SARSA (\(\lambda\)) In RKHS

Matthew W. Robards, Peter Sunehag, Scott Sanner

COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

June 16, 2010
We are primarily interested in reinforcement learning in large and continuous spaces which requires good feature selection.
Motivation

- We are primarily interested in reinforcement learning in large and continuous spaces which requires good feature selection.
- Hand engineering features results in poor generalization in an agent across domains.
Motivation

- We are primarily interested in reinforcement learning in large and continuous spaces which requires good feature selection.
- Hand engineering features results in poor generalization in an agent across domains.
- We use kernels to automatically linearize a non-linear problem.
Motivation

- We are primarily interested in reinforcement learning in large and continuous spaces which requires good feature selection.
- Hand engineering features results in poor generalization in an agent across domains.
- We use kernels to automatically linearize a non-linear problem.
- We introduce the first memory efficient kernel TD algorithm which allows for eligibility traces.
Motivation

- We are primarily interested in reinforcement learning in large and continuous spaces which requires good feature selection.
- Hand engineering features results in poor generalization in an agent across domains.
- We use kernels to automatically linearize a non-linear problem.
- We introduce the first memory efficient kernel TD algorithm which allows for eligibility traces with sparsification.
- Furthermore, this is a surprisingly easy to implement algorithm which gives a nice interpretation of the eligibility trace.
Several previous methods have been proposed.
Several previous methods have been proposed.

KLSTD is an interesting offline algorithm for offline policy evaluation, extending LSTD to kernel learning.
Several previous methods have been proposed.

KLSTD is an interesting offline algorithm for offline policy evaluation, extending LSTD to kernel learning.

KLSPI was introduced to do policy iteration as an extension of KLSTD, still in the offline (or batch) setting.
Several previous methods have been proposed.

KLSTD is an interesting offline algorithm for offline policy evaluation, extending LSTD to kernel learning.

KLSPI was introduced to do policy iteration as an extension of KLSTD, still in the offline (or batch) setting.

Gaussian Processes TD learning was proposed to do online kernel TD learning.
Several previous methods have been proposed.

KLSTD is an interesting offline algorithm for offline policy evaluation, extending LSTD to kernel learning.

KLSPI was introduced to do policy iteration as an extension of KLSTD, still in the offline (or batch) setting.

Gaussian Processes TD learning was proposed to do online kernel TD learning.

These works proposed novel kernel algorithms with novel tricks for memory efficiency.
Several previous methods have been proposed.

KLSTD is an interesting offline algorithm for offline policy evaluation, extending LSTD to kernel learning.

KLSPI was introduced to do policy iteration as an extension of KLSTD, still in the offline (or batch) setting.

Gaussian Processes TD learning was proposed to do online kernel TD learning.

These works proposed novel kernel algorithms with novel tricks for memory efficiency.

They do not allow for eligibility trace.
Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)
Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)
- $\langle S, A, R, T, \gamma \rangle$
Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)
- $\langle S, A, R, T, \gamma \rangle$
- State space $S$, action space $A$
Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)
- $\langle S, A, R, T, \gamma \rangle$
- State space $S$, action space $A$
- $T : S \times A \times S \rightarrow [0, 1]$ is transition function where $T(s, a, s')$ defines probability of transitioning from state $s$ to $s'$ through action $a$
Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)
- $\langle S, A, R, T, \gamma \rangle$
- State space $S$, action space $A$
- $T : S \times A \times S \rightarrow [0, 1]$ is transition function where $T(s, a, s')$ defines probability of transitioning from state $s$ to $s'$ through action $a$
- $R : S \times A \times S \rightarrow \mathbb{R}$ is a (possibly stochastic) reward function
Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)
- \(\langle S, A, R, T, \gamma \rangle\)
- State space \(S\), action space \(A\)
- \(T : S \times A \times S \rightarrow [0, 1]\) is transition function where \(T(s, a, s')\) defines probability of transitioning from state \(s\) to \(s'\) through action \(a\)
- \(R : S \times A \times S \rightarrow \mathbb{R}\) is a (possibly stochastic) reward function
- \(r_t = R(s_t, a_t, s'|s' = s_{t+1})\) defines the reward when action \(a\) in state \(s\) results in transition to state \(s'\)
Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)
- $\langle S, A, R, T, \gamma \rangle$
- State space $S$, action space $A$
- $T: S \times A \times S \rightarrow [0, 1]$ is transition function where $T(s, a, s')$ defines probability of transitioning from state $s$ to $s'$ through action $a$
- $R: S \times A \times S \rightarrow \mathbb{R}$ is a (possibly stochastic) reward function
- $r_t = R(s_t, a_t, s'|s' = s_{t+1})$ defines the reward when action $a$ in state $s$ results in transition to state $s'$
- $R_t$ denotes return at time $t$ which gives expected infinite discounted total reward given by $\sum_{i=t}^{\infty} \gamma^{i-t}r_t$, and $0 < \gamma < 1$
Markov Decision Processes

- We assume a (finite, countable infinite, or even continuous) Markov decision process (MDP)
- $\langle S, A, R, T, \gamma \rangle$
- State space $S$, action space $A$
- $T : S \times A \times S \rightarrow [0, 1]$ is transition function where $T(s, a, s')$ defines probability of transitioning from state $s$ to $s'$ through action $a$
- $R : S \times A \times S \rightarrow \mathbb{R}$ is a (possibly stochastic) reward function
- $r_t = R(s_t, a_t, s'|s' = s_{t+1})$ defines the reward when action $a$ in state $s$ results in transition to state $s'$
- $R_t$ denotes return at time $t$ which gives expected infinite discounted total reward given by $\sum_{i=t}^{\infty} \gamma^{i-t} r_t$, and $0 < \gamma < 1$
- Assume first order Markov property. ie. $(s_{t+1}, a_{t+1}, r_{t+1})$ is independent of $(s_{t-1}, a_{t-1}, r_{t-1})$ given $(s_t, a_t, r_t)$
SARSA(\(\lambda\))

- For larger MDPs, SARSA (\(\lambda\)) is performed using linear function approximation:
For larger MDPs, SARSA (λ) is performed using linear function approximation:

\[ Q(s, a) = \langle w, \phi(s, a) \rangle \]  \hspace{1cm} (1)
SARSA(\(\lambda\))

- For larger MDPs, SARSA (\(\lambda\)) is performed using linear function approximation:
  \[
  Q(s, a) = \langle w, \phi(s, a) \rangle
  \]  

- Traditional update rule for SARSA (\(\lambda\)) using function approximation with regularizer is
SARSA(λ)

- For larger MDPs, SARSA (λ) is performed using linear function approximation:

\[ Q(s, a) = \langle w, \phi(s, a) \rangle \]  

(1)

- Traditional update rule for SARSA (λ) using function approximation with regularizer is

\[ w_{t+1} = w_t - \eta_t \left[ err(s_t, a_t, R_t)e_t - \xi w_t \right] \]  

(2)
For larger MDPs, SARSA (λ) is performed using linear function approximation:

\[ Q(s, a) = \langle w, \phi(s, a) \rangle \]  

(1)

Traditional update rule for SARSA (λ) using function approximation with regularizer is

\[ w_{t+1} = w_t - \eta_t \left[ err(s_t, a_t, R_t)e_t - \xi w_t \right] \]  

(2)

Where \( err(s_t, a_t, R_t) = (Q(s_t, a_t) - R_t) \) and \( R_t = r_t + \gamma Q(s_{t+1}, a_{t+1}) \)
SARSA(λ)

- Where $e_t$ is updated through
SARSA(\(\lambda\))

- Where \(e_t\) is updated through

\[
e_t := \gamma \lambda e_{t-1} + \phi(s_t, a_t), \quad \phi(s, a) = k((s, a), \cdot) \tag{3}
\]
SARSA($\lambda$)

- Where $e_t$ is updated through

$$e_t := \gamma \lambda e_{t-1} + \phi(s_t, a_t), \quad \phi(s, a) = k((s, a), \cdot)$$  

- And is set to $\vec{0}$ at the beginning of each episode
**SARSA(λ)**

- Where $e_t$ is updated through

$$
e_t := \gamma \lambda e_{t-1} + \phi(s_t, a_t), \quad \phi(s, a) = k((s, a), \cdot)$$

(3)

- And is set to $\tilde{0}$ at the beginning of each episode
- Equivalently
SARSA(λ)

- Where $e_t$ is updated through

$$e_t := \gamma \lambda e_{t-1} + \phi(s_t, a_t), \phi(s, a) = k((s, a), \cdot)$$

(3)

- And is set to 0 at the beginning of each episode

- Equivalently

$$e_t := \sum_{i=t_0}^{t} (\gamma \lambda)^{t-i} \phi(s_i, a_i).$$

(4)
SARSA(λ)

- Where $e_t$ is updated through

$$e_t := \gamma \lambda e_{t-1} + \phi(s_t, a_t), \; \phi(s, a) = k((s, a), \cdot) \tag{3}$$

- And is set to $\tilde{0}$ at the beginning of each episode

- Equivalently

$$e_t := \sum_{i=t_0}^{t} (\gamma \lambda)^{t-i} \phi(s_i, a_i). \tag{4}$$

- Where $t_0$ is the time at which the current episode began
SARSA($\lambda$)

- Where $e_t$ is updated through

$$e_t := \gamma \lambda e_{t-1} + \phi(s_t, a_t), \quad \phi(s, a) = k((s, a), \cdot)$$

(3)

- And is set to $\vec{0}$ at the beginning of each episode
- Equivalently

$$e_t := \sum_{i=t_0}^{t} (\gamma \lambda)^{t-i} \phi(s_i, a_i).$$

(4)

- Where $t_0$ is the time at which the current episode began
- Typically such a representation would be undesirable since it requires storing all past samples
SARSA(λ)

- Where $e_t$ is updated through

$$e_t := \gamma \lambda e_{t-1} + \phi(s_t, a_t), \quad \phi(s, a) = k((s, a), \cdot)$$

(3)

- And is set to $\tilde{0}$ at the beginning of each episode

- Equivalently

$$e_t := \sum_{i=t_0}^{t} (\gamma \lambda)^{t-i} \phi(s_i, a_i).$$

(4)

- Where $t_0$ is the time at which the current episode began

- Typically such a representation would be undesirable since it requires storing all past samples

- For now let's assume that kernalizing our algorithm means storing all previously visited state action pairs anyway!
We now do two things:

- We substitute the summed form of the eligibility trace into the update equation.
- We note that by similarly summing the updates of $\theta$ we get:

$$
\theta_t = \sum_{i=1}^{t} \alpha_i \phi(s_i, a_i) = \sum_{i=1}^{t} \alpha_i \kappa((s_i, a_i), \cdot)
$$

By doing this, we get nice update equations for the new dual parameters $\alpha$:

$$
\alpha_t^{\pi_i} = (1 - \eta \xi) \alpha_i + \eta t_{\text{err}}(s_t, a_t, R_t) \gamma \lambda
$$

where $t_0$ is the time at which the current episode began.
**RKHS-SARSA(λ)**

- We now do two things:
  - We substitute the the summed form of the eligibility trace into the update equation, and
We now do two things:

- We substitute the summed form of the eligibility trace into the update equation, and
- We note that by similarly summing the updates of $\theta$ we get

$$\theta_t = \sum_{i=1}^{t} \alpha_i \phi(s_i, a_i) = \sum_{i=1}^{t} \alpha_i k((s_i, a_i), \cdot)$$
We now do two things:

- We substitute the summed form of the eligibility trace into the update equation, and
- We note that by similarly summing the updates of \( \theta \) we get
  \[
  \theta_t = \sum_{i=1}^{t} \alpha_i \phi(s_i, a_i) = \sum_{i=1}^{t} \alpha_i k((s_i, a_i), \cdot)
  \]

By doing this we get nice update equations for the new dual parameters \( \alpha \):
We now do two things:

- We substitute the summed form of the eligibility trace into the update equation, and
- We note that by similarly summing the updates of $\theta$ we get

$$\theta_t = \sum_{i=1}^{t} \alpha_i \phi(s_i, a_i) = \sum_{i=1}^{t} \alpha_i k((s_i, a_i), \cdot)$$

By doing this we get nice update equations for the new dual parameters $\alpha$:

$$\alpha'_i = (1 - \eta \xi) \alpha_i, i = 1, \ldots, t_0 - 1$$  \hspace{1cm} (5)

$$\alpha'_i = (1 - \eta \xi) \alpha_i - \eta_{t \text{err}}(s_t, a_t, R_t)(\gamma \lambda)^{t-i-1}, i = t_0, \ldots, t - 1$$  \hspace{1cm} (6)

$$\alpha'_t = \eta_{t \text{err}}(s_t, a_t, R_t)$$  \hspace{1cm} (7)
**RKHS-SARSA(λ)**

- We now do two things:
  - We substitute the summed form of the eligibility trace into the update equation, and
  - We note that by similarly summing the updates of $\theta$ we get
    \[ \theta_t = \sum_{i=1}^{t} \alpha_i \phi(s_i, a_i) = \sum_{i=1}^{t} \alpha_i k((s_i, a_i), \cdot) \]

- By doing this we get nice update equations for the new dual parameters $\alpha$:
  
  \[ \alpha'_i = (1 - \eta \xi) \alpha_i, i = 1, \ldots, t_0 - 1 \]  
  \[ \alpha'_i = (1 - \eta \xi) \alpha_i - \eta_t err(s_t, a_t, R_t)(\gamma \lambda)^{t-i-1}, i = t_0, \ldots, t - 1 \]  
  \[ \alpha'_t = \eta_t err(s_t, a_t, R_t). \]

where $t_0$ is the time at which the current episode began.
This provides the foundations for a powerful kernel based reinforcement learning algorithm.
This provides the foundations for a powerful kernel based reinforcement learning algorithm.

Number of samples grows linearly with time. PROBLEM!!!
This provides the foundations for a powerful kernel based reinforcement learning algorithm.

Number of samples grows linearly with time. PROBLEM!!!

We use ideas from the projectron method of Orabona et. al to make our algorithm more efficient in memory.
Before adding new sample, we ask ourselves:
Before adding new sample, we ask ourselves:

- How well can this new sample be represented as a linear combination of old ones
Before adding new sample, we ask ourselves:

- How well can this new sample be represented as a linear combination of old ones
- For poly kernels, in fact, we will eventually span the RKHS and never need to add new samples
Before adding new sample, we ask ourselves:

- How well can this new sample be represented as a linear combination of old ones
- For poly kernels, in fact, we will eventually span the RKHS and never need to add new samples

Rather than storing all new samples, consider projecting the newest hypothesis in $\mathcal{H}_t$ onto $\mathcal{H}_{t-1}$
Projectron RKHS-SARSA(\(\lambda\))

Now rather than updating the \(Q\) function immediately, we consider the projection of \(Q_{t+1}\) onto \(\mathcal{H}_{t-1}\)
Projectron RKHS-SARSA($\lambda$)

- Now rather than updating the $Q$ function immediately, we consider the projection of $Q_{t+1}$ onto $\mathcal{H}_{t-1}$
- Take “temporal hypothesis” $Q'_t = Q_{t+1}$ and its projection $Q''_t = P_{t-1}Q'_t$
Projectron RKHS-SARSA(\(\lambda\))

- Now rather than updating the \(Q\) function immediately, we consider the projection of \(Q_{t+1}\) onto \(H_{t-1}\)
- Take “temporal hypothesis” \(Q'_t = Q_{t+1}\) and its projection \(Q''_t = P_{t-1}Q'_t\)
- Using linear projection operator \(P_{t-1}\)
Now rather than updating the $Q$ function immediately, we consider the projection of $Q_{t+1}$ onto $\mathcal{H}_{t-1}$.

- Take “temporal hypothesis” $Q'_t = Q_{t+1}$ and its projection $Q''_t = P_{t-1}Q'_t$
- Using linear projection operator $P_{t-1}$

**Figure:** Projection of temporal hypothesis onto lower RKHS.
Dealing With the Eligibility Trace

- This now, however, breaks our previous vital assumption on the eligibility trace that we store all previous samples.
Dealing With the Eligibility Trace

- This now, however, breaks our previous vital assumption on the eligibility trace that we store all previous samples.
- Realize that the eligibility trace is now an eligibility function in $\mathcal{H}_k$ given by

\[
e_t = \sum_{i=0}^{T} \beta_i k((s_i, a_i), \cdot)
\]
Dealing With the Eligibility Trace

- This now, however, breaks our previous vital assumption on the eligibility trace that we store all previous samples.
- Realize that the eligibility trace is now an eligibility function in $\mathcal{H}_k$ given by

$$e_t := \sum_{i=t_0}^{t} \beta_i k((s_i, a_i), \cdot)$$  \hspace{1cm} (8)

Where $\beta$ is a second set of dual variables. Now we can also perform the projection method on the eligibility trace.
Dealing With the Eligibility Trace

- This now, however, breaks our previous vital assumption on the eligibility trace that we store all previous samples.
- Realize that the eligibility trace is now an eligibility function in $\mathcal{H}_k$ given by

$$e_t := \sum_{i=t_0}^{t} \beta_i k((s_i, a_i), \cdot)$$  \hspace{1cm} (8)

- Where $\beta$ is a second set of dual variables.
Dealing With the Eligibility Trace

- This now, however, breaks our previous vital assumption on the eligibility trace that we store all previous samples.
- Realize that the eligibility trace is now an eligibility function in $\mathcal{H}_k$ given by

$$e_t := \sum_{i=t_0}^{t} \beta_i k((s_i, a_i), \cdot)$$

(8)

- Where $\beta$ is a second set of dual variables.
- Now we can also perform the projectron method on the eligibility trace.
Dealing With the Eligibility Trace

- Our new update equations are given by

\[ \alpha'_i = (1 - \eta \xi) \alpha_i - \eta \text{err}(s_t, a_t, R_t) \gamma \lambda \beta_i, \quad \text{for } i = 1, \ldots, |S| \quad (9) \]
Dealing With the Eligibility Trace

- Our new update equations are given by

\[ \alpha'_i = (1 - \eta \xi) \alpha_i - \eta \text{err}(s_t, a_t, R_t) \gamma \lambda \beta_i, \quad \text{for } i = 1, \ldots, |\mathcal{S}| \]  

(9)

and
Dealing With the Eligibility Trace

- Our new update equations are given by

\[
\alpha'_i = (1 - \eta \xi) \alpha_i - \eta \text{err}(s_t, a_t, R_t) \gamma \lambda \beta_i, \quad \text{for } i = 1, \ldots, |\mathcal{S}| \tag{9}
\]

- and

\[
\beta'_i = \gamma \lambda \beta_i + d_i, \quad \text{for } i = 1, \ldots, |\mathcal{S}|. \tag{10}
\]

- If \( \delta_t < \epsilon \) where \( \delta \) is the norm of the difference between the temporal hypothesis and its projection.

- Moreover, \( d_i \)'s are the parameters of the projection and \( |\mathcal{S}| \) is the support set of stored basis functions.
Projectron RKHS-SARSA(\(\lambda\)) Updates

- If \(\delta_t > \epsilon\) we use the old updates for \(\alpha\)
If $\delta_t > \epsilon$ we use the old updates for $\alpha$

$$
\alpha'_i = (1 - \eta \xi) \alpha_i \quad i = 1, \ldots, t_0 - 1 
$$

$$
\alpha'_i = (1 - \eta \xi) \alpha_i - \eta t \text{err}(s_t, a_t, R_t) (\gamma \lambda) \beta_i \quad i = t_0, \ldots, |S|
$$

$$
\alpha'_{|S|+1} = \eta t \text{err}(s_t, a_t, R_t).
$$

and simply update $\beta$ through $\beta'_i = \gamma \lambda \beta_i$ for $i = 1, \ldots, |S|$ and $\beta_{|S|+1} = 1$
Figure: Moving average time per episode with window 10 evaluated for various algorithms at the end of each episode on mountain car.
Mountain Car

Figure: Moving average time per episode with window 10 evaluated for our algorithm with various values of $\lambda$ on the mountain car problem 2.
Figure: Moving average time per episode with window 10 evaluated for various algorithms at the end of each episode on the cart pole problem.
Figure: Moving average time per episode with window 10 evaluated for our algorithm with various values of $\lambda$ on the cart pole problem.
Memory Efficiency

Figure: Number of samples stored by the memory efficient version of our algorithm on each problem.
Algorithm In Summary

- Novel easy to implement algorithm with nice update equations
Algorithm In Summary

- Novel easy to implement algorithm with nice update equations
- Nice way to constrain memory growth
Algorithm In Summary

- Novel easy to implement algorithm with nice update equations
- Nice way to constrain memory growth
- First online kernel TD algorithm to incorporate eligibility traces.
QUESTIONS???