Least Squares SVM
for
Least Squares TD Learning

Tobias Jung, University of Mainz, Germany
Daniel Polani, University of Hertfordshire, U.K.
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(λ))
- Least squares SVM (superior to fixed parameterized function approximators)
Dynamic Programming I

A Markov decision process basically consists of

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

**Objective:** choose actions to maximize performance

**Hitch:** usually *delayed reward*

→ choose actions to maximize **long-term reward**
Dynamic Programming II

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

How do we get there?

- **Policy:** $\pi : S \rightarrow A$ (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 $V^\pi = T^\pi V^\pi$, where

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

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

\[
\pi_{k+1}(s) = \arg\max_a \left\{ \sum_{s'} P(s'|s,a) \left[ R(s'|s,a) + \gamma V^{\pi_k}(s') \right] \right\}, \forall s
\]

**Problems:**

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

$\rightarrow$ Approximate policy iteration, approximate policy evaluation (with least squares)
Approximate dynamic programming

Assume: value function is linearly parameterized

$$\tilde{V}(s) = [\phi_m(s)]^T w$$

where $\phi_m(s) = [\phi_1(s), \ldots, \phi_s]$ a $m \times 1$ feature vector

w a $m \times 1$ weight vector

$\phi_i(s) : S \rightarrow \mathbb{R}$ 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)

$$s_0 \quad a_0 \quad s_1 \quad a_1 \quad s_2 \quad \ldots \quad r_{t-1} \quad a_{t-1} \quad s_t$$

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 $w$ in $V^\pi_k(s) = [\phi_m(s)]^T w$ from trajectory $s_0, s_1, \ldots, s_t$ and rewards $r_1, \ldots, r_t$.

- **Bellman residual minimization approach:** determine weights $w$ by

$$w = \arg\min_w \left\{ \sum_{i=0}^{t-1} [\phi_m(s_i)]^T \hat{w} - \sum_{s'} P(s'|s_i, \pi_k(s_i)) \left[ R(s'|s_i, \pi_k(s_i)) + \gamma [\phi_m(s_{i+1})]^T \hat{w} \right] \right\}^2$$

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

$$w = \arg\min_w \left\{ \sum_{i=0}^{t-1} [\phi_m(s_i)]^T \hat{w} - \left[ r_i + \gamma [\phi_m(s_{i+1})]^T \hat{w} \right] \right\}^2$$

(For **stochastic transitions** this is not possible $\rightarrow$ need ‘doubled’ samples)

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

$$w = \arg\min_w \left\{ \sum_{i=0}^{t-1} [\phi_m(s_i)]^T \hat{w} - \left[ r_i + \gamma [\phi_m(s_{i+1})]^T w \right] \right\}^2$$

(For now we will ignore LSTD(0), but see recent work at end.)
Least squares in approximate policy evaluation:

What we are looking for:

Bellman residual minimization:

Fixed point approximation:

At a glance:

- **Bellman residual minimization approach:** \( \| \hat{V} - T_\pi \hat{V} \|^2 \rightarrow \min \hat{V} \)

- **Fixed point approximation approach (LSTD):** \( V = \arg\min \hat{V} \| \hat{V} - T_\pi V \|^2 \)

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 $\{x_i, y_i\}_{i=1}^t$
- 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 $\{\tilde{x}_i\}_{i=1}^m$ of the training data, where $m \ll t$
- Represent: the solution by $f(\cdot) = \sum_{i=1}^m k(\tilde{x}_i, \cdot)w_i$
- Solve: the quadratic $t$-by-$m$ problem to obtain the weights $w$

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

where

$[K_{tm}]_{ij} = k(x_i, \tilde{x}_j)$  a $t \times m$ matrix

$[K_{mm}]_{ij} = k(\tilde{x}_i, \tilde{x}_j)$  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 $V^\pi(\cdot) = \sum_{i=1}^{m} k(\tilde{s}_i, \cdot) w_i$ where $\{\tilde{s}_i\}_{i=1}^{m}$ is a subset
- Solve the quadratic t-by-m problem corresponding to Bellman

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

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:

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

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

  ![Diagram 1](image1)

  ![Diagram 2](image2)

- **Criterion:** if \( k(s_t, s_t) - [k_m(s_t)]^T K_{mm}^{-1} k_m(s_t) > \text{TOL} \) 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 $s_{t-1} \rightarrow s_t$ under reward $r_t$

- **Goal:** update $w_{tm} = \begin{bmatrix} H_{t-1,m}^T \\ h_t^T \end{bmatrix} T \begin{bmatrix} H_{t-1,m} \\ h_t T \end{bmatrix} + \lambda K_{mm} \begin{bmatrix} H_{t-1,m} \\ h_t T \end{bmatrix}^{-1} \begin{bmatrix} H_{t-1,m}^T \\ h_t^T \end{bmatrix} \begin{bmatrix} r_{t-1} \\ r_t \end{bmatrix}$

- **Definition:** $P_{t-1,m} = (H_{t-1,m}^T H_{t-1,m} + \lambda K_{mm})^{-1}$, $s_{t-1,m} = H_{t-1,m}^T r_{t-1}$

- **Rank-1 updates** via:
  - Route 1: Formula of Sherman-Morrison-Woodbury ("recursive least squares")
  - Route 2: Update of Cholesky factorization: $P_{t-1,m} = \Phi_{t-1,m}^{1/2} \Phi_{t-1,m}^{T/2}$

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

- Update $\{P_{t-1,m}, s_{t-1,m}, w_{t-1,m}\} \rightarrow \{P_{tm}, s_{tm}, w_{tm}\}$
- Costs $O(m^2)$

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

- Add $s_t$ as new basis function to the dictionary
- Update $\{P_{t-1,m}, s_{t-1,m}, w_{t-1,m}\} \rightarrow \{P_{t,m+1}, s_{t,m+1}, w_{t,m+1}\}$
- Costs $O(m^2)$

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)

![Graph showing performance comparison](image)

**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 $\implies$ 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 $\text{sarsa}(\lambda) + \text{tilecoding}$ vs. our approach
Demonstration?

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