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 $O(m^2)$ time [normal would be $O(tm)$]
- Compute the contribution (reduction of error) of one basis function candidate during greedy forward selection in $O(m^2)$ time [normal would be $O(tm)$]
- Compute predictive variance with augmentation in $O(m^2)$ time [normal would be $O(tm)$]
Regularization networks

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

**Objective:**

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

\[
\min_{f \in \mathcal{H}_k} \sum_{i=1}^t (y_i - f(x_i))^2 + \lambda \|f\|^2_{\mathcal{H}_k}
\]

**Solution:**

- **Representer theorem:** may assume \( f(\cdot) = \sum_i k(x_i, \cdot)w_i \). Thus solve \( t \)-by-\( t \) problem

\[
\min_{w \in \mathbb{R}^t} \|Kw - y\|^2 + \lambda w^T Kw
\]

**Solution:**

\[
w = (K + \lambda I)^{-1}y
\]

where \( k(\cdot, \cdot) \) is the kernel function (e.g. polynomials, Gaussian RBF, ...), \( K \) is the \( t \times t \) kernel matrix \( [K]_{ij} = k(x_i, x_j) \) and \( \lambda \) 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 \( \{\tilde{x}\}_{i=1}^{m} \), where \( m \ll t \)
- **Approximate:** the kernel by (also arises from the Nyström approximation)

\[
k(x, x') = [k_m(x)]^T K_{mm}^{-1} k_m(x') \quad \forall x, x'
\]

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

Reduced problem:

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

\[
\min_{w \in \mathbb{R}^m} \|K_{tm} w - y\|^2 + \lambda w^T K_{mm} w
\]

- **Solution:**

\[
w = (K_{tm}^T K_{tm} + \lambda K_{mm})^{-1} K_{tm}^T y
\]

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

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 \( x_t \) from the current dictionary:

  ![Diagram](image)

- **Criterion:** if \( k(x_t, x_t) - [k_m(x_t)]^T K_{mm}^{-1} k_m(x_t) > \text{TOL} \) 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 some $x^*$, that is

$$K_{t,m+1} = \begin{bmatrix} K_{tm} & q \end{bmatrix} = \begin{bmatrix} K_{t-1,m} & K_{t-1,m}a^* \\ k_m(x_t)^T & k(x_t, x^*) \end{bmatrix},$$

where $a^* = K_{mm}^{-1}k_m(x^*)$, only costs $O(m^2)$ operations instead of the usual $O(tm)$.

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

Outline:

- **Solve:** $\min_w J_{tm}(w) = \|K_{tm}w - y_t\|^2 + \lambda w^T K_{mm}w$
- **Normal equations:** $w_{tm} = (K_{tm}^T K_{tm} + \lambda K_{mm})^{-1} K_{tm}^T y_t$
- **Abbreviations:**
  - Cross-product matrix $P_{tm} := (K_{tm}^T K_{tm} + \lambda K_{mm})$
  - Regularized costs $\xi_{tm} := J_{tm}(w_{tm})$
- **Goal:** update $\{P_{tm}^{-1}, w_{tm}, \xi_{tm}\}$ 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 $(x_{t+1}, y_{t+1})$
- Propagate $\{P_{tm}^{-1}, w_{tm}, \xi_{tm}\}$ forward in time by the operations
  - **Normal step:** $\{P_{tm}^{-1}, w_{tm}, \xi_{tm}\} \rightarrow \{P_{t+1,m}^{-1}, w_{t+1,m}, \xi_{t+1,m}\}$
  - **Growing step:** $\{P_{tm}^{-1}, w_{tm}, \xi_{tm}\} \rightarrow \{P_{t,m+1}^{-1}, w_{t,m+1}, \xi_{t,m+1}\}$
  - **Pruning step:** $\{P_{tm}^{-1}, w_{tm}, \xi_{tm}\} \rightarrow \{P_{t,m\backslash i}^{-1}, w_{t,m\backslash i}, \xi_{t,m\backslash i}\}$
Normal step: $\{P_{tm}^{-1}, w_{tm}, \xi_{tm}\} \rightarrow \{P_{t+1,m}^{-1}, w_{t+1,m}, \xi_{t+1,m}\}$

Easy: just the usual recursive least squares update: $P_{t+1,m} = P_{tm} + k_{t+1}k_{t+1}^T$, thus

$$P_{t+1,m}^{-1} = P_{tm}^{-1} - \frac{P_{tm}^{-1}k_{t+1}k_{t+1}^TP_{tm}^{-1}}{\Delta}, \quad w_{t+1,m} = w_{tm} + \frac{\varrho}{\Delta}P_{tm}^{-1}k_{t+1}$$

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

where

- $k_{t+1} := k_m(x_{t+1})$ (m x 1 vector)
- $\Delta = 1 + k_{t+1}^TP_{tm}^{-1}k_{t+1}$ (scalar)
- $\varrho = y_{t+1} - k_{t+1}^Tw_{tm}$ (scalar)

Operation count is $\mathcal{O}(m^2)$. 
Growing step: \( \{P_{tm}^{-1}, w_{tm}, \xi_{tm}\} \rightarrow \{P_{t,m+1}^{-1}, w_{t,m+1}, \xi_{t,m+1}\} \)

**How to add a basis function:** centered on \( x_{t+1} \)

\[
P_{t,m+1}^{-1} = \begin{bmatrix} P_{tm}^{-1} & 0 \\ 0 & 0 \end{bmatrix} + \frac{1}{\Delta_b} \begin{bmatrix} -w_b \\ 1 \end{bmatrix} \begin{bmatrix} -w_b \\ 1 \end{bmatrix}^T
\]

where

- \( w_b = a_{t+1} + \frac{\delta}{\Delta} P_{t-1,m}^{-1} k_{t+1}, \quad \Delta_b = \frac{\delta^2}{\Delta} + \lambda \delta \)
- \( \delta = k(x_{t+1}, x_{t+1}) - k_{t+1}^T a_{t+1} \)

can be obtained in \( O(m) \) operations (caching and reusing terms).

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

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

- **Novelty:** \( \delta = k(x_{t+1}, x_{t+1}) - k_{t+1}^T a_{t+1} \) (i.e. the reconstruction error from sparse greedy approximation)
- **Usefulness:** \( \delta^2 \frac{\varphi^2}{(\Delta_b \Delta^2)} \) (reduction of costs when we would add it to the basis)

Operation count is \( O(m^2) \).
Pruning step: \( \{ \mathbf{P}_{tm}^{-1}, \mathbf{w}_{tm}, \xi_{tm} \} \rightarrow \{ \mathbf{P}_{t,m\backslash i}^{-1}, \mathbf{w}_{t,m\backslash i}, \xi_{t,m\backslash i} \} \)

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

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

(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)

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

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

Operation count is \( \mathcal{O}(m^2) \).
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
Large-scale real world benchmark: Our approach vs. OLS-59

Recent work: application to approximate policy evaluation in reinforcement learning...
**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(\(\lambda\)) + tilecoding

![Graph showing 3vs2 keepaway (field size 20m x 20m)]
Talked about:

- **Topic**: Online learning with 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**:
  - 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