbf{x}_i)\alpha_i$$

  for some dual variables $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_t)^\mathsf{T}$

Now consider expressing LSPE in dual variables, replacing all $\mathbf{w}$'s by $\Phi^\mathsf{T}\alpha$'s ...

Kernelization II: Dual

- Replacing iterates $\hat{\mathbf{w}}_{t+1}, \mathbf{w}_{t+1}$ by dual variables

$$\hat{\boldsymbol{\alpha}}_{t+1} = \boldsymbol{\alpha}_t + (\boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T} + \sigma^2\mathbf{I})^{-1}\boldsymbol{\Lambda}(\mathbf{r} - \bar{\boldsymbol{\Phi}}\boldsymbol{\Phi}^\mathsf{T}\boldsymbol{\alpha}_t)$$

- **Kernel trick:** kernel computes ip in feature space: $k(\mathbf{x}, \mathbf{y}) = \langle \varphi(\mathbf{x}), \varphi(\mathbf{y}) \rangle$
- Define:
  - $\mathbf{K} := \boldsymbol{\Phi}\boldsymbol{\Phi}^\mathsf{T}$, $[\mathbf{K}]_{ij} = k(\mathbf{x}_i, \mathbf{y}_j)$
  - $\mathbf{H} := \bar{\boldsymbol{\Phi}}\boldsymbol{\Phi}^\mathsf{T}$, $[\mathbf{H}]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) - \gamma k(\mathbf{x}_{i+1}, \mathbf{x}_j)$
  - $k(\cdot) := \big(k(\mathbf{x}_1, \cdot,), \ldots, k(\mathbf{x}_t, \cdot)\big)^\mathsf{T}$
- Therefore

$$\hat{\boldsymbol{\alpha}}_{t+1} = \boldsymbol{\alpha}_t + (\mathbf{K} + \sigma^2\mathbf{I})^{-1}\boldsymbol{\Lambda}(\mathbf{r} - \mathbf{H}\boldsymbol{\alpha}_t)$$

- Hence, the LSPE update is (in dual variables)

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t + \eta(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\boldsymbol{\Lambda}(\mathbf{r} - \mathbf{H}\boldsymbol{\alpha}_t)$$

Now we can state the dualized LSPE(λ) algorithm

Kernelizing III

- Predictions with $\tilde{Q}$ in arbitrary $\mathbf{x}^*$ also possible from dual variables:

$$\tilde{Q}(\mathbf{x}^*; \mathbf{w}_t) = \langle \varphi(\mathbf{x}^*), \mathbf{w}_t \rangle = \langle \varphi(\mathbf{x}^*), \sum_{i=0}^{t} \alpha_t^{(i)} \varphi(\mathbf{x}_i) \rangle = \sum_{i=0}^{t} \alpha_t^{(i)} k(\mathbf{x}_i, \mathbf{x}^*) = \tilde{Q}(\mathbf{x}^*; \boldsymbol{\alpha}_t)$$

- Hence: complete LSPE algorithm may be stated in dual variables.

Advantages of kernel-based approach

- eliminates explicit choice of nodes/basis functions $\varphi_i(\cdot)$ (at first glance)
- just choose a kernel, e.g. the Gaussian $k(\mathbf{x}, \mathbf{y}) = \exp -h^{-1} \|\mathbf{x} - \mathbf{y}\|^2$

Problem: LSPE is an incremental and online algorithm that continually updates the solution. For each time-step t we need to tackle a t-by-t problem ...

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)

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

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

Reduced problem:

- **Replace:** every occurance of $k(\mathbf{x}, \mathbf{x}')$ by $\mathbf{k}_m(\mathbf{x})^\mathsf{T} \mathbf{K}_{mm}^{-1} \mathbf{k}_m(\mathbf{x}')$:
  - $\tilde{Q}(\cdot\,; \boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha^{(i)} k(\tilde{\mathbf{x}}_i, \cdot\,)$
  - $\boldsymbol{\alpha}_{t+1,m} = \boldsymbol{\alpha}_{tm} + \eta \mathbf{P}_{t+1,m}^{-1} (\mathbf{Z}_{t+1,m}^\mathsf{T} \mathbf{r}_{t+1} - \mathbf{Z}_{t+1,m}^\mathsf{T} \mathbf{H}_{t+1,m} \boldsymbol{\alpha}_{tm})$

  where $\mathbf{P}_{t+1,m} = (\mathbf{K}_{t+1,m}^\mathsf{T} \mathbf{K}_{t+1,m} + \sigma^2 \mathbf{K}_{mm})$, $[\mathbf{K}_{t+1,m}]_{ij} = k(\mathbf{x}_i, \tilde{\mathbf{x}}_j)$, $[\mathbf{H}_{t+1,m}]_{ij} = k(\mathbf{x}_i, \tilde{\mathbf{x}}_j) - \gamma k(\mathbf{x}_{i+1}, \tilde{\mathbf{x}}_j)$, $\mathbf{Z}_{t+1,m}^\mathsf{T} := \mathbf{K}_{t+1,m}^\mathsf{T} \boldsymbol{\Lambda}_{t+1}$
- **If the subset $\{\tilde{\mathbf{x}}\}_{i=1}^{m}$ were somehown known in advance:**
  - Reduces to a *fixed* basis function network
  - Recursive implementation possible at $\mathcal{O}(m^2)$ per-step

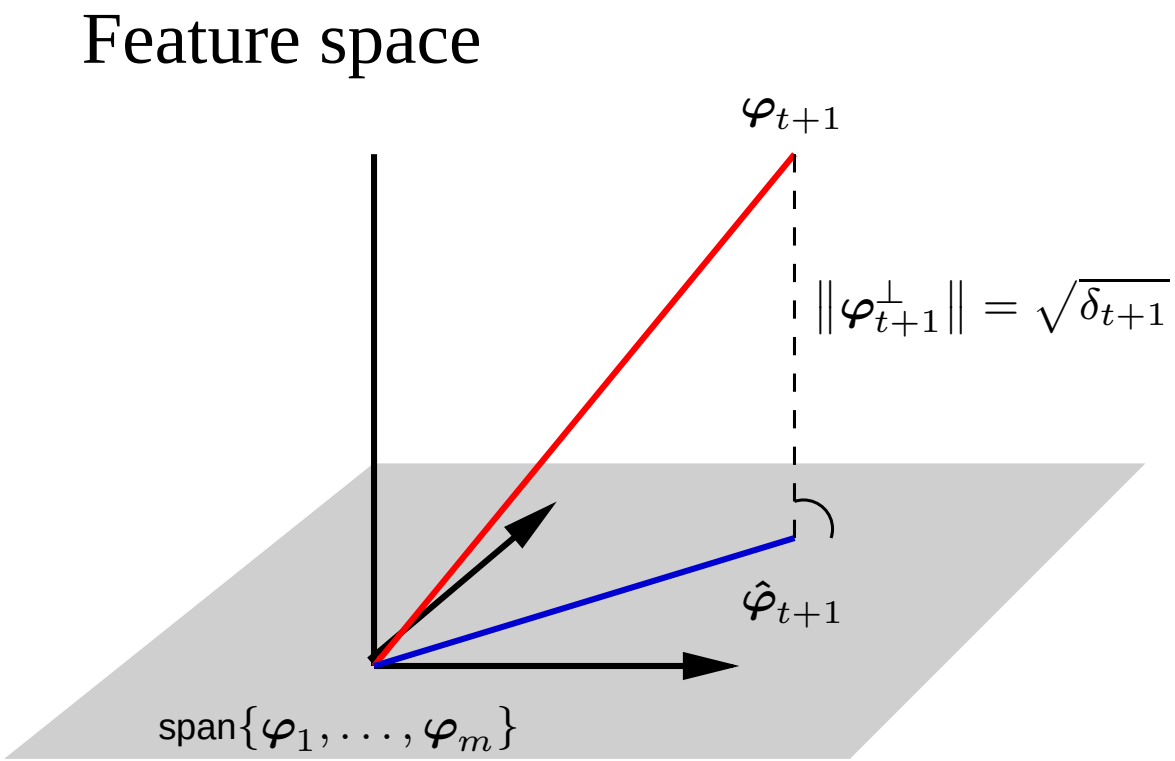
But: how to select the subset, when data arrives only sequentially ...

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 $\delta_{t+1} = k(\mathbf{x}_t, \mathbf{x}_t) - \mathbf{k}_m(\mathbf{x}_t)^{\mathsf{T}} \mathbf{K}_{mm}^{-1} \mathbf{k}_m(\mathbf{x}_t) > \texttt{TOL}$ 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 $\mathbf{x}_{t+1}$, that is

$$\mathbf{K}_{t+1,m+1} = \begin{bmatrix} \mathbf{K}_{t+1,m} & \mathbf{q} \end{bmatrix} \approx \begin{bmatrix} \mathbf{K}_{tm} & \mathbf{K}_{tm}\mathbf{a}_{t+1} \\ \mathbf{k}_m(\mathbf{x}_{t+1})^\mathsf{T} & k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) \end{bmatrix} =: \tilde{\mathbf{K}}_{t+1,m+1},$$

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

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

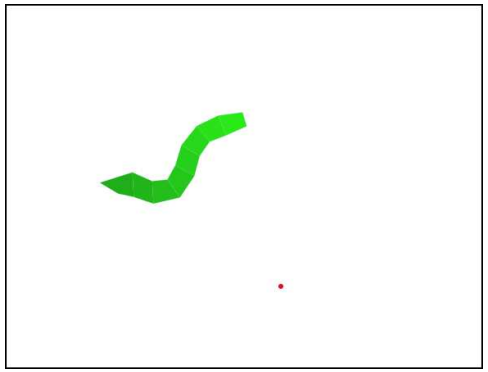
Kernel-LSPE(λ) /w online basis selection

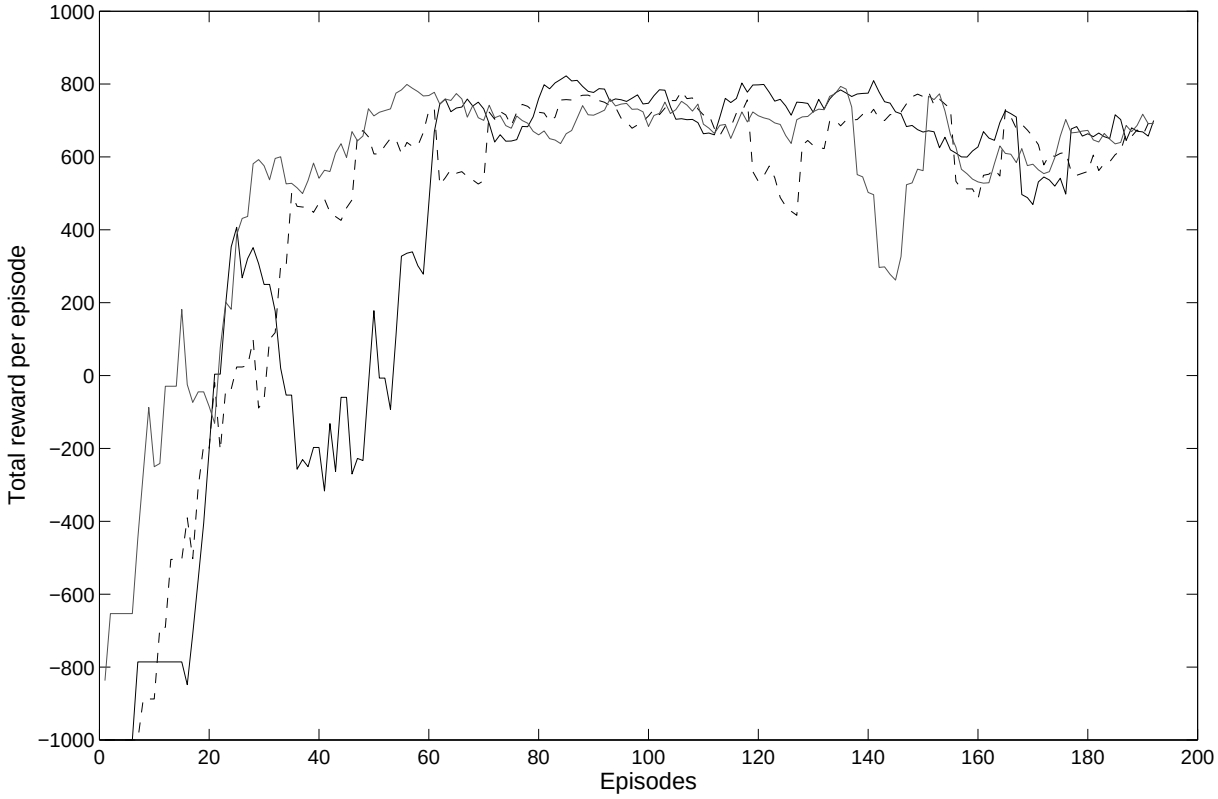
Input: policy π	
Output: dictionary $\mathcal{D}$ of basis functions	
Q-function: $\tilde{Q}(\cdot) = \sum_{i=1}^{\lvert\mathcal{D}\rvert} \alpha_i k(\tilde{\mathbf{x}}_i, \cdot)$	
Initialize	
FOR $t = 1, 2, \ldots$	
Execute action a_t	
Observe next state s_{t+1} and reward r_{t+1}	
Choose action $a_{t+1} = \pi(s_{t+1})$. Form $\mathbf{x}_{t+1} := (s_{t+1}, a_{t+1})$.	
IF $\left\lVert \boldsymbol{\varphi}_{t+1}^{\perp} \right\rVert^2 > \mathsf{TOL}$	$\mathcal{O}(m^2)$
// Update weight vector and augment basis	
$\boldsymbol{\alpha}_{t+1,m+1} = \begin{bmatrix} \boldsymbol{\alpha}_{tm} \\ 0 \end{bmatrix} + \cdots$	$\mathcal{O}(m^2)$
$\mathcal{D} = \mathcal{D} \cup \{\mathbf{x}_{t+1}\}$, $m = m + 1$	
ELSE	
// Update weight vector without augmenting basis	
$\boldsymbol{\alpha}_{t+1,m} = \boldsymbol{\alpha}_{tm} + \cdots$	$\mathcal{O}(m^2)$
$s_t = s_{t+1}, a_t = a_{t+1}$	

Experiments I: Octopus

Testbed: 8-compartment 2D-octopus arm with 'HardTask' from ICML-06 benchmark

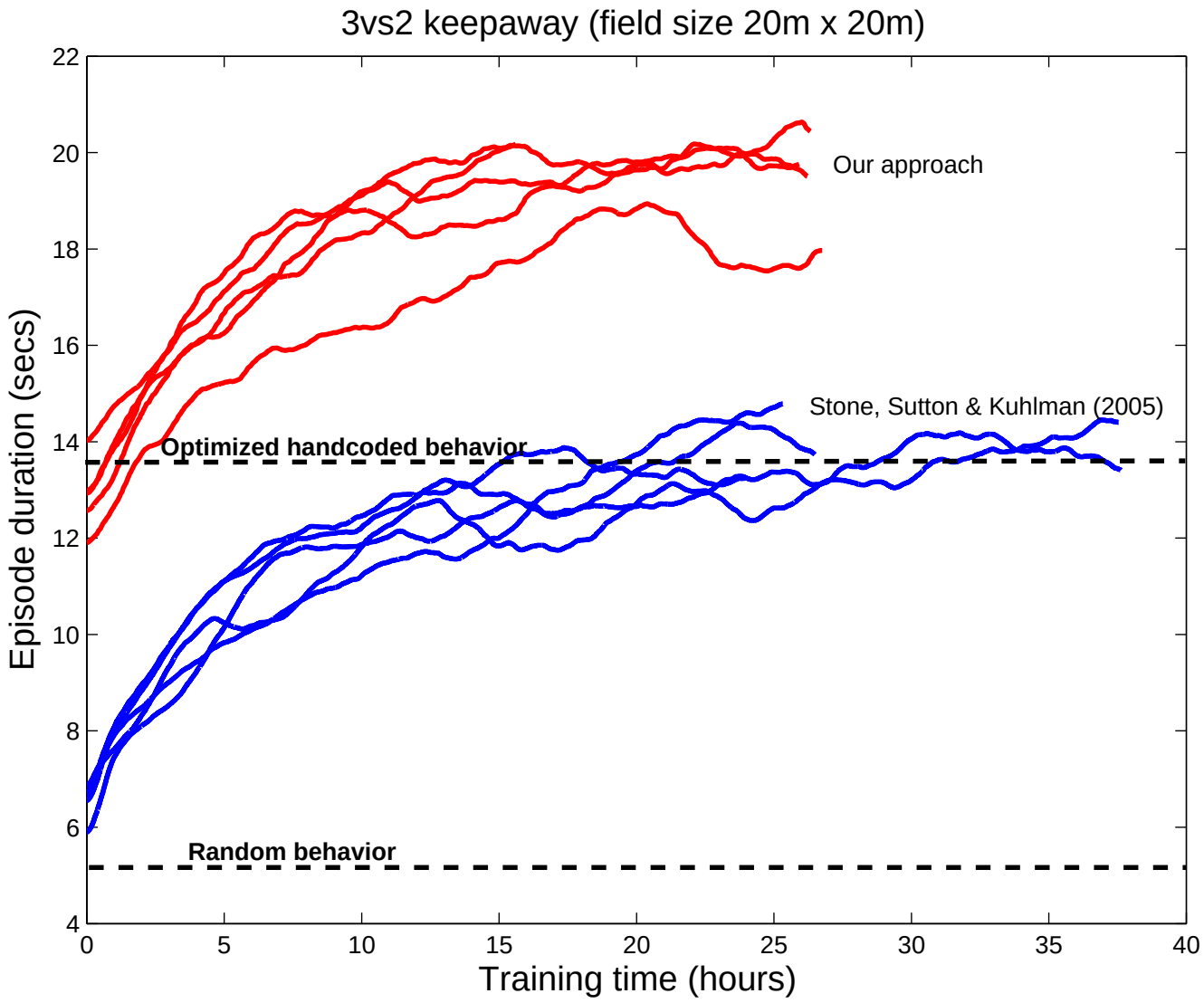
Result: OPI with Kernel-LSPE





Experiments II: RoboCup Keepaway 3vs2

Compare: Our Kernel-LSTD with API (no OPI) vs. the textbook approach sarsa(λ) + tilecoding



Summary

Talked about:

- **Topic:** Online APE with LSPE and 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:**
  - Effortlessly scales to high-dimensional control tasks
  - However, ultimately limited by MAX-size of dictionary ($m \sim 2000$, depending on CPU)
- **Applications:**
  - LSTD, LSPE, fitted value iteration (not tried),...
  - In fact, everything that is a least-squares type of problem and would benefit from sequential learning (e.g. time-series prediction ...)