STOCHASTIC KERNEL TEMPORAL DIFFERENCE FOR REINFORCEMENT LEARNING

Jihye Bae[1], Luis Sanchez Giraldo[1], Pratik Chhatbar[2], Joseph Francis[2], Justin Sanchez[3], Jose Principe[1]

[1]Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611
[2]Department of Physiology, SUNY Downstate Medical Center, Brooklyn, NY 11203
[3]Department of Biomedical Engineering, University of Miami, Miami, FL 33146

ABSTRACT

This paper introduces a kernel adaptive filter using the stochastic gradient on temporal differences, kernel TD(λ), to estimate the state-action value function Q in reinforcement learning. Kernel methods are powerful for solving nonlinear problems, but the growing computational complexity and memory size limit their applicability on practical scenarios. To overcome this, the quantization approach introduced in [1] is applied. To help understand the behavior and illustrate the role of the parameters, we apply the algorithm on a 2-dimentional spatial navigation task. Eligibility traces are commonly applied in TD learning to improve data efficiency, so the relations of eligibility trace λ and step size and filter size are observed. Moreover, kernel TD (0) is applied to neural decoding of an 8 target center-out reaching task performed by a monkey. Results show the method can effectively learn the brain-state action mapping for this task.

Index Terms— Temporal difference learning, kernel methods, reinforcement learning, adaptive filtering

1. INTRODUCTION

In last two decades kernel methods have been subject of growing interest within the machine learning community. One of the major appeals of kernel methods is the ability to handle nonlinear operations on the data by indirectly computing an underlying nonlinear mapping to a representation space (Reproducing Kernel Hilbert Spaces (RKHS)) where linear operations can be carried out. These linear solutions correspond to universal approximation in the input space, and many of the related optimization problems can be posed as convex (no local minima) with algorithms that are still reasonably easy to compute (using the kernel trick [2]).

The introduction of the support vector machine algorithm for pattern recognition [3] rekindled the interest on kernel methods, and nowadays, kernel methods have become standard tools for supervised and unsupervised learning problems [4, 5, 6]. This increasing popularity has also motivated recent attempts to apply kernel methods to reinforcement learning (RL) [7, 8, 9, 10, 11, 12]. Previous work demonstrated the advantage of kernel based applications in reinforcement learning. A kernel approach to TD (0) learning employing Gaussian processes and kernel recursive least squares are introduced in [9], and the extended model for eligibility traces, the kernel based least-squares TD(λ) learning, is presented in [10]. However, previous work did not develop the stochastic gradient versions for TD(λ) learning.

Recent work has demonstrated the usefulness of kernel methods in solving nonlinear adaptive filtering problems [13] where even purely stochastic approximation such as kernel least mean squares (KLMS) can perform well with low computational complexity [6]. Although, the standard setting for kernel adaptive filtering algorithms differs from RL, elements such as the adaptive gain on an error term can be well exploited in solving RL problems as we shall see below. Despite of the advantages of using kernel methods, this approach is not widely followed in reinforcement learning community because of the high computational cost that comes from the inherent growing structure. Overcoming this limitation has become an active research topic [1, 14, 15]. In our work, we adopt the quantization approach introduced in [1] to make kernel RL feasible.

In this paper, we propose a learning approach for RL using kernels, trained with the least mean square (LMS) algorithm and TD learning called kernel Temporal Difference (TD)(λ) for estimation of Q function. Our algorithm shares many features with the KLMS presented in [6] except that the error is obtained using the temporal difference framework, i.e. the difference of consecutive outputs is used as the error.

To test the efficacy of the proposed method, the performance on a 2-dimensional spatial navigation task is observed. The experiment is also aimed at understanding the behavior and illustrating the role of the algorithm parameters. To understand this model fully, especially the relation of eligibility trace (λ) and step size and filter size, experimental evaluation of different configurations in the parameter set were observed. In addition, we test the algorithm in a signal processing application for brain machine interfaces; namely, kernel TD (0) is applied to

neural decoding of an 8 target center-out reaching task performed by a monkey. Experimental results show the method's can effectively learn the brain-state action mapping for this task.

2. KERNEL TEMPORAL DIFFERENCE (λ)

2.1. Reproducing kernel and the kernel LMS algorithm

Here, we provide a brief review of the main concepts behind kernel methods and a short derivation of the kernel least mean square (KLMS) algorithm.

Let $\mathcal{X}$ be a set. For a positive definite function $\kappa: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ [2], there exists a Hilbert space $\mathcal{H}$ and a mapping $\phi: \mathcal{X} \to \mathcal{H}$ such that $\kappa(x, y) = \langle \phi(x), \phi(y) \rangle$ for $x, y \in \mathcal{X}$. The space $\mathcal{H}$ is called a reproducing kernel Hilbert space (RKHS) satisfying for $f \in \mathcal{H}$

$$f(x) = \langle f, \phi(x) \rangle = \langle f, \kappa(x, \cdot) \rangle . \quad (1)$$

The KLMS algorithm attempts to minimize the risk functional $E\left[(d - f(x))^2\right]$ by minimizing the empirical risk $J(f) = \frac{1}{N}\sum_{i=1}^{N}(d(i) - f(x(i)))^2$ on the space $\mathcal{H}$ defined by the kernel κ. Using (1), we can rewrite

$$J(f) = \frac{1}{N}\sum_{i=1}^{N}\left(d(i) - \langle f, \phi(x(i)) \rangle\right)^2 \quad (2)$$

By differentiating the empirical risk $J(f)$ with respect to f and approximating the sum by the current difference (stochastic gradient), we can derive the following update rule:

$$f_i = f_{i-1} + \eta e(i)\phi(x(i)) \quad (3)$$

where $e(i) = d(i) - f_{i-1}(x(i))$ and $f_0 = 0$ which corresponds to the KLMS algorithm [6].

2.2. Kernel temporal difference (TD) (λ)

As in [16], consider the multistep prediction problem in supervised learning, for which we have the observed input sequence $x(1), x(2), \cdots, x(m)$ and desired d. Then, a system will produce a sequence of predictions based on the observed sequence. Notice that, in general, the predicted output is a function of all previous inputs:

$$y(i) = f_i(x(1), x(2), \cdots, x(i)) . \quad (4)$$

Here, we assume that $y(i) = f(x(i))$ for simplicity. Let the function f belong to an RKHS $\mathcal{H}$ as in KLMS. By treating the observed input sequence and the desired as a sequence of pairs $(x(1), d), (x(2), d), \cdots, (x(m), d)$, we can obtain the updates of function f after the whole sequence of m inputs has been observed as

$$f \leftarrow f + \eta\sum_{i=1}^{m}\Delta f_i , \quad (5)$$

where $\Delta f_i = \left(d - \langle f, \phi(x(i)) \rangle\right)\phi(x(i))$ are the instantaneous updates of the function f from input data based on (1). By taking $d \triangleq y(m+1)$ and representing the error as $d - y(i) = \sum_{k=i}^{m}\left(y(k+1) - y(k)\right)$, we can arrive at

$$f \leftarrow f + \eta\sum_{i=1}^{m}\left[\left(\langle f, \phi(x(i+1)) \rangle - \langle f, \phi(x(i)) \rangle\right)\sum_{k=1}^{i}\phi(x(k))\right] . \quad (6)$$

That is, $\Delta\tilde{f}_i = \langle f, \phi(x(i+1)) - \phi(x(i)) \rangle \sum_{k=1}^{i}\phi(x(k))$. Thus, generalizing for λ yields:

$$\Delta\tilde{f}_i = \langle f, \phi(x(i+1)) - \phi(x(i)) \rangle \sum_{k=1}^{i}\lambda^{i-k}\phi(x(k)). \quad (7)$$

This approach will be called Kernel TD(λ).

2.3. Q-learning via kernel temporal difference (λ)

Off-policy RL using Q-learning [17] updates the state-action value function Q as follows:

$$Q(x(i), a(i)) \leftarrow Q(x(i), a(i)) + \eta\left[r(i+1) + \gamma \max_a Q(x(i+1), a) - Q(x(i), a(i))\right] \quad (8)$$

to maximize the expected reward,

$$R^a_{xx'} = E\left[r(i+1) \mid x(i) = x, a(i) = a, x(i+1) = x'\right] . \quad (9)$$

So, we can set the desired output as the cumulative reward. For the optimal trained system, the output will equal the desired response, and the following relation will be satisfied:

$$d(i) = \sum_{k=0}^{\infty}\gamma^k r(i+k+1) = y(i) = r(i+1) + \gamma y(i+1) , \quad (10)$$

where, γ is discount-rate parameter with range [0,1], r is a reward value. Therefore, based on TD error $(r(i+1) + \gamma y(i+1)) - y(i)$, the generalized update for Q-learning via Kernel TD(λ) has the form,

$$\Delta\tilde{f}_i = \left(r(i+1) + \langle f, \gamma\phi(x(i+1)) - \phi(x(i)) \rangle\right)\sum_{k=1}^{i}\lambda^{i-k}\phi(x(k)) . \quad (11)$$

Notice that unlike the case of kernel LMS, the current estimate of the error propagates to the weights corresponding to the last i input points. In the case of $\lambda = 0$, the contribution of each observed input to the update equation becomes

$$\Delta\tilde{f}_i = \left(r(i+1) + \gamma\langle f, \phi(x(i+1)) \rangle - \langle f, \phi(x(i)) \rangle\right)\phi(x(i)) , \quad (12)$$

which in the case of single updates yields

$$Q_n(x(i)) = \eta\sum_{j=1}^{i-1} e_{TD}(j) I_k(j) \kappa(x(i), x(j)) . \quad (13)$$

Here, for discrete actions $Q_n(x(i)) = Q(x(i), a = n)$ and $I_k(i)$ is an indicator vector with the same size as the number of outputs which has only the k th value as 1 based on the selected action unit k at time i based on ε-greedy approach

$$I_k(i) = \begin{cases} 1, & k\text{-th action is chosen} \\ 0, & \text{otherwise.} \end{cases} \quad (14)$$

Therefore, only the weight corresponding to the winning action gets updated. Also, $e_{TD}(i)$ is TD error defined as

$$e_{TD}(i) = r_n + \gamma Q_m(x(i+1)) - Q_n(x(i)). \quad (15)$$

Recall, that the reward r_n corresponds to the action selected by the current policy with input $x(i)$ because it is assumed that this action causes the next input state $x(i+1)$.

The initial error is set to zero, and the first input vector is assigned as the first unit's center. The number of units (kernel evaluations) increase as new training data arrives, and each added unit is centered at the previous inputs $x(1), x(2), \cdots, x(i-1)$.

Each output component represents the possible action directions. Based on the outputs (Q-value), one action which has the maximum value is selected by ε-greedy method [17], and only the weights connected to selected action unit get updated.

2.4. Quantization approach

The quantization approach introduced in [1] is a simple yet effective approximation heuristic that limits the growing structure of the filter by adding units in a selective fashion. Once a new state input $x(i)$ arrives, its distances to each existing unit $C(i-1)$ are calculated

$$dist(x(i), C(i-1)) = \min_{1 \le j \le size(C(i-1))} \left\| x(i) - C_j(i-1) \right\|. \quad (16)$$

If the minimum distance $dist(x(i), C(i-1))$ is smaller than the quantization size ε_U, the new input state $x(i)$ is absorbed by the closest existing unit to it, and hence no new unit is added to the structure. In this case, unit centers remain the same $C(i) = C(i-1)$, but the connection weights to the closest unit are updated.

3. EXPERIMENTS AND RESULTS

3.1. Two dimensional spatial navigation task

In this example, we have 2-dimensional state-space that corresponds to a square of 20 unit side-length. The goal is navigate from any position on the square to a target located within the square. In our experiments, one target is located at the center of the square (10,10) and any approximations within a 2 unit radius are considered successful. A fixed set of 25 points distributed in lattice configuration are taken as initial seeds for random initial states. Each initial state corresponds to drawing randomly one of these 25 points with equal probability. The location of the selected point is further perturbed with Gaussian noise with unit variance. To navigate through the state space, we can choose to move 3 units of length in one of the 8 possible directions allowed. The maximum number of steps per trial is limited to 20.

The agent gets a reward +0.6 every time it reaches the goal and a new trial starts. Otherwise, a punishment value of -0.6 is given. Exploration rate of 0.05 and discount factor $\gamma = 0.9$ are used. The kernel employed is Gaussian $\kappa(x, x') = \exp\left(-\left\|x - x'\right\|^2 / 2h^2\right)$ with size $h = 4$. This kernel size is selected based on the distribution of squared distance between pairs of input states.

To assess the performance of the algorithm, we count the cumulated reward within a group of 25 trials, that is, every 25 trials, we calculate the success rate of the learned mapping as $N_{Success}/25$. With a fixed kernel size of 4, the performance over the various stepsizes ($\eta = 0.01, 0.1 \sim 0.9$ with 0.1 intervals) and values of eligibility trace ($\lambda = 0, 0.2, 0.5, 0.8, 1$) are observed.

The stepsize η mainly affects the speed of learning, and within the stability limits, larger stepsizes provide faster convergence. However, due to the effect of eligibility trace λ, the stability limits suggested in [13] must be adjusted accordingly:

$$\eta < \frac{N}{tr[G_\varphi]} = \frac{N}{\sum_{j=1}^{N} \kappa(x(j), x(j))} = \frac{N}{N(\lambda^0 + \cdots + \lambda^{m-1})}. \quad (17)$$

This upper bound assumes that the maximum number of steps per trial m has been reached. Hence, for $\lambda = 0, 0.2, 0.5, 0.8, 1$, the stable stepsizes η lie below 1, 0.8, 0.5, 0.2, and 0.05 respectively.

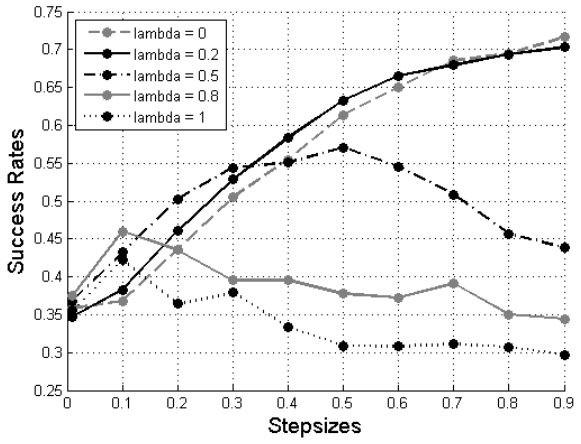

Fig. 1. The average success rates over 125 trials and 50 implementations.

Figure 1 shows the average performance over 125 trials and 50 implementations for different combination of stepsizes η and eligibility trace λ.

The trade-off between stepsize and eligibility trace can be observed from the plot. It is clear how these two factors can be associated with the speed of learning. At intermediate points on the horizontal axis, the influence of λ, becomes

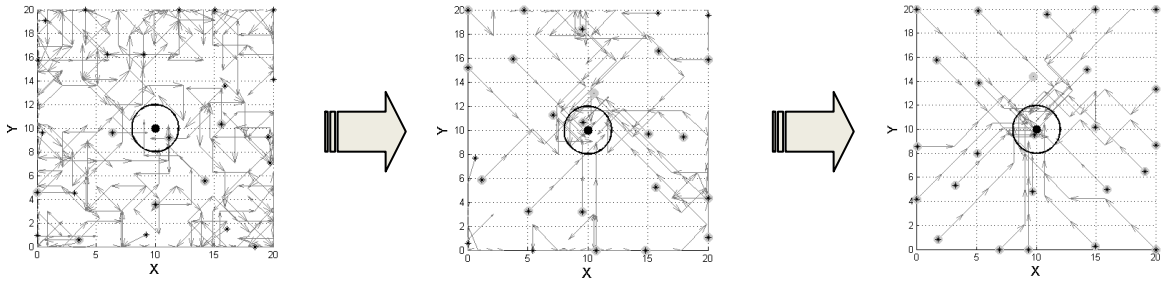

Figure 3. Two dimensional state transitions of the first, third, and fifth sets with $\eta = 0.9$ and $\lambda = 0$. The black starts represent the 25 initial states, and gray arrows shows the action chosen at each state. Black dot at the center is the target and black circle shows the reward zone.

more evident as the stepsize decreases. On the other hand, if the stepsize increases, we can see how performance degrades for larger values of λ since the bound set by (17) is not satisfied.

The relation between final filter size versus stepsize and eligibility trace rate is also observed (Figure 2).

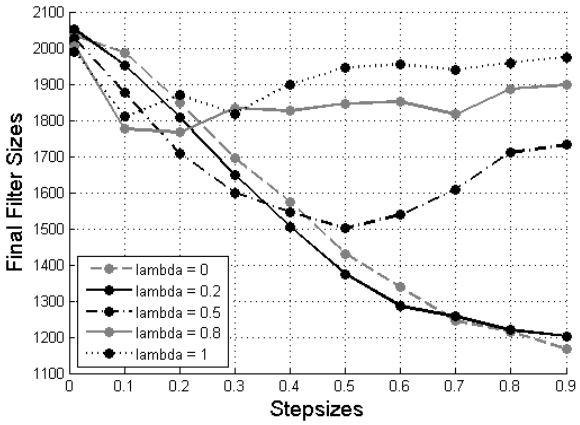

Fig. 2. The average filter size after 125 trials over 50 implementations.

Since each trial allows maximum number of 20 steps, the largest final filter size is 2500. However, with a good adaptive system, the final filter size can be reduced. The final filter size matches with the success rates. High success rates mean that a system has learnt the state-action mapping, whereas a system that has not adapted to new environment, keeps exploring the space.

Both average success rate and final filter size show that stepsize $\eta = 0.9$ and eligibility trace $\lambda = 0$ have the best performance, and with the selected values, the success rates approach to over 95% after 100 trials.

From Figure 3, we can observe how learning is accomplished. At the beginning, the system explores more the spatial space based on the reward information and trajectories looks rather erratic. Once the system starts learning, actions corresponding to states near target point toward the reward zone, and as time goes by this area becomes larger and larger until it covers the whole state space.

Here, we show how the quantization approach presented [1] can be employed to ameliorate the limitations imposed by growing filter sizes. For a fixed set of 125 inputs, we consider quantization sizes ε_U : 40, 30, 20, 10, 5, 2, and 1.

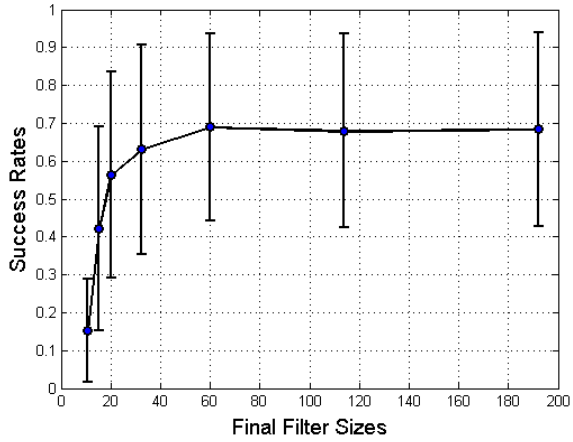

Fig. 4. The average success rates over 125 trials and 50 implementations with respect to different filter sizes.

Figure 4 shows the effect of different quantization sizes on the final performance. Notice that the minimum size for stable performance of the filter is reached around approximately 60 units. Therefore, the quantization size

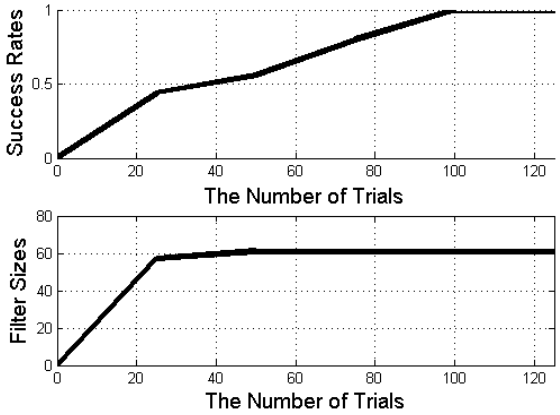

Fig. 5. The change of success rates (top) and final filter size (bottom) with $\varepsilon_U = 5$.

$\varepsilon_U = 5$ can be selected, and the maximum success rate is still being achieved (see Figure 5).

3.2. Eight target center-out reaching task

Neural decoding to find an optimal functional mapping between neural states and action directions is implemented with reinforcement learning. In this RLBMI framework, an agent learns how to transfer the neural states into actions based on predefined reward values from the environment. The agent must interpret the subject's brain activity correctly to facilitate the rewards [18, 19].

A female bonnet macaque is trained for a center-out reaching task allowing 8 action directions. After about 80% success rate, microelectrode arrays are implanted in motor cortex (M1). In this implementation, only the successful trials are used for neural decoding (total number of 178 trials) with 185 units obtained from 96 channels. The neural decoding aims at 8 targets with allowing 8 possible action directions. Every trial starts at the center point, and the system learns to reach one assigned target per trial. The distance from the center to a target is 4cm with target radius of 0.8~1cm (Figure 6).

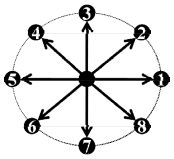

Fig.6. The center-out reaching task for 8 targets.

The neural states are represented by the firing rates of 185 units over a 100ms window and their past 6 values, creating a 7 dimensional time embedding per channel; this amounts to an input state vector of 1295 dimensions. After input states are preprocessed by normalizing their dynamic range between -1 and 1, they are input to the system.

Eight output units represent the possible action-direction, and the actions are selected according to the ε-greedy method [17]. Here, tests are carried out for a single-step reaching task towards the assigned target. So, after the single-step update of the cursor position, the reward is measured based on the Hamming distance to the target. Once the distance reaches the target, the agent earns a reward (+0.6), otherwise it receives punishment (-0.6) [20].

Based on the selected action with exploration rate 0.01 and the reward value, the system is adapted by Kernel TD(λ) with discount rate $\gamma = 0.9$. In our case, eligibility trace $\lambda = 0$ is specially considered since our experiment performs single step updates per trial. Here, stepsize $\eta = 0.5$ is selected based on [13] and Gaussian kernel is used. Without any mechanism to control the filter size, the filter structure can grow up to 1780 units with 10 epochs. Quantization approach is used to observe the effect of filter size. Intuitively, there is an intrinsic relation between quantization size ε_U and kernel size h. Consequently, based on the distribution of squared distance between pairs of input states, various kernel sizes ($h = 0.5, 1, 1.5, 2, 3, 5, 7$) and quantization sizes ($\varepsilon_U = 1, 110, 120, 130$) are tested. The corresponding success rates for final filter sizes of 178, 133, 87, and 32 are obtained (see Figure 7).

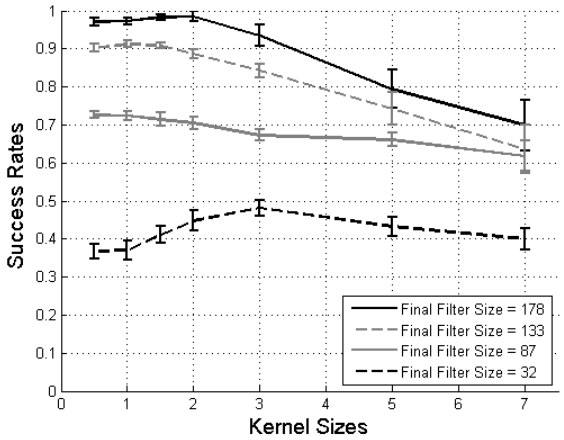

Fig. 7. The average success rates over 50 implementations after 10 epochs.

Since all the parameters are fixed over these 50 experiments, the narrow error bars show the effect of the random action selection for exploration. Here, it is noticeable that distinctly from traditionally TD learning such as neural nets, this kernel approach does not heavily depend on initialization. With a final filter size of 178 (solid black line) the success rates are superior to any other sizes all over the different kernel sizes. Especially for small kernel sizes ($h \le 2$), above 96% of success rates are observed. It supports the feature of kernel methods that more information of data improves the performance. In addition, even with reduction of the state information (dashed gray line), the system still produces acceptable success rates with kernel sizes from 0.5 to 2 (around 90%).

To obtain generalization, kernel size $h = 2$ is selected since it allows largest kernel size and reduced filter size with good neural state to action mapping (around 90%). In the case of kernel size $h = 2$ with final filter size of 178, the system reaches to 100% success rate after 6 epochs with maximum variance of 4%. To observe the learning process, the predicted Q-values are displayed after each trial, and for each epoch, success rates are calculated (i.e. 1 epoch containing 178 trials). Based on the neural states, the system generates corresponding Q-values, and in general, as shown in Figure 8, the maximum Q-values match with target index even with reduced filter sizes.

4. CONCLUSIONS AND FUTURE WORK

We introduced a novel approach using stochastic gradient approximation (LMS) using the TD framework for the

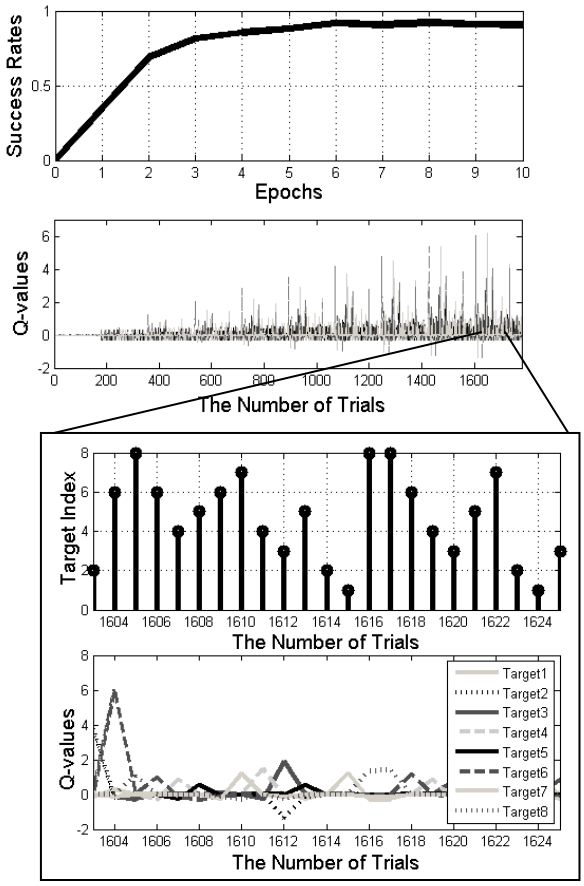

Fig. 8. The change of success rates and Q-values (up) and the matching of target index and Q-values (down) by Kernel TD(0) with kernel size $h = 2$ with filter size 133.

approximation of state-action value function Q in RL, kernel TD(λ).The algorithm was applied to a 2 dimensional spatial navigation task, and relations between the eligibility trace, stepsize and filter size were observed. The experiments showed how the parameter λ bounds the set of allowable stepsizes and the close relation between learning and final filter size. In addition, it was show how the quantization approach can be successfully applied to control the filter size. Results of the algorithm to solve an 8 target center-out reaching task performed by a monkey showed that the method is still feasible and performs well in a more realistic scenario. As future work, we propose to introduce a more systematical approach to optimal parameter selection, as well as more experiments that involve other capacity control methods.

5. ACKNOWLEDGEMENT

This work was partially supported by DARPA Contract N66001-10-C-2008. The authors would like to thank Brandi Marsh for collecting data for the 8 target center-out task.

6. REFERENCES

[1] B. Chen, S. Zhao, P. Zhu, and J. C. Principe, "Quantized Kernel Least Mean Square Algorithm," *IEEE computational Intelligence*, submitted for publication.

[2] B. Scholkopf and A.J. Smola, *Learning with Kernels.* The MIT Press, 2002.

[3] B. E. Boser, I. M. Guyon, and V. N. Vapnik, *A training algorithm for optimal margin classifiers*. 5th Annual ACM Workshop on COLT, PA: ACM Press, pp. 144-152, 1992.

[4] B. Scholkopf, S. Mika, C. J. C. Burges, P. Knirsch, K. R. Muller, G. Ratsch, and A. J. Smola, "Input Space Versus Feature Space in Kernel-Based Methods," *IEEE Trans. Neural Networks*, vol. 10, no. 3, pp. 1000-1017, 1999.

[5] F. R. Bach and M. I. Jordan, "Kernel Independent Component Analysis," *JMLR*, vol.3, pp. 1-48, 2002.

[6] W. Liu, P. P. Pokharel, and J. C. Principe, "The Kernel Least-Mean Square Algorithm," *IEEE Trans. Signal Processing*, vol. 56, no. 3, Feb. 2008.

[7] D. Ormoneit and S. Sen, "Kernel-Based Reinforcement Learning," Kluwer Academic Publishers, Netherlands, vol. 49, pp. 161-178, 2002.

[8] C. E. Rasmussen, M. Kuss, "Gaussian Process in Reinforcement Learning," *NIPS*, 2003.

[9] Y. Engel, "Algorithms and Representations for Reinforcement Learning," Ph.D. dissertation, School of Eng. and Computer Science, Hebrew Univ., Jerusalem, 2005.

[10] X. Xu, T. Xie, D. Hu, and X. Lu, "Kernel Least-Squares Temporal Difference Learning," *International Journal of Information Technology*, vol. 1, no. 9, 2005.

[11] N. K. Jong and T. Stone, "Kernel-Based Models for Reinforcement Learning," *kernel machines for reinforcement learning workshop*, Pittsburgh, PA, 2006.

[12] X. Xu, D. Hu, and X. Lu, "Kernel-Based Least Squares Policy Iteration for Reinforcement Learning," *IEEE Trans. Neural Network*, vol. 18, no. 4, 2007.

[13] W. Liu, J. C. Principe, and S. Haykin, *Kernel Adaptive Filtering*. Wiley, 2010.

[14] A. M. S. Barreto and D. Precup, "Kernel-Based Stochastic Factorization for Batch Reinforcement Learning," unpublished.

[15] W. Liu, I. Park, J. C. Principe, "An Information Theoretic Approach of Designing Sparse Kernel Adaptive Filters," *IEEE Trans. Neural Network*, vol. 20, no. 12, 2009.

[16] R. S. Sutton, "Learning to Predict by the Methods of Temporal Differences," Kluwer Academic Publishers, Boston, pp. 9-44, 1988.

[17] C. J. C. H. Watkins and P. Dayan, "Q-learning," in *Machine Learning*, vol. 8, num. 3-4, Springer, 1992, pp.279-292.

[18] J. C. Sanchez, "Co-adaptive Brain-Machine Interface via Reinforcement Learning," *IEEE Trans. Biomedical Engineering*, vol. 56, no. 1, Jan, 2009.

[19] J. Bae, P. Chhatbar, J. T. Francis, J. C. Sanchez, and J. C. Principe, "Reinforcement Learning via Kernel Temporal Difference," to appear *EMBC*, 2011.

[20] J. C. Sanchez, A. Tarigoppula, J. S. Choi, B. T. Marsh, P. Y. Chhatbar, B. Mahmoudi, and J. T. Francis, "Control of a Center-Out Reaching Task using a Reinforcement Learning Brain-Machine Interface," submitted for publication.