Kernelizing LSPE(\(\lambda\))

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

N compartments

Challenges:

- high-dimensional state space $\subset \mathbb{R}^{(2N+2)\times 4}$ (e.g. $N=8$ compartments $\rightarrow \mathbb{R}^{72}$)
- high-dimensional and continuous 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 $O(n^3)$ 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 $O(m^2)$. 
**Policy evaluation:** (Model-free!)

\[
Q^\pi(s, a) = E_{s'\mid s, a}\left\{ R(s'\mid s, a) + \gamma Q^\pi(s', \pi(s')) \right\}
\]

where

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

Kernelizing LSPE(\( \lambda \)) — ADPRL 2007 – p.5/19
Approximate policy evaluation

1. **Function approximation**: represent \( Q^\pi \) through linearly parameterized approximation

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

where

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

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

\[
\begin{array}{c}
\begin{array}{c}
 s_t \\
 \xrightarrow{\pi_k(s_t)}
\end{array}
\begin{array}{c}
 r_{t+1}
\end{array}
\begin{array}{c}
 s_{t+1} \\
 \xrightarrow{\pi_k(s_{t+1})}
\end{array}
\begin{array}{c}
 r_{t+2}
\end{array}
\begin{array}{c}
 s_{t+2} \\
 \rightarrow
\end{array}
\end{array}
\]

Of course, we’re not just interested in merely obtaining \( Q^\pi \) ...
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 $w$, 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

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

2. Why least-squares based policy evaluation?
   - LSPE(\(\lambda\))/LSTD(\(\lambda\)) shares convergence with TD(\(\lambda\))
   - 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(\(\lambda\)) algorithm (Nedić & Bertsekas 2003)**

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

- **Initialize** \(w_0 = 0\), choose learning rate \(\eta \in [0, 1]\), generate \(x_0 = (s_0, \pi(s_0))\)
- **For** \(t = 0, 1, \ldots\) observe transition to \(x_{t+1}, r^*\) (simulating \(\pi\))

\[
\hat{w}_{t+1} := \arg\min_{w} \| \Phi_{t+1} w - \Phi_{t+1} w_t - \Lambda_{t+1}(r_{t+1} - \Phi_{t+1} w_t) \|^2
\]

(this is just a standard least-squares problem!)

- \(w_{t+1} := w_t + \eta(\hat{w}_{t+1} - w_t)\)
- **Convergence:** \(w_t \rightarrow w^*\) the same as LSTD(\(\lambda\)), TD(\(\lambda\))

**Notation:**

- **Feature vector:** \(\varphi_{t+1} := \varphi(x_{t+1}) = (\varphi_1(x_{t+1}), \ldots, \varphi_m(x_{t+1}))^T\)
- **Recursion:** (every new observation adds a new row)

\[
\Phi_{t+1} = \begin{bmatrix} \Phi_t \\ \varphi_t^T \end{bmatrix} \quad \Phi_{t+1} = \begin{bmatrix} \Phi_t \\ \varphi_t^T - \gamma \varphi_{t+1}^T \end{bmatrix} \quad r_{t+1} = \begin{bmatrix} r_t \\ r^* \end{bmatrix} \quad \Lambda_{t+1} = \ldots
\]

Now we will kernelize LSPE(\(\lambda\)) to do away with basis functions
Kernelization I: Primal

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

\[
\hat{w}_{t+1} := \arg\min_w \| \Phi w - \Phi w_t - \Lambda (r - \overline{\Phi} w_t) \|^2 + \sigma^2 \| w - w_t \|^2
\]

Computing derivative wrt \( w \) and equating with zero

\[
\implies \hat{w}_{t+1} = w_t + (\Phi^T \Phi + \sigma^2 I)^{-1} \Phi^T \Lambda (r - \overline{\Phi} w_t)
\]

Apply Sherman-Morrison-Woodbury

\[
(\Phi^T \Phi + \sigma^2 I)^{-1} \Phi^T = \Phi^T (\Phi \Phi^T + \sigma^2 I)^{-1}
\]

to obtain

\[
\hat{w}_{t+1} = w_t + \Phi^T (\Phi \Phi^T + \sigma^2 I)^{-1} \Lambda (r - \overline{\Phi} w_t)
\]

\( \implies \forall t \) all solutions \( \hat{w}_t \) lie in column space of \( \Phi^T \), i.e. my be expressed as

\[
\hat{w}_t = \Phi^T \alpha = \sum_{i=1}^{t} \varphi(x_i) \alpha_i
\]

for some dual variables \( \alpha = (\alpha_1, \ldots, \alpha_t)^T \)

Now consider expressing LSPE in dual variables, replacing all \( w \)'s by \( \Phi^T \alpha \)'s ...
Replacing iterates $\hat{w}_{t+1}, w_{t+1}$ by dual variables

$$\hat{\alpha}_{t+1} = \alpha_t + (\Phi \Phi^T + \sigma^2 I)^{-1} \Lambda (r - \Phi \Phi^T \alpha_t)$$

**Kernel trick:** kernel computes $\langle \phi(x), \phi(y) \rangle$

**Define:**
- $K := \Phi \Phi^T$, $[K]_{ij} = k(x_i, y_j)$
- $H := \Phi \Phi^T$, $[H]_{ij} = k(x_i, x_j) - \gamma k(x_{i+1}, x_j)$
- $k(\cdot) := (k(x_1, \cdot), \ldots, k(x_t, \cdot))^T$

Therefore

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

Hence, the LSPE update is (in dual variables)

$$\alpha_{t+1} = \alpha_t + \eta (K + \sigma^2 I)^{-1} \Lambda (r - H \alpha_t)$$

Now we can state the dualized LSPE($\lambda$) algorithm
Predictions with $\tilde{Q}$ in arbitrary $x^*$ also possible from dual variables:

$$\tilde{Q}(x^*; w_t) = \langle \varphi(x^*), w_t \rangle = \langle \varphi(x^*), \sum_{i=0}^{t} \alpha_{t}^{(i)} \varphi(x_i) \rangle = \sum_{i=0}^{t} \alpha_{t}^{(i)} k(x_i, x^*) = \tilde{Q}(x^*; \alpha_t)$$

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

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

\[
k(x, x') = k_m(x)^\top K_{mm}^{-1} k_m(x') \quad \forall x, x'
\]

where \( k_m(\cdot) = (k(\tilde{x}_1, \cdot), \ldots, k(\tilde{x}_m, \cdot))^\top \) and \( [K_{mm}]_{ij} = k(\tilde{x}_i, \tilde{x}_j) \).

Reduced problem:

- **Replace**: every occurrence of \( k(x, x') \) by \( k_m(x)^\top K_{mm}^{-1} k_m(x') \):

  - \( \tilde{Q}(\cdot; \alpha) = \sum_{i=1}^{m} \alpha^{(i)} k(\tilde{x}_i, \cdot) \)
  - \( \alpha_{t+1, m} = \alpha_{t m} + \eta P_{t+1, m}^{-1} (Z_{t+1, m} r_{t+1} - \tilde{Z}_{t+1, m} H_{t+1, m} \alpha_{t m}) \)

where \( P_{t+1, m} = (K_{t+1, m}^\top K_{t+1, m} + \sigma^2 K_{m m}) \), \( [K_{t+1, m}]_{ij} = k(x_i, \tilde{x}_j) \), \( [H_{t+1, m}]_{ij} = k(x_i, \tilde{x}_j) - \gamma k(x_{i+1}, \tilde{x}_j) \), \( \tilde{Z}_{t+1, m} \) := \( K_{t+1, m} \Lambda_{t+1} \)

- **If the subset** \( \{\tilde{x}\}_{i=1}^{m} \) **were somehow known in advance**:
  - Reduces to a fixed basis function network
  - Recursive implementation possible at \( O(m^2) \) 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 $x_t$ from the current dictionary:

**Feature space**

$$\varphi_{t+1}$$

$$\text{span}\{\varphi_1, \ldots, \varphi_m\}$$

- **Criterion:** if $\delta_{t+1} = k(x_t, x_t) - k_m(x_t)^\top K_{mm}^{-1} k_m(x_t) > \text{TOL}$ then $x_t$ 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?
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 $x_{t+1}$, that is

$$K_{t+1,m+1} = \begin{bmatrix} K_{t+1,m} & q \end{bmatrix} \approx \begin{bmatrix} K_{tm} & K_{tm} a_{t+1} \\ k_m(x_{t+1})^T & k(x_{t+1}, x_{t+1}) \end{bmatrix} =: \tilde{K}_{t+1,m+1},$$

where $a_{t+1} = K_{mm}^{-1} k_m(x_{t+1})$, only costs $O(m^2)$ operations instead of the usual $O(tm)$.

Now we can piece together an efficient online implementation...
**Kernel-LSPE(\( \lambda \)) /w online basis selection**

<table>
<thead>
<tr>
<th><strong>Input:</strong> policy ( \pi )</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output:</strong> dictionary ( \mathcal{D} ) of basis functions</td>
<td></td>
</tr>
<tr>
<td>Q-function: ( \tilde{Q}(\cdot) = \sum_{i=1}^{\left</td>
<td>\mathcal{D} \right</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Initialize</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FOR</strong> ( t = 1, 2, \ldots )</td>
<td></td>
</tr>
<tr>
<td>Execute action ( a_t )</td>
<td></td>
</tr>
<tr>
<td>Observe next state ( s_{t+1} ) and reward ( r_{t+1} )</td>
<td></td>
</tr>
<tr>
<td>Choose action ( a_{t+1} = \pi(s_{t+1}) ). Form ( x_{t+1} := (s_{t+1}, a_{t+1}) ).</td>
<td></td>
</tr>
<tr>
<td><strong>IF</strong> ( | \varphi_{t+1}^\perp |^2 &gt; \text{TOL} )</td>
<td>( \mathcal{O}(m^2) )</td>
</tr>
<tr>
<td>// Update weight vector and augment basis</td>
<td></td>
</tr>
<tr>
<td>( \alpha_{t+1,m+1} = \begin{bmatrix} \alpha_{tm} \ 0 \end{bmatrix} + \cdots )</td>
<td>( \mathcal{O}(m^2) )</td>
</tr>
<tr>
<td>( \mathcal{D} = \mathcal{D} \cup {x_{t+1}}, m = m + 1 )</td>
<td></td>
</tr>
<tr>
<td><strong>ELSE</strong></td>
<td></td>
</tr>
<tr>
<td>// Update weight vector without augmenting basis</td>
<td></td>
</tr>
<tr>
<td>( \alpha_{t+1,m} = \alpha_{tm} + \cdots )</td>
<td>( \mathcal{O}(m^2) )</td>
</tr>
<tr>
<td>( s_t = s_{t+1}, a_t = a_{t+1} )</td>
<td></td>
</tr>
</tbody>
</table>
Testbed: 8-compartment 2D-octopus arm with 'HardTask' from ICML-06 benchmark

Result: OPI with Kernel-LSPE
Compare: Our Kernel-LSTD with API (no OPI) vs. the textbook approach \textit{sarsa}(\lambda) + 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: $O(m^2)$ 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 ...)

Kernelizing LSPE($\lambda$) — ADPRL 2007 – p.19/19