Hindawi Publishing Corporation
Computational Intelligence and Neuroscience
Volume 2015, Article ID 481375, 17 pages
http://dx.doi.org/10.1155/2015/481375

Research Article

Kernel Temporal Differences for Neural Decoding

Jihye Bae,[1] Luis G. Sanchez Giraldo,[1] Eric A. Pohlmeyer,[2] Joseph T. Francis,[3] Justin C. Sanchez,[2] and José C. Príncipe[1]

[1] *Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611, USA*
[2] *Department of Biomedical Engineering, University of Miami, Coral Gables, FL 33146, USA*
[3] *Department of Physiology and Pharmacology, Robert F. Furchgott Center for Neural & Behavioral Science, SUNY Downstate Medical Center, Brooklyn, NY 11203, USA*

Correspondence should be addressed to Jihye Bae; jbae1013@gmail.com

Received 8 September 2014; Revised 28 January 2015; Accepted 3 February 2015

Academic Editor: Daoqiang Zhang

We study the feasibility and capability of the kernel temporal difference (KTD)(λ) algorithm for neural decoding. KTD(λ) is an online, kernel-based learning algorithm, which has been introduced to estimate value functions in reinforcement learning. This algorithm combines kernel-based representations with the temporal difference approach to learning. One of our key observations is that by using strictly positive definite kernels, algorithm's convergence can be guaranteed for policy evaluation. The algorithm's nonlinear functional approximation capabilities are shown in both simulations of policy evaluation and neural decoding problems (policy improvement). KTD can handle high-dimensional neural states containing spatial-temporal information at a reasonable computational complexity allowing real-time applications. When the algorithm seeks a proper mapping between a monkey's neural states and desired positions of a computer cursor or a robot arm, in both open-loop and closed-loop experiments, it can effectively learn the neural state to action mapping. Finally, a visualization of the coadaptation process between the decoder and the subject shows the algorithm's capabilities in reinforcement learning brain machine interfaces.

1. Introduction

Research in brain machine interfaces (BMIs) is a multidisciplinary effort involving fields such as neurophysiology and engineering. Developments in this area have a wide range of applications, especially for subjects with neuromuscular disabilities, for whom BMIs may become a significant aid. Neural decoding of motor signals is one of the main tasks that needs to be executed by the BMI.

Ideas from system theory can be used to frame the decoding problem. Bypassing the body can be achieved by modelling the transfer function from brain activity to limb movement and utilizing the output of the properly trained model to control a robotic device to implement the intention of movement. The design of neural decoding systems has been approached using machine learning methods. In order to choose the appropriate learning method, factors such as learning speed and stability help in determining the usefulness of a particular method.

Reinforcement learning brain machine interfaces (RLBMI) [1] have been shown to be a promising avenue for practical implementations. Fast adaptation under changing environments and neural decoding capability of an agent have been shown in [2, 3] using the actor-critic paradigm. Adaptive classification of event-related potential (ERP) in electroencephalography (EEG) using RL in BMI was proposed in [4]. Moreover, partially observable Markov decision processes (POMDPs) have been applied in the agent to account for the uncertainty when decoding noisy brain signals [5]. In a RLBMI, a computer agent and a user in the environment cooperate and learn coadaptively. The decoder learns how to correctly translate neural states into action direction pairs that indicate the subject's intent. In the agent, the proper neural decoding of the motor signals is essential to control an external device that interacts with the physical environment.

However, to realize the advantages of RLBMIs in practice, there are several challenges that need to be addressed.

The neural decoder must be able to handle high-dimensional neural states containing spatial-temporal information. The mapping from neural states to actions must be flexible enough to avoid making strong assumptions. Moreover, the computational complexity of the decoder should be reasonable such that real time implementations are feasible.

Temporal difference learning provides an efficient learning procedure that can be applied to reinforcement learning problems. In particular, TD(λ) [6] can be applied to approximate a value function that is utilized to compute an approximate solution to Bellman's equation. The algorithm allows incremental computation directly from new experience without having an associated model of the environment. This provides a means to efficiently handle high-dimensional states and actions by using an adaptive technique for function approximation that can be trained directly from the data. Also, because TD learning allows system updates directly from the sequence of states, online learning is possible without having a desired signal at all times.

Note that TD(λ) and its variants (least squares TD(λ) [7], recursive least squares TD [8], incremental least squares TD(λ) [9], Gradient TD [10], and linear TD with gradient correction [11]) have been mostly treated in the context of parametric linear function approximation. This can become a limiting factor in practical applications where little prior knowledge can be incorporated. Therefore, here, our interest focuses on a more general class of models with nonlinear capabilities. In particular, we adopt a kernel-based function approximation methodology.

Kernel methods are an appealing choice due to their elegant way of dealing with nonlinear function approximation problems. Unlike most of the nonlinear variants of TD algorithms which are prone to fall into local minima [12, 13], the kernel based algorithms have nonlinear approximation capabilities yet the cost function can be convex [14]. One of the major appeals of kernel methods is the ability to handle nonlinear operations on the data by an implicit mapping to the so called *feature space* (reproducing kernel Hilbert space (RKHS)) which is endowed with an inner product. A linear operation in the RKHS corresponds to a nonlinear operation in the input space. In addition, algorithms based on kernel methods are still reasonably easy to compute based on the kernel trick [14].

Temporal difference algorithms based on kernel expansions have shown superior performance in nonlinear approximation problems. The close relation between Gaussian processes and kernel recursive least squares was exploited in [15] to provide a Bayesian framework for temporal difference learning. Similar work using kernel-based least squares temporal difference learning with eligibilities (KLSTD(λ)) was introduced in [16]. Unlike the Gaussian process temporal difference algorithm (GPTD), KLSTD(λ) is not a probabilistic approach. The idea in KLSTD is to extend LSTD(λ) [7] using the concept of duality. However, the computational complexity of KLSTD(λ) per time update is $O(n^3)$ which precludes its use for online learning.

An online kernel temporal difference learning algorithm called kernel temporal differences (KTD)(λ) was proposed in [17]. By using stochastic gradient updates, KTD(λ) reduces the computational complexity from $O(n^3)$ to $O(n^2)$. This reduction along with other capacity control mechanisms such as sparsification make real time implementations of KTD(λ) feasible.

Even though nonparametric techniques are inherently of growing structure, these techniques produce better solutions than any other simple linear function approximation methods. This has motivated work on methods that help overcome scalability issues such as the growing filter size [18]. In the context of kernel based TD algorithms, sparsification methods such as approximate linear dependence (ALD) [19] have been applied to GPTD [15] and KLSTD [20]. A Quantization approach proposed in [21] has been used in KTD(λ) [22]. In a similar flavor, the kernel distance based online sparsification method was proposed for a KTD algorithm in [23]. Note that ALD is $O(n^2)$ complexity, whereas quantization and kernel distances are $O(n)$. The main difference between the quantization approach and the kernel distance is the space where the distances are computed. Quantization approach uses criterion of input space distances whereas kernel distance computes them in the RKHS associated with the kernel [23].

In this paper, we investigate kernel temporal differences (KTD)(λ) [17] for neural decoding. We first show the advantages of using kernel methods. Namely, we show that convergence results of TD(λ) in policy evaluation carry over KTD(λ) when the kernel is strictly positive definite. Examples of the algorithm's capability for nonlinear function approximation are also presented. We apply the KTD algorithm to neural decoding in open-loop and closed-loop RLBMI experiments where the algorithm's ability to find proper neural state to action map is verified. In addition, the trade off between the value function estimation accuracy and computation complexity under growing filter size is studied. Finally, we provide visualizations of the coadaptation between the decoder and the subject highlighting the usefulness of KTD(λ) for reinforcement learning brain machine interfaces.

This paper starts with a general background on reinforcement learning which is given in Section 2. Section 3 introduces the KTD algorithm and provides its convergence properties for policy evaluation. This algorithm is extended in Section 4 to policy improvement using Q-learning. Section 5 introduces some of the kernel sparsification methods for the KTD algorithm that address the naturally growing structure of kernel adaptive algorithms. Section 6 shows empirical results on simulations for policy evaluation, and Section 7 presents experimental results and comparisons to other methods in neural decoding using real data sets for both, open-loop and closed-loop RLBMI frameworks. Conclusions are provided in Section 8.

2. Reinforcement Learning Brain Machine Interfaces and Value Functions

In reinforcement learning brain machine interfaces (RLBMI), a neural decoder interacts with environment over time and adjusts its behavior to improve performance [1].

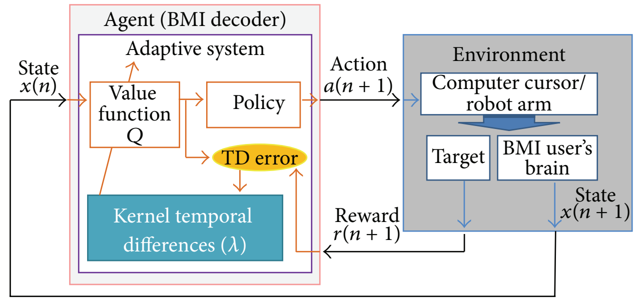


Figure 1: The decoding structure of reinforcment learning model in a brain machine interface using a Q-learning based function approximation algorithm.

The controller in the BMI can be considered as a neural decoder, and the environment includes the BMI user (Figure 1).

Assuming the environment is a stochastic and stationary process that satisfies the Markov condition, it is possible to model the interaction between the learning agent and the environment as a Markov decision process (MDP). For the sake of simplicity, we assume the states and actions are discrete, but they can also be continuous.

At time step n, the decoder receives the representation of the user's neural state $x(n) \in \mathscr{X}$ as input. According to this input, the decoder selects an action $a(n) \in \mathscr{A}$ which causes the state of the external device to change, namely, the position of a cursor on a screen or a robot's arm position. Based on the updated position, the agent receives a reward $r(n+1) \in \mathbb{R}$. At the same time, the updated position of the actuator will influence the user's subsequent neural states; that is, going from $x(n)$ to $x(n+1)$ because of the visual feedback involved in the process. The new state $x(n+1)$ follows the state transition probability $\mathscr{P}^{a}_{xx'}$ given the action $a(n)$ and the current state $x(n)$. At the new state $x(n+1)$, the process repeats; the decoder takes an action $a(n+1)$, and this will result in a reward $r(n+2)$ and a state transition from $x(n+1)$ to $x(n+2)$. This process continues either indefinitely or until a terminal state is reached depending on the process.

Note that the user has no direct access to actions, and the decoder must interpret the user's brain activity correctly to facilitate the rewards. Also, both systems act symbiotically by sharing the external device to complete their tasks. Through iterations, both systems learn how to earn rewards based on their joint behavior. This is how the two intelligent systems (the decoder and the user) learn coadaptively, and the closed loop feedback is created. This coadaptation allows for continuous synergistic adaptation between the BMI decoder and the user even in changing environments [1].

The value function is a measure of long-term performance of an agent following a policy π starting from a state $x(n)$. The state value function is defined as

$$V^{\pi}(x(n)) = E_{\pi}[\mathbf{R}(n) \mid x(n)], \tag{1}$$

and action value function is given by

$$Q^{\pi}(x(n), a(n)) = E_{\pi}[\mathbf{R}(n) \mid x(n), a(n)], \tag{2}$$

where $\mathbf{R}(n)$ is known as the return. Here, we apply a common choice for the return, the *infinite-horizon* discounted model

$$\mathbf{R}(n) = \sum_{k=0}^{\infty} \gamma^{k} r(n+k+1), \quad 0 < \gamma < 1 \tag{3}$$

that takes into account the rewards in the long run but weighs them with a discount factor to prevent the function from growing unbounded as $k \to \infty$ and provides mathematical tractability [24]. Note that our goal is to find a policy $\pi : \mathscr{X} \to \mathscr{A}$ which maps a state $x(n)$ to an action $a(n)$. Estimating the value function is an essential step towards finding a proper policy.

3. Kernel Temporal Difference(λ)

In this section, we provide a brief introduction to kernel methods followed by the derivation of the KTD algorithm [17, 22]. One of the contributions of the present work is the convergence analysis of KTD(λ) presented at the end of this section.

3.1. Kernel Methods.

Kernel methods are a family of algorithms for which input data are nonlinearly map to a high-dimensional feature space of vectors where linear operations are carried out. Let $\mathscr{X}$ be a nonempty set. For a positive definite function $\kappa : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$ [14, 25], there exists a Hilbert space $\mathscr{H}$ and a mapping $\phi : \mathscr{X} \to \mathscr{H}$ such that

$$\kappa(x, y) = \langle \phi(x), \phi(y) \rangle. \tag{4}$$

The inner product in the high-dimensional feature space can be calculated by evaluating the kernel function in the input space. Here, $\mathscr{H}$ is called a reproducing kernel Hilbert space (RKHS), for which the following property holds,

$$f(x) = \langle f, \phi(x) \rangle = \langle f, \kappa(x, \cdot) \rangle, \quad \forall f \in \mathscr{H}. \tag{5}$$

The mapping implied by the use of the kernel function κ can also be understood through Mercer's Theorem [26]. The implicit map ϕ allows one to transform conventional linear algorithms in the feature space to nonlinear systems in the input space, and the kernel function κ provides an implicit way to compute inner products in the RKHS without explicitly dealing with the high-dimensional space.

3.2. Kernel Temporal Difference(λ).

In the multistep prediction problem, we consider a sequence of input-output pairs $(x(1), d(1)), (x(2), d(2)), \ldots, (x(m), d(m))$, for which the desired output d is only available at time $m+1$. Consequently, the system should produce a sequence of predictions $y(1), y(2), \ldots, y(m)$ based solely on the observed input sequences before it gets access to the desired response. In general, the predicted output is a function of all previous inputs $y(n) = f(x(1), x(2), \ldots, x(n))$. Here, we assume that $y(n) = f(x(n))$ for simplicity, and let the function f belong to a RKHS $\mathscr{H}$.

In supervised learning, by treating the observed input sequence and the desired prediction as a sequence of pairs

$(x(1), d), (x(2), d), \ldots, (x(m), d)$ and making $d \triangleq y(m+1)$, we can obtain the updates of function f after the whole sequence of m inputs has been observed as

$$f \longleftarrow f + \sum_{n=1}^{m} \Delta f_n \tag{6}$$

$$= f + \eta \sum_{n=1}^{m} [d - f(x(n))] \phi(x(n)). \tag{7}$$

Here, $\Delta f_n = \eta[d - \langle f, \phi(x(n)) \rangle] \phi(x(n))$ are the instantaneous updates of the function f from input data based on the kernel expansion (5).

The key observation to extend the supervised learning approach to the TD method is that the difference between desired and predicted output at time n can be written as

$$d - y(n) = \sum_{k=n}^{m} (y(k+1) - y(k)), \tag{8}$$

where $y(m+1) \triangleq d$. Using this expansion in terms of the differences between sequential predictions, we can update the system at each time step. By replacing the error $d - f(x(n))$ in (7) using the relation with temporal differences (8) and rearranging the equation as in [6], we obtain the following update:

$$f \longleftarrow f + \eta \sum_{n=1}^{m} [f(x(n+1)) - f(x(n))] \sum_{k=1}^{n} \phi(x(k)). \tag{9}$$

In this case, all predictions are used equally. Using exponential weighting on recency yields the following update rule:

$$f \longleftarrow f + \eta \sum_{n=1}^{m} [f(x(n+1)) - f(x(n))] \sum_{k=1}^{n} \lambda^{n-k} \phi(x(k)). \tag{10}$$

Here, λ represent an eligibility trace rate that is added to the averaging process over temporal differences to emphasize on the most recently observed states and to efficiently deal with delayed rewards.

The above update rule (10) is called kernel temporal difference (KTD)(λ) [17]. The difference between predictions of sequential inputs is called temporal difference (TD) error,

$$e_{\mathrm{TD}}(n) = f(x(n+1)) - f(x(n)). \tag{11}$$

Note that the temporal differences $f(x(n+1)) - f(x(n))$ can be rewritten using the kernel expansions as $\langle f, \phi(x(n+1)) \rangle - \langle f, \phi(x(n)) \rangle$. This yields the instantaneous update of the function f as $\Delta \overline{f_n} = \eta \langle f, \phi(x(n+1)) - \phi(x(n)) \rangle \sum_{k=1}^{n} \lambda^{n-k} \phi(x(k))$. Using the RKHS properties, the evaluation of the function f at a given x can be calculated from the kernel expansion.

In reinforcement learning, the prediction $y(n) = f(x(n))$ can be considered as the value function (1) or (2). This is how the KTD algorithm provides a nonlinear function approximation to Bellman's equation. When the prediction $y(n)$ represents the state value function, the TD error (11) is extended to the combination of a reward and sequential value function predictions. For instance, in the case of policy evaluation, the TD error is defined as

$$e_{\mathrm{TD}}(n) = r(n+1) + \gamma V(x(n+1)) - V(x(n)). \tag{12}$$

3.3. Convergence of Kernel Temporal Difference(λ). It has been shown in [6, 27] that for an absorbing Markov chain, TD(λ) converges with probability 1 under certain conditions. Recall that the conventional TD algorithm assumes the function class to be linearly parametrized satisfying $y = w^{\top} x$. KTD(λ) can be viewed as a linear function approximation in the RKHS. Using this relation, convergence of KTD(λ) can be obtained as an extension of the convergence guarantees already established for TD(λ).

When $\lambda = 1$, by definition, the KTD($\lambda = 1$) procedure is equivalent to the supervised learning method (7). KTD(1) yields the same per-sequence weight changes as the least square solution since (9) is derived directly from supervised learning by replacing the error term in (8). Thus, the convergence of KTD(1) can be established based on the convergence of its equivalent supervised learning formulation, which was proven in [25].

Proposition 1. *The KLMS algorithm converges asymptotically in the mean sense to the optimal solution under the "small-step-size" condition.*

Theorem 2. *When the stepsize η_n satisfies $\eta_n \geq 0$, $\sum_{n=1}^{\infty} \eta_n = \infty$ and $\sum_{n=1}^{\infty} \eta_n^2 < \infty$, KTD(1) converges asymptotically in the mean sense to the least square solution.*

Proof. Since by (8) the sequence of TD errors can be replaced by a multistep prediction with error $e(n) = d - y(n)$, the result of Proposition 1 also applies to this case. □

In the case of $\lambda < 1$, as shown by [27], the convergence of linear TD(λ) can be proved based on the ordinary differential equation (ODE) method introduced in [28]. This result can be easily extended to KTD(λ) as follows. Let us consider the Markov estimation problem as in [6]. An absorbing Markov chain can be described by the terminal and nonterminal sets of states $\mathcal{T}$ and $\mathcal{N}$, transition probabilities p_{ij} between nonterminal states, the transition probabilities s_{ij} from nonterminal states to terminal states, the vectors x_i representing the nonterminal states, the expected terminal returns $\overline{d}_j$ from the jth terminal state, and the probabilities μ_i of starting at state i. Given an initial state $i \in \mathcal{N}$, an absorbing Markov chain generates an observation sequence of m vectors $x_{i_1}, x_{i_2}, \ldots, x_{i_m}$, where the last element x_{i_m} of the sequence corresponds to a terminal state $i_m \in \mathcal{T}$. The expected outcome $\overline{d}$ given a sequence starting at $i \in \mathcal{N}$ is given by

$$\overline{e}_i^* \equiv E[d \mid i] \tag{13}$$

$$= \sum_{j \in \mathcal{T}} s_{ij} \overline{d}_j + \sum_{j \in \mathcal{N}} p_{ij} \sum_{k \in \mathcal{T}} p_{jk} \overline{d}_k + \cdots \tag{14}$$

$$= \left[\sum_{k=0}^{\infty} Q^k h\right]_i = \left[(I-Q)^{-1} h\right]_i, \qquad (15)$$

where $[x]_i$ denotes the ith element of the array x, Q is the transition matrix with entries $[Q]_{ij} = p_{ij}$ for $i, j \in \mathcal{N}$, and $[h]_i = \sum_{j\in\mathcal{T}} s_{ij}\overline{d}_j$ for $i \in \mathcal{N}$. In linear TD(λ), a sequence of vectors $w_1, w_2, \ldots$, is generated. Each one of these vectors w_n is generated after having a complete observation sequence; that is, a sequence staring at state $i \in \mathcal{N}$ and ending at state $j \in \mathcal{T}$ with the respective return d_j. Similar to linear TD(λ), in KTD(λ) we have a sequence of functions $f_1, f_2, \ldots$, (vectors in a RKHS) for which we can also write a linear update of the mean estimates of terminal return after n sequences have been observed. If f_n is the actual function estimate after sequence n, and $\overline{f}_{n+1}$ is the expected function estimate after the next sequence, we have that

$$\overline{f}_{n+1}(X) = f_n(X) + \eta_{n+1}\mathbf{H}\left(f_n(X) - \overline{e}^*\right), \qquad (16)$$

where $\mathbf{H} = -\mathbf{K}D[I - (1-\lambda)Q(I-\lambda Q)^{-1}]$, $[\mathbf{K}]_{ij} = \kappa(x_i, x_j)$ with $i, j \in \mathcal{N}$, D is a diagonal matrix and $[D]_{ii}$ the expected number of times the state i is visited during a sequence, and $f_n(X)$ is a column vector of function evaluations of the state representations such that $[f_n(X)]_i = f_n(x_i) = \langle f_n, \phi(x_i)\rangle$. Analogously to [27], the mean estimates in (16) converge appropriately if $\mathbf{H}$ has a full set of eigenvalues with negative real parts, for which we need $\mathbf{K}$ to be full rank. For the above to be true, it is required the set of vectors $\{\phi(x_i)\}_{i\in\mathcal{N}}$ to be linearly independent in the RKHS. This is exactly the case when the kernel κ is strictly positive definite as shown in the following proposition.

Proposition 3. *If $\kappa : \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ is a strictly positive definite kernel, for any finite set $\{x_i\}_{i=1}^N \subseteq \mathcal{X}$ of distinct elements, the set $\{\phi(x_i)\}$ is linearly independent.*

Proof. If κ is strictly positive definite, then $\sum \alpha_i\alpha_j\kappa(x_i, x_j) > 0$ for any set x_i where $x_i \neq x_j$, for all $i \neq j$, and any $\alpha_i \in \mathbb{R}$ such that not all $\alpha_i = 0$. Suppose there exists a set $\{x_i\}$ for which $\{\phi(x_i)\}$ are not linearly independent. Then, there must be a set of coefficients $\alpha_i \in \mathbb{R}$ not all equal to zero such that $\sum \alpha_i\phi(x_i) = 0$, which implies that $\|\sum \alpha_i\phi(x_i)\|^2 = 0$

$$0 = \sum \alpha_i\alpha_j \left\langle \phi(x_i), \phi(x_j)\right\rangle = \sum \alpha_i\alpha_j\kappa(x_i, x_j), \qquad (17)$$

which contradicts the assumption. □

The following Theorem is the resulting extension of *Theorem T* in [27] to KTD(λ).

Theorem 4. *For any absorbing Markov chain, for any distribution of starting probailities μ_i such that there are not inaccessible states, for any outcome distributions with finite expected values $\overline{d}_j$, for any strictly positive definite kernel κ, and any set of observation vectors $\{x_i, i \in \mathcal{N}\}$ such that $x_i = x_j$ if and only if $i = j$, there exists an $\epsilon > 0$ such that, if $\eta_n = \eta$ where $0 < \eta < \epsilon$ and for any initial function estimate, the predictions of KTD(λ) converge in expected value to the ideal predictions of (15). If f_n denotes the function estimate after experiencing n sequences, then*

$$\lim_{n\to\infty} E\left[f_n(x_i)\right] = E[d \mid i] = \left[(I-Q)^{-1} h\right]_i, \quad \forall i \in \mathcal{N}. \qquad (18)$$

4. Q-Learning via Kernel Temporal Differences(λ)

Since the value function represents the expected cumulative rewards given a policy, the policy π is better than the policy π' when the policy π gives greater expected return than the policy π'. In other words, $\pi \geq \pi'$ if and only if $Q^{\pi}(x, a) \geq Q^{\pi'}(x, a)$ for all $x \in \mathcal{X}$ and $a \in \mathcal{A}$. Therefore, the optimal action value function Q can be written as $Q^*(x(n), a(n)) = \max_{\pi} Q^{\pi}(x(n), a(n))$. The estimation can be done online. To maximize the expected reward $E[r(n+1) \mid x(n), a(n), x(n+1)]$, one-step Q-learning update was introduced in [29],

$$\begin{aligned} Q(x(n), a(n)) \longleftarrow{}& Q(x(n), a(n)) \\ &+ \eta\Big[r(n+1) + \gamma\max_a Q(x(n+1), a) \\ &- Q(x(n), a(n))\Big]. \end{aligned} \qquad (19)$$

At time n, an action $a(n)$ can be selected using methods such as ϵ-greedy or the Boltzmann distribution, which are popular for exploration and exploitation trade-off [30].

When we consider the prediction y as action value function Q^{π} with respect to a policy π, KTD(λ) can approximate the value function $\widetilde{Q}^{\pi}$ using a family of functions of the form

$$\widetilde{Q}(x(n), a = i) = f(x \mid a = i) = \left\langle f, \phi(x(n))\right\rangle. \qquad (20)$$

Here, $\widetilde{Q}(x(n), a = i)$ denotes a state-action value given a state $x(n)$ at time n and a discrete action i. Therefore, the update rule for Q-learning via kernel temporal difference (Q-KTD)(λ) can be written as

$$\begin{aligned} f \longleftarrow f + \eta\sum_{n=1}^{m}\Big[&r(n+1) + \gamma\max_a Q(x(n+1), a) \\ &- Q(x(n), a(n))\Big]\sum_{k=1}^{n}\lambda^{n-k}\phi(x(k)). \end{aligned} \qquad (21)$$

We can see that the temporal difference (TD) error at time n includes reward and action value function terms. For single-step prediction problems ($m = 1$), (10) yields single updates for Q-KTD(λ) of the form:

$$Q_i(x(n)) = \eta\sum_{j=1}^{n-1} e_{\mathrm{TD}i}(j)\, I_k(j)\, \kappa\left\langle x(n), x(j)\right\rangle. \qquad (22)$$

Here, $Q_i(x(n)) = Q(x(n), a = i)$ and $e_{\mathrm{TD}i}(n)$ denotes the TD error defined as $e_{\mathrm{TD}i}(n) = r_i + \gamma Q_{ii}(x(n+1)) - Q_i(x(n))$, and $I_k(n)$ is an indicator vector of size determined by the number

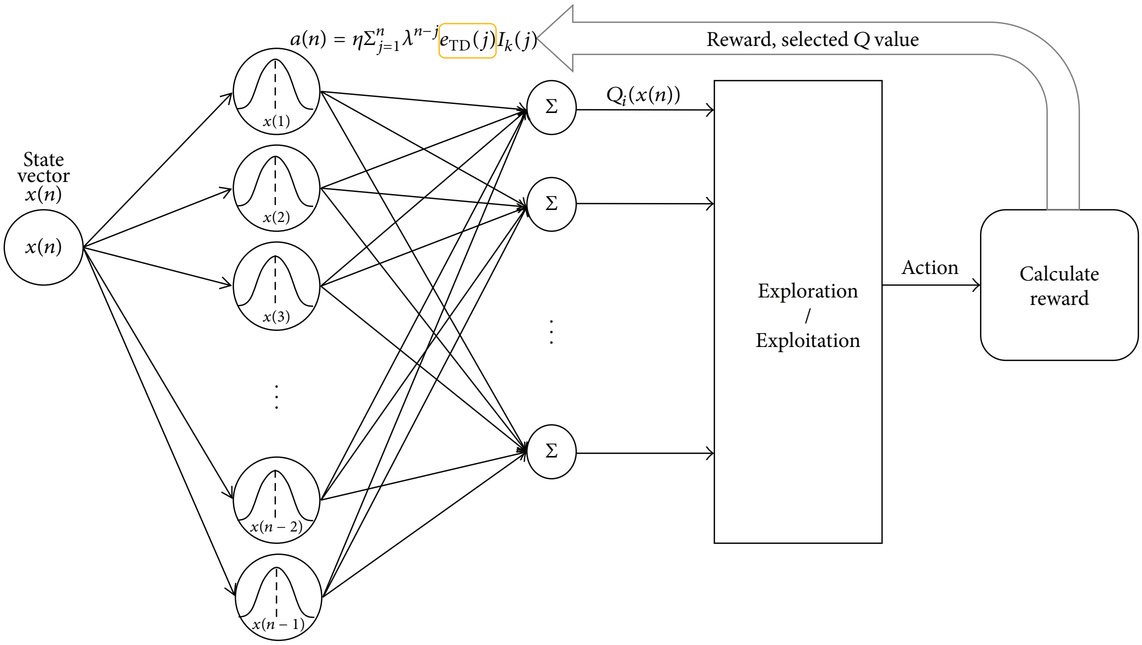

$a(n) = \eta\Sigma_{j=1}^{n}\lambda^{n-j}e_{\mathrm{TD}}(j)I_k(j)$
Reward, selected Q value
State vector $x(n)$
$x(n)$
$x(1)$
$x(2)$
$x(3)$
$x(n-2)$
$x(n-1)$
Σ
Σ
Σ
$Q_i(x(n))$
Exploration / Exploitation
Action
Calculate reward

Figure 2: The structure of Q-learning via kernel temporal difference(λ).

of outputs (actions). Only the kth entry of the vector is set to 1 and the other entries are set to 0. The selection of the action unit k at time n can be based on a greedy method. Therefore, only the weight (parameter vector) corresponding to the winning action gets updated. Recall that the reward r_i corresponds to the action selected by the current policy with input $x(n)$ because it is assumed that this action causes the next input state $x(n+1)$.

The structure of Q-learning via KTD(0) is shown in Figure 2. The number of units (kernel evaluations) increases as more input data arrives. Each added unit is centered at the previous input locations $x(1), x(2), \ldots, x(n-1)$.

In the reinforcement learning brain machine interface (RLBMI) paradigm, kernel temporal difference(λ) helps model the agent (see Figure 1). The action value function Q can be approximated using KTD(λ), for which the kernel based representations enhance the functional mapping capabilities of the system. Based on the estimated Q values, a policy decides a proper action. Note that the policy corresponds to the learning policy which changes over time in Q-learning.

5. Online Sparsification

One characteristic of nonparametric approaches is their inherently growing structure which is usually linear in the number of input data points. This rate of growth becomes prohibitive for practical applications that handle increasing amounts of incoming data over time. Various methods have been proposed to alleviate this problem (see [31] and references therein). These methods, known as kernel sparsification methods, can be applied to the KTD algorithm to control the growth of the terms in the function expansion, also known as filter size. Popular examples of kernel sparsification methods are the approximate linear dependence (ALD) [19], Surprise criterion [32], Quantization approach [21], and the kernel distance based method [23]. The main idea of sparsification is to only consider a reduced set of samples, called the dictionary, to represent the function of interest. The computational complexity of ALD is $O(d^2)$, where d is the size of the dictionary. For the other methods mentioned above, the complexity is $O(d)$.

Each of these methods has its own criterion to determine whether an incoming sample should be added to the current dictionary. The Surprise criterion [32] measures the subjective information of exemplar $\{x, d\}$ with respect to a learning system Γ:

$$S_{\Gamma}(x, d) = -\ln p(x, d \mid \Gamma). \tag{23}$$

Only samples with high values of Surprise are considered as candidates for the dictionary. In the case of the Quantization approach introduced in [21], the distance between a new input $x(n)$ and the existing dictionary elements $C(n-1)$ is evaluated. The new input sample is added to the dictionary if the distance between the new input $x(n)$ and the closest element in $C(n-1)$,

$$\min_{x_i \in C(n-1)} \left\| x(n) - x_i \right\| > \epsilon_U, \tag{24}$$

is larger than the Quantization size ϵ_U. Otherwise, the new input state $x(n)$ is absorbed by the closest existing unit. Very similar to the quantization approach, the method presented in [23] applies a distance threshold criterion in the RKHS. The kernel distance based criterion given a state dictionary

$D(n-1)$ adds a new unit when the new input state $x(n)$ satisfies following condition;

$$\min_{x_i \in D(n-1)} \left\| \phi(x(n)) - \phi(x_i) \right\|^2 > \mu_1. \tag{25}$$

For some kernels such as Gaussian, the Quantization method and the kernel distance based criterion can be shown to be equivalent.

6. Simulations

Note that the KTD algorithm has been introduced for value function estimation. To evaluate the algorithm's nonlinear capability, we first examine the performance of the KTD(λ) in the problem of state value function estimation $\widetilde{V}$ given a fixed policy π. We carry out experiments on a simple illustrative Markov chain initially described in [33]. This is a popular experiment involving an episodic task to test TD learning algorithms. The experiment is useful in illustrating linear as well as nonlinear functions of the state representations and shows how the state value function is estimated using the adaptive system.

6.1. Linear Case. Even though we emphasize the capability of KTD(λ) as a nonlinear function approximator, under the appropriate kernel size, KTD(λ) should approximate linear functions on a region of interest as well. To test its efficacy, we observe the performance on a simple Markov chain (Figure 3). There are 13 states numbered from 12 to 0. Each trial starts at state 12 and terminates at state 0. Each state is represented by a 4-dimensional vector, and the rewards are assigned in such a way that the value function V is a linear function on the states; namely, V^* takes the values $[0, -2, -4, \ldots, -22, -24]$ at states $[0, 1, 2, \ldots, 11, 12]$. In the case of $V = w^\top x$, the optimal weights are $w^* = [-24, -16, -8, 0]$.

To assess the performance, the updated estimate of the state value function $\widetilde{V}(x)$ is compared to the optimal value function V^* at the end of each trial. This is done by computing the RMS error of the value function over all states

$$\text{RMS} = \sqrt{\frac{1}{n} \sum_{x \in \mathcal{X}} \left(V^*(x) - \widetilde{V}(x) \right)^2}, \tag{26}$$

where n is the number of states, $n = 13$.

Stepsize scheduling is applied as follows:

$$\eta(n) = \eta_0 \frac{a_0 + 1}{a_0 + n}, \quad \text{where } n = 1, 2, \ldots, \tag{27}$$

where η_0 is the initial stepsize, and a_0 is the annealing factor which controls how fast the stepsize decreases. In this experiment, $a_0 = 100$ is applied. Furthermore, we assume that the policy π is guaranteed to terminate, which means that the value function V^π is well-behaved without using a discount factor γ in (3), that is, $\gamma = 1$.

In KTD(λ), we employ the Gaussian kernel:

$$\kappa(x(i), x(j)) = \exp\left(\frac{-\left\| x(i) - x(j) \right\|^2}{2h^2} \right), \tag{28}$$

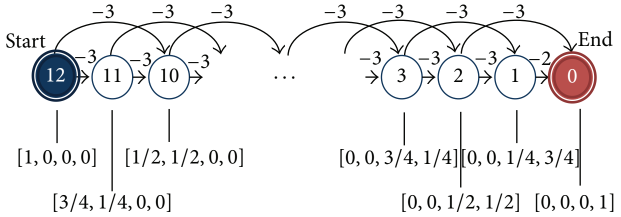


FIGURE 3: A 13-state Markov chain [33]. For states from 2 to 12, the state transition probabilities are 0.5 and the corresponding rewards are −3. State 1 has state transition probability of 1 to the terminal state 0 and a reward of −2. States 12, 8, 4, and 0 have the 4-dimensional state space representations $[1, 0, 0, 0]$, $[0, 1, 0, 0]$, $[0, 0, 1, 0]$, and $[0, 0, 0, 1]$, respectively. The representations of the other states are linear interpolations between the above vectors.

which is a universal kernel commonly encountered in practice. To find the optimal kernel size, we fix all the other free parameters around median values, $\lambda = 0.4$ and $\eta_0 = 0.5$, and the average RMS error over 10 Monte Carlo runs is compared. For this specific experiment, smaller kernel sizes yield better performance since the state representations are finite. However, in general, applying too small kernel sizes leads to over-fitting and slow learning. In particular, choosing a very small kernel makes the algorithm behave very similar to the table look up method. Thus, we choose the kernel size $h = 0.2$ to be the largest kernel size for which we obtain similar mean RMS values as for smaller kernel sizes.

After fixing the kernel size to $h = 0.2$, the experimental evaluation of different combinations of eligibility trace rates λ and initial step sizes η_0 are observed. Figure 4 shows the average performance over 10 Monte Carlo runs for 1000 trials.

All λ values with optimal stepsize show good approximation to V^* after 1000 trials. Notice that KTD($\lambda = 0$) shows slightly better performance than KTD($\lambda = 1$). This may be attributed to the local nature of KTD when using the Gaussian kernel. In addition, varying the stepsize has a relatively small effect on KTD(λ). The Gaussian kernel as well as other shift-invariant kernels provide an implicit normalized update rule which is known to be less sensitive to stepsize. Based on Figure 4, the optimal eligibility trace rate and initial stepsize value $\lambda = 0.6$ and $\eta_0 = 0.3$ are selected for KTD with kernel size $h = 0.2$.

The learning curve of KTD(λ) is compared to the conventional TD algorithm, TD(λ). The optimal parameters employed in both algorithms are based on the experimental evaluation. In TD(λ), $\lambda = 1$ and $\eta_0 = 0.1$ are applied. The RMS error is averaged over 50 Monte Carlo runs for 1000 trials. Comparative learning curves are given in Figure 5.

In this experiment, we confirm the ability of KTD(λ) to handle the function approximation problem when the fixed policy yields a state value function that is linear in the state representation. Both algorithms reach the mean RMS value of around 0.06. As we expected, TD(λ) converges faster to the optimal solution because of the linear nature of the problem. KTD(λ) converges slower than TD(λ), but it is also able to approximate the value function properly. In this sense,

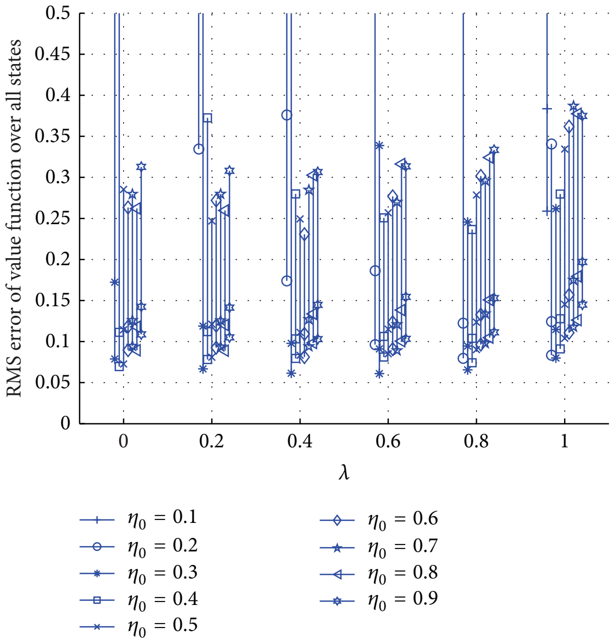


Figure 4: Performance comparison over different combinations of eligibility trace rates λ and initial step sizes η_0 in KTD(λ) with $h = 0.2$. The vertical line segments contain the mean RMS value after 100 trials (top marker), 500 trials (middle marker), and 1000 trials (bottom marker).

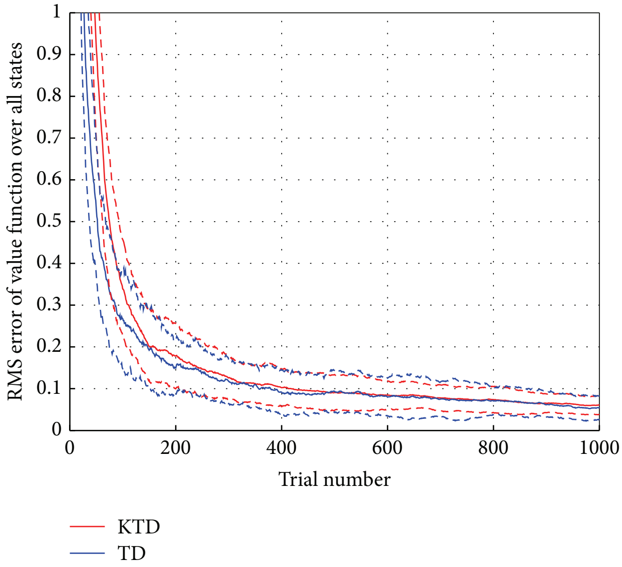


Figure 5: Learning curve of KTD(λ) and TD(λ). The solid line shows the mean RMS error, and the dashed line shows the +/− standard deviations over 50 Monte Carlo runs.

the KTD algorithm is open to wider class of problems than its linear counterpart.

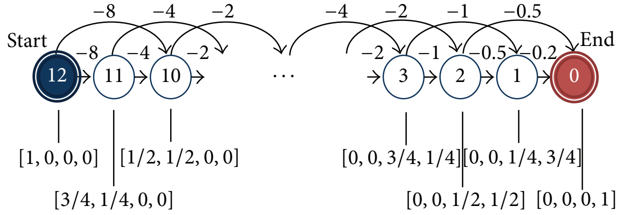


Figure 6: A 13-state Markov chain. In states from 2 to 12, each state transition has probability 0.5, and state 1 has transition probability 1 to the absorbing state 0. Note that optimal state value functions can be represented as a nonlinear function of the states, and corresponding reward values are assigned to each state.

6.2. Nonlinear Case. Previous section show the performances of KTD(λ) on the problem of estimating a state value function, which is a linear function of the given state representation. The same problem can be turned into a nonlinear one by modifying the reward values in the chain such that the resulting state value function V^* is no longer a linear function of the states.

The number of states and the state representations remain the same as in the previous section. However, the optimal value function V^* becomes nonlinear with respect to the representation of the states; namely, $V^* = [0\ \ -0.2\ \ -0.6\ \ -1.4\ \ -3\ \ -6.2\ \ -12.6\ \ -13.4\ \ -13.5\ \ -14.45\ \ -15.975\ \ -19.2125\ \ -25.5938]$ for states 0 to 12. This implies that the reward values for each state are different from the ones given for the linear case (Figure 6).

Again, to evaluate the performance, after each trial is completed, the estimated state value $\widetilde{V}$ is compared to the optimal state value V^* using RMS error (26). For KTD(λ), the Gaussian kernel (28) is applied and kernel size $h = 0.2$ is chosen. Figure 7 shows the average RMS error over 10 Monte Carlo runs for 1000 trials.

The combination of $\lambda = 0.4$ and $\eta_0 = 0.3$ shows the best performance, but the $\lambda = 0$ case also shows good performances. Unlike TD(λ) [6], there is no dominant value for λ in KTD(λ). Recall that it has been proved that convergence is guaranteed for linearly independent representations of the states, which is automatically fulfilled in KTD(λ) when the kernel is strictly positive definite. Therefore, the differences are rather due to the convergence speed controlled by the interaction between the step size and the elegibilty trace.

The average RMS error over 50 Monte Carlo runs is compared with Gaussian process temporal difference (GPTD) [15] and TD(λ) in Figure 8. The purpose of GPTD implementation is to have comparison among kernelized value function approximations. Here, the applied optimal parameters for KTD(λ) are $\lambda = 0.4$, $\eta_0 = 0.3$, and $h = 0.2$, for GPTD, $\lambda = 1$, $\sigma^2 = 0.5$, and $h = 0.2$, and for TD(λ), $\lambda = 0.8$ and $\eta_0 = 0.1$.

The linear function approximation, TD(λ) (blue line), cannot estimate the optimal state values. KTD(λ) outperforms the linear algorithm as expected since the Gaussian kernel is strictly positive definite. GPTD also learns the target state values, but the system fails to reach as low error values as KTD. GPTD is sensitive to the selection of the covariance value in the noise, σ^2; if the value is small, the system becomes unstable, and larger values cause the the learning to slow down. GPTD models the residuals, the difference between

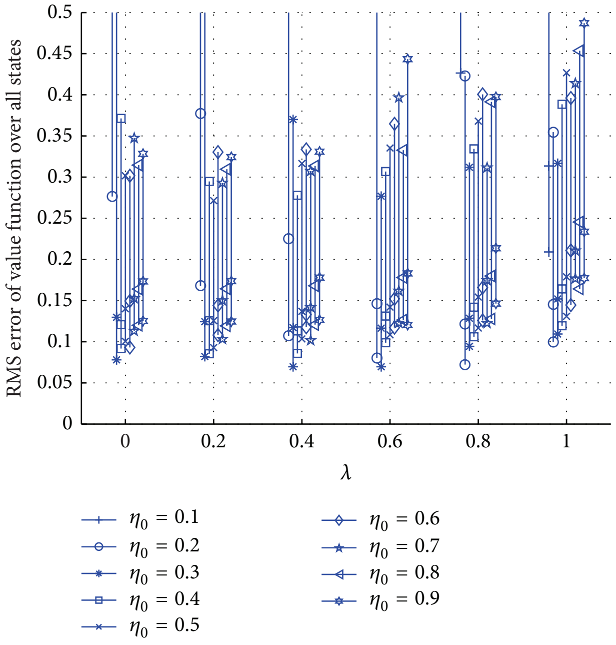


FIGURE 7: Performance comparison over different combinations of λ and the initial stepsize η_0 in KTD(λ) with $h = 0.2$. The plotted segment is the mean RMS value after 100 trials (top segment), 500 trials (middle segment), and 1000 trials (bottom segment).

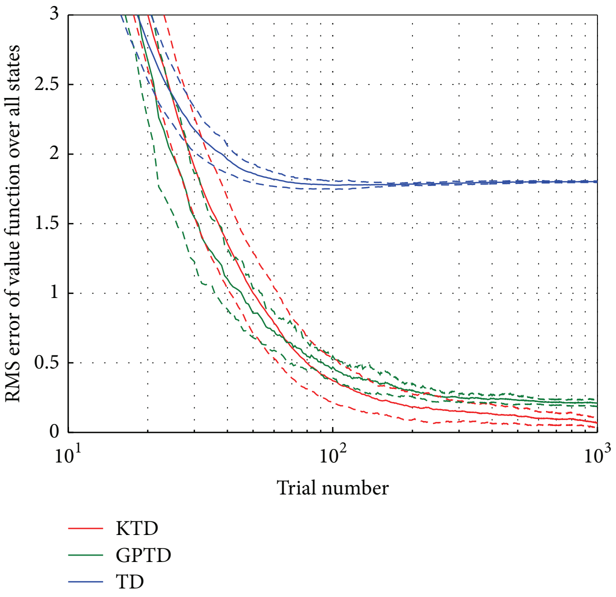


FIGURE 8: Learning curves of KTD(λ), GPTD, and TD(λ). The solid lines show the mean RMS error, and the dashed lines represent the (+/−) standard deviation over 50 Monte Carlo runs.

expected return and actual return, as a Gaussian process. This assumption does not hold true for the Markov chain in Figure 6. As we can observe in Figure 8, KTD(λ) reaches to the mean value around 0.07, and the mean value of GPTD and TD(λ) are around 0.2 and 1.8, respectively.

In the synthetic examples, we presented experimental results to approximate the state value function under a fixed policy. We observed that KTD(λ) performs well on both linear and nonlinear function approximation problems. In addition, in the previous section, we showed how the linear independence of the input state representations can affect the performance of algorithms. The use of strictly positive definite kernels in KTD(λ) implies the linear independence condition, and thus this algorithm converges for all $\lambda \in [0, 1]$. In the following section, we will apply the extended KTD algorithm to estimate the action value function which can be employed in finding a proper control policy for RLBMI tasks.

7. Experimental Results on Neural Decoding

In our RLBMI experiments, we map the monkey's neural signal to action-directions (computer cursor/robot arm position). The agent starts at a naive state, but the subject has been trained to receive rewards from the environment. Once it reaches the assigned target, the system and the subject earn a reward, and the agent updates its neural state decoder. Through iteration, the agent learns how to correctly translate neural states into action-directions.

7.1. Open-Loop RLBMI.

7.1. Open-Loop RLBMI. In open-loop RLBMI experiments, the output of the agent does not directly change the state of the environment because this is done with prerecorded data. The external device is updated based only on the actual monkey's physical response. In this sense, we only consider the monkey's neural state from successful trials to train the agent. The goal of these experiments is to evaluate the system's capability to predict the proper state to action mapping based on the monkey's neural states and to assess the viability of further closed-loop experiments.

7.1.1. Environment. The data employed in these experiments is provided by SUNY Downstate Medical Center. A female bonnet macaque is trained for a center-out reaching task allowing 8 action-directions. After the subject attains about 80% success rate, microelectrode arrays are implanted in the motor cortex (M1). Animal surgery is performed under the Institutional Animal Care and Use Committee (IACUC) regulations and assisted by the Division of Laboratory Animal Resources (DLAT) at SUNY Downstate Medical Center.

From 96-channel recordings, a set of 185 units are obtained after sorting. The neural states are represented by the firing rates of each unit on 100 ms window. There is a set of 8 possible targets and action directions. Every trial starts at the center point, and the distance from the center to each target is 4 cm; anything within a radius of 1 cm from the target point is considered as a valid reach.

7.1.2. Agent. In the agent, Q-learning via kernel temporal difference (Q-KTD)(λ) is applied to neural decoding. For Q-KTD(λ), we employ the Gaussian kernel (28). After the neural states are preprocessed by normalizing their dynamic range to lie between −1 and 1, they are input to the system. Based on the preprocessed neural states, the system predicts which

Table 1: Average success rates of Q-KTD in open-loop RLBMI.

Epochs	1	2	3	4	5	6	7
2 target	0.44	0.96	0.99	0.99	0.97	0.99	0.99
4 target	0.41	0.73	0.76	0.95	0.99	0.99	0.99
8 target	0.32	0.65	0.79	0.89	0.96	0.98	0.98

direction the computer cursor will move. Each output unit represents one of the 8 possible directions, and among the 8 outputs one action is selected by the ϵ-greedy method [34]. The action corresponding to the unit with the highest Q value gets selected with probability $1 - \epsilon$. Otherwise, any other action is selected at random. The performance is evaluated by checking whether the updated position reaches the assigned target, and depending on the updated position, a reward value is assigned to the system.

7.1.3. Results on Single Step Tasks. Here, the targets should be reached within a single step; rewards from the environment are received after a single step, and one action is performed by the agent per trial. The assignment of reward is based on the 1-0 distance to the target, that is, $\text{dist}(x, d) = 0$ if $x = d$, and $\text{dist}(x, d) = 1$, otherwise. Once the cursor reaches the assigned target, the agent gets a positive reward +0.6, else it receives negative reward −0.6 [35]. Exploration rate $\epsilon = 0.01$ and discount factor $\gamma = 0.9$ are applied. Also, we consider $\lambda = 0$ since our experiment performs single step updates per trial. In this experiment, the firing rates of the 185 units on 100 ms windows are time-embedded using 6th order tap delay. This creates a representation space where each state is a vector with 1295 dimensions.

We start with the simplest version of the problem by considering only 2-targets (right and left). The total number of trials is 43 for the 2-targets. For Q-KTD, the kernel size h is heuristically chosen based on the distribution of the mean squared distances between pairs of input states; let $s = E[\|x_i - x_j\|^2]$, then $h = \sqrt{s/2}$. For this particular data set, the above heuristic gives a kernel size $h = 7$. The stepsize $\eta = 0.3$ is selected based on the stability bound that was derived for the kernel least mean square algorithm [25],

$$\eta < \frac{N}{\operatorname{tr}\left[G_\phi\right]} = \frac{N}{\sum_{j=1}^{N} \kappa\left(x\left(j\right), x\left(j\right)\right)} = 1, \qquad (29)$$

where G_ϕ is the gram matrix. After 43 trials, we count the number of trials which received a positive reward, and the success rate is averaged over 50 Monte Carlo runs. The performance of the Q-KTD algorithm is compared with Q-learning via time delayed neural net (Q-TDNN) and the online selective kernel-based temporal difference learning algorithm (Q-OSKTD) [23] in Figure 9. Note that TDNN is a conventional approach to function approximation and has already been applied to RLBMI experiments for neural decoding [1, 2]. OSKTD is a kernel-based temporal difference algorithm emphasizing on the online sparsifications.

Both Q-KTD and Q-OSKTD reach around 100% success rate after 2 epochs. In contrast, the average success rate of Q-TDNN slowly increases yet never reaches the same performance as Q-KTD. In the case of Q-OSKTD, the value function updates require one more parameter μ_2 to decide the subspace. To validate the algorithm's capability to estimate proper policy, we set the sparsified dictionary as the same size as the number of sample observations. In Q-OSKTD, we observed that the subspace selection parameter plays an important role in terms of the speed of learning. It turns out that for the above experiment, smaller subspaces allow faster learning. In the extreme case of Q-OSKTD, where only the current state is affected, the updates become equivalent to the update rule of Q-KTD.

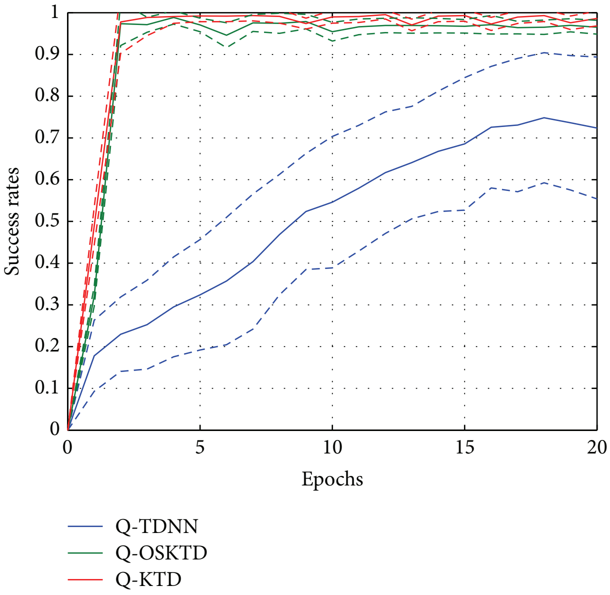


Figure 9: The comparison of average learning curves from 50 Monte Carlo runs among Q-TDNN, Q-OSKTD, and Q-KTD. Solid lines show the mean success rates and the dashed lines show the confidence interval based on one standard deviation.

Since all the experimental parameters are fixed over 50 Monte Carlo runs, the confidence interval for Q-KTD can be simply associated with the random effects introduced by the ϵ-greedy method employed for action selection with exploration, thus, the narrow interval. However, with Q-TDNN a larger variation of performance is observed, which shows how the initialization, due to local minima, influences the success of learning; it is observed that Q-TDNN is able to approximate the Q-KTD performance, but most of the times, the system falls on local minima. This highlights one of the advantages of KTD compared to TDNN, which is the insensitivity to initialization.

Table 1 shows average success rates over 50 Monte Carlo runs with respect to different number of targets. The first

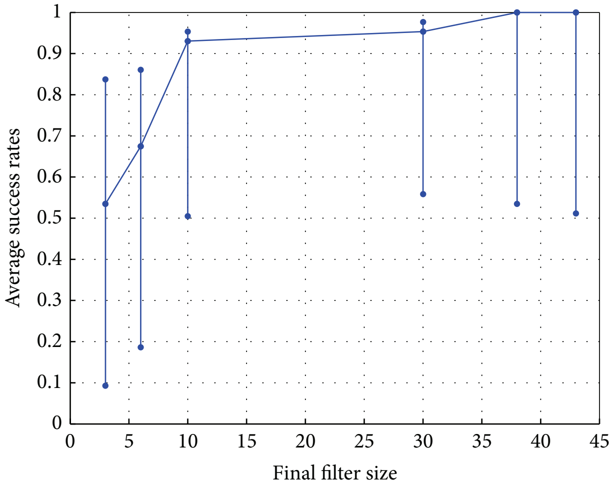


Figure 10: Average success rates over 50 Monte Carlo runs with respect to different filter sizes. The vertical line segments are the mean success rates after 1 epoch (bottom markers), 2 epochs (middle markers), and 20 epochs (top markers).

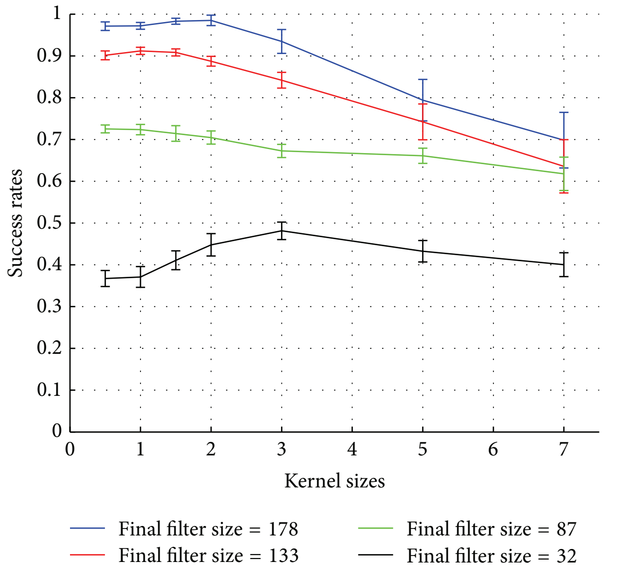


Figure 11: The effect of filter size control on 8-target single-step center-out reaching task. The average success rates are computed over 50 Monte Carlo runs after the 10th epoch.

row corresponds to the mean success rates displayed on Figure 9 (red solid line). This is included in the Table 1 to ease comparison with 4 and 8-target experiments. The 4 target task involves reaching right, up, left and down positions from the center. Note that in all tasks, 8 directions are allowed at each step. The standard deviation of each epoch is around 0.02.

One characteristic of nonparametric approaches is the growing filter structure. Here, we observe how filter size influences the overall performance in Q-KTD by applying Surprise criterion [32] and Quantization [21] methods. In the case of the 2-target center-out reaching task, we should expect the filter size to become as large as 861 units after 20 epochs without any control of the filter size. Using the Surprise criterion, the filter size can be reduced to 87 centers with acceptable performance. However, Quantization allows the filter size to be reduced to 10 units while maintaining performance above 90% for success rates. Figure 10 shows the effect of filter size in the 2-target experiment using the Quantization approach. For filter sizes as small as 10 units, the average success rates remain stable. With 10 units, the algorithm shows similar learning speed to the linearly growing filter size, with success rates above 90%. Note that quantization limits the capacity of the kernel filter since less units than samples are employed and thus it helps to avoid over-fitting.

In the 2-target center-out reaching task, quantized Q-KTD shows satisfactory results in terms of initialization and computational cost. Further analysis of Q-KTD is conducted on a larger number of targets. We increase the number of targets from 2 to 8. All experimental parameters are kept the same as for the 2-target experiment. The only change is step-size $\eta = 0.5$. The 178 trials are applied for the 8-target reaching task.

To gain more insight about the algorithm, we observe the interplay between Quantization size ϵ_U and kernel size h. Based on the distribution of squared distances between pairs of input states, various kernel sizes ($h = 0.5, 1, 1.5, 2, 3, 5, 7$) and Quantization sizes ($\epsilon_U = 1, 110, 120, 130$) are considered. The corresponding success rates for final filter sizes of 178, 133, 87, and 32 are displayed in Figure 11.

With a final filter size of 178 (blue line), the success rates are superior to any other filter sizes for every kernel sizes tested, since it contains all input information. Especially for small kernel sizes ($h \leq 2$), success rates above 96% are observed. Moreover, note that even after reduction of the state information (red line), the system still produces acceptable success rates for kernel sizes ranging from 0.5 to 2 (around 90% success rates).

Among the best performing kernel sizes, we favor the largest one since it provides better generalization guarantees. In this sense, a kernel size $h = 2$ can be selected since this is the largest kernel size that considerably reduces the filter size and yields a neural state to action mapping that performs well (around 90% of success rates). In the case of kernel size $h = 2$ with final filter size of 178, the system reaches 100% success rates after 6 epochs with a maximum variance of 4%. As we can see from the number of units, higher representation capacity is required to obtain the desired performance as the task becomes more complex. Nevertheless, results on the 8-target center-out reaching task show that the method can effectively learn the brain state-action mapping for this task with a reasonable complexity.

7.1.4. Results on Multistep Tasks. Here, we develop a more realistic scenario; we extend the task to multistep and multitarget experiments. This case allows us to explore the role of the eligibility traces in Q-KTD(λ). The price paid for this extension is that now, the eligibility trace rate λ selection needs to be carried out according to the best observed performance. Testing based on the same experimental set

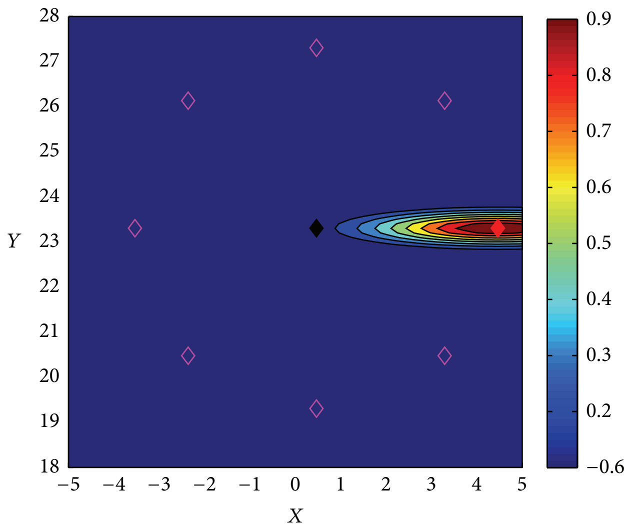


Figure 12: Reward distribution for right target. The black diamond is the initial position, and the purple diamond shows the possible directions including the assigned target direction (red diamond).

up employed for the single step task, that is, a discrete reward value is assigned at the target, causes extremely slow learning since not enough guidance is given. The system requires long periods of exploration until it actually reaches the target. Therefore, we employ a continuous reward distribution around the selected target defined by the following expression:

$$r(s) = \begin{cases} p_{\text{reward}} G(s) & \text{if } G(s) > 0.1, \\ n_{\text{reward}} & \text{if } G(s) \leq 0.1, \end{cases} \quad (30)$$

where $G(s) = \exp[(s-\boldsymbol{\mu})^{\top} \mathbf{C}_{\theta}^{-1}(s-\boldsymbol{\mu})]$, $s \in \mathbb{R}^2$ is the position of the cursor, $p_{\text{reward}} = 1$, and $n_{\text{reward}} = -0.6$. The mean vector $\boldsymbol{\mu}$ corresponds to the selected target location and the covariance matrix,

$$\mathbf{C}_{\theta} = \mathbf{R}_{\theta} \begin{pmatrix} 7.5 & 0 \\ 0 & 0.1 \end{pmatrix} \mathbf{R}_{\theta}^{\top}, \qquad \mathbf{R}_{\theta} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}, \quad (31)$$

which depends on the angle θ of the selected target as follows: for target index one and five, the angle is 0, two and six are for $-\pi/4$, three and seven are for $\pi/2$, and four and eight are for $\pi/4$. (Here, the target indexes follow the location depicted on Figure 6 in [22].) Figure 12 shows the reward distribution for target index one. The same form of distribution is applied to the other directions centred at the assigned target point.

Once the system reaches the assigned target, the system earns a maximum reward of +1 and receives partial rewards according to (30) during the approaching stage. When the system earns the maximum reward, the trial is classified as a successful trial. The maximum number of steps per trial is limited such that the cursor must approach the target in a straight line trajectory. Here, we also control the complexity of the task by allowing different number of targets and steps. Namely, 2-step 4-target (right, up, left, and down) and 4-step 3-target (right, up, and down) experiments are performed. Increasing the number of steps per trial amounts to making smaller jumps according to each action. After each epoch, the number of successful trials is counted for each target direction. Figure 13 shows the learning curves for each target and the average success rates.

Larger number of steps results in lower success rates. However, the two cases (two and four steps) obtain an average success rate above 60% for 1 epoch. The performances show all directions can achieve success rates above 70% after convergence, which encourage the application of the algorithm to online scenarios.

7.2. Closed-Loop RLBMI Experiments. In closed loop RLBMI experiments, the behavioral task is a reaching task using a robotic arm. The decoder controls the robot arm's action direction by predicting the monkey's intent based on its neuronal activity. If the robot arm reaches the assigned target, a reward is given to both the monkey (food reward) and the decoder (positive value). Notice that the two intelligent systems learn coadaptively to accomplish the goal. These experiments are conducted in cooperation with the Neuroprosthetics Research Group at the University of Miami. The performance is evaluated in terms of task completion accuracy and speed. Furthermore, we provide a methodology to tease apart the influence of each one of the systems of the RLBMI in the overall performance.

7.2.1. Environment. During pretraining, a marmoset monkey was trained to perform a target reaching task, namely, moving a robot arm to two spatial locations denoted as A trial and B trial. The monkey was taught to associate changes in motor activity during A trials and produce static motor responses during B trials. Once a target is assigned, a beep signals the start of the trial. To control the robot during the user training phase, the monkey is required to steadily place its hand on a touch pad for 700~1200 ms. This action produces a *go* beep that is followed by the activation of one of the two target LEDs (A trial: red light for left direction or B trial: green light for right direction). The robot arm goes to a *home position*, namely, the center position between the two targets. Its gripper shows an object (food reward such as waxworm or marshmallow for A trial and undesirable object (wooden bead) for B trial). To move the robot to the A location, the monkey needed to reach out and touch a sensor within 2000 ms, and to make the robot reach to the B target, the monkey needed to keep its arm motionless on the touch pad for 2500 ms. When the monkey successfully moved the robot to the correct target, the target LEDs would blink and the monkey would receive a food reward (for both the A and B targets).

After the monkey is trained to perform the assigned task properly, a microelectrode array (16-channel tungsten microelectrode arrays, Tucker Davis Technologies, FL) is surgically implanted under isoflurane anesthesia and sterile conditions. Neural states from the motor cortex (M1) are recorded. These neural states become the inputs to the neural decoder. All surgical and animal care procedures were

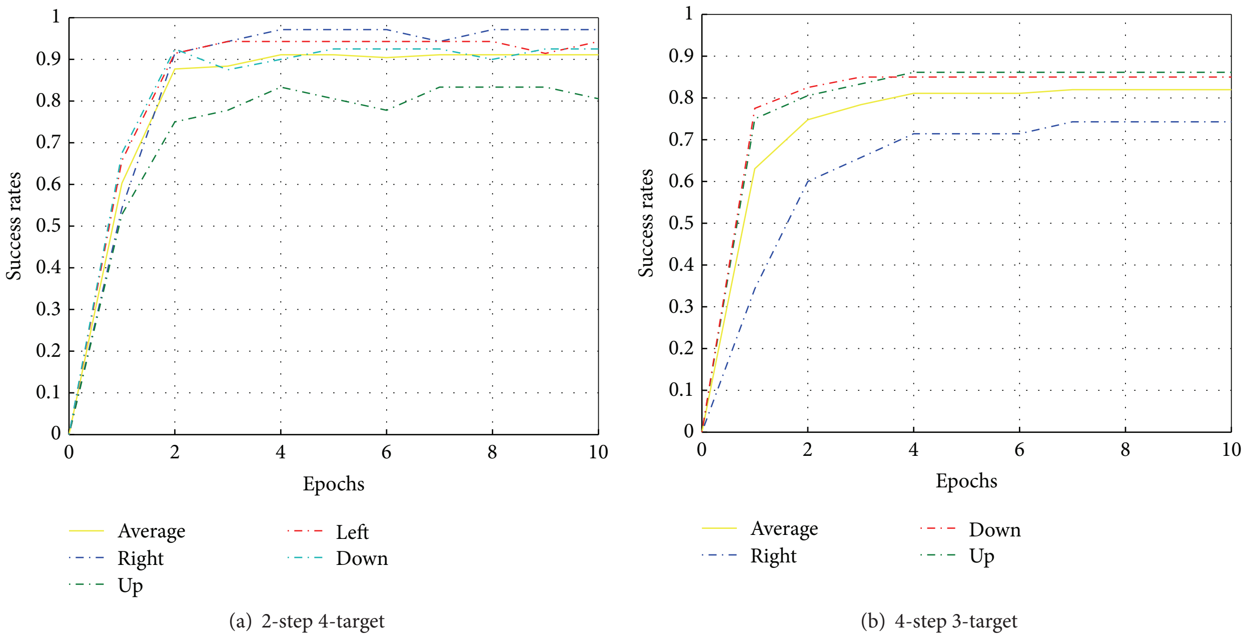


Figure 13: The learning curves for multistep multitarget tasks.

consistent with the National Research Council Guide for the Care and Use of Laboratory Animals and were approved by the University of Miami Institutional Animal Care and Use Committee.

In the closed-loop experiments, after the initial holding time that produces the *go* beep, the robotic arm's position is updated based solely on the monkey's neural states. Differently from the user pretraining sessions, the monkey is not required to perform any movement. During the real-time experiment, 14 neurons are obtained from 10 electrodes. The neural states are represented by the firing rates on a 2 sec window following the *go* signal.

7.2.2. Agent. For the BMI decoder, we use Q-learning via kernel Temporal Differences (Q-KTD)(λ). One big difference between open-loop and closed-loop applications is the amount of accessible data; in the closed-loop experiments, we can only get information about the neural states that have been observed up to the present. However, in the previous offline experiments, normalization and kernel selection were conducted offline based on the entire data set. It is not possible to apply the same method to the online setting since we only have information about the input states up to the present time. Normalization is a scaling procedure that interplays with the choice of the kernel size. Proper selection of the kernel size brings proper scaling to the data. Thus, in contrast to the previous open-loop experiments, normalization of the input neural states is not applied, and the kernel size is automatically selected given the inputs.

The Gaussian kernel (28) is employed, and the kernel size h is automatically selected based on the history of inputs. Note that in the closed-loop experiments, the dynamic range of states varies from experiment to experiment. Consequently, the kernel size needs to be re-adjusted each time a new experiment takes place, and it cannot be determined beforehand. At each time, the distances between the current state and the previously observed states are computed to obtain the output values, $\widetilde{Q}$ in this case. Therefore, we use the distance values to select the kernel size as follows:

$$h_{\text{temp}}(n) = \sqrt{\frac{1}{2(n-1)}\sum_{i=1}^{n-1}\|x(i)-x(n)\|^2},$$
$$h(n) = \frac{1}{n}\left[\sum_{i=1}^{n-1}h(i) + h_{\text{temp}}(n)\right]. \tag{32}$$

Using the squared distance between pairs of previously seen input states, we can obtain an estimate of the mean distance. This value is also averaged along with past kernel sizes to obtain the current kernel size.

Moreover, we consider $\gamma = 1$ and $\lambda = 0$ since our experiments perform single step trials. Stepsize $\eta = 0.5$ is applied. The output represents the 2 possible directions (left and right), and the robot arm moves based on the estimated output from the decoder.

7.2.3. Results. The overall performance is evaluated by checking whether the robot arm reaches the assigned target. Once the robot arm reaches the target, the decoder gets a positive reward +1, otherwise, it receives negative reward −1.

Table 2 shows the decoder performance over 4 days in terms of success rates. Each day corresponds to a separate experiment. In Day 1, the experiment has a total of 20 trials (10 A trials and 10 B trials). The overall success rate was 90%. Only the first trial for each target was incorrectly assigned.

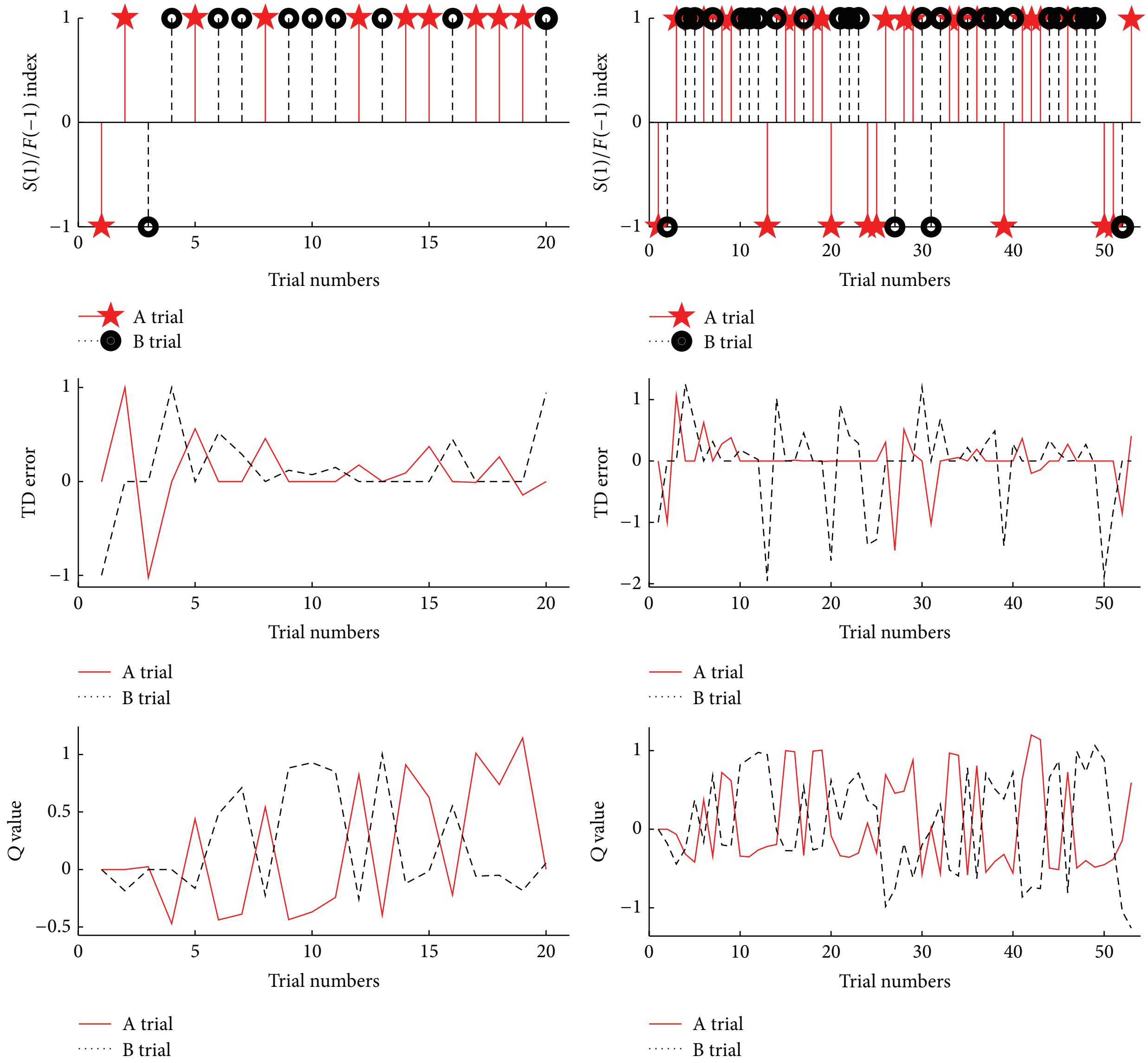


FIGURE 14: Performance of Q-learning via KTD in the closed loop RLBMI controlled by a monkey for Day 1 (left) and Day 3 (right); the success (+1) index and failure (−1) index of each trial (top), the change of TD error (middle), and the change of Q-values (down).

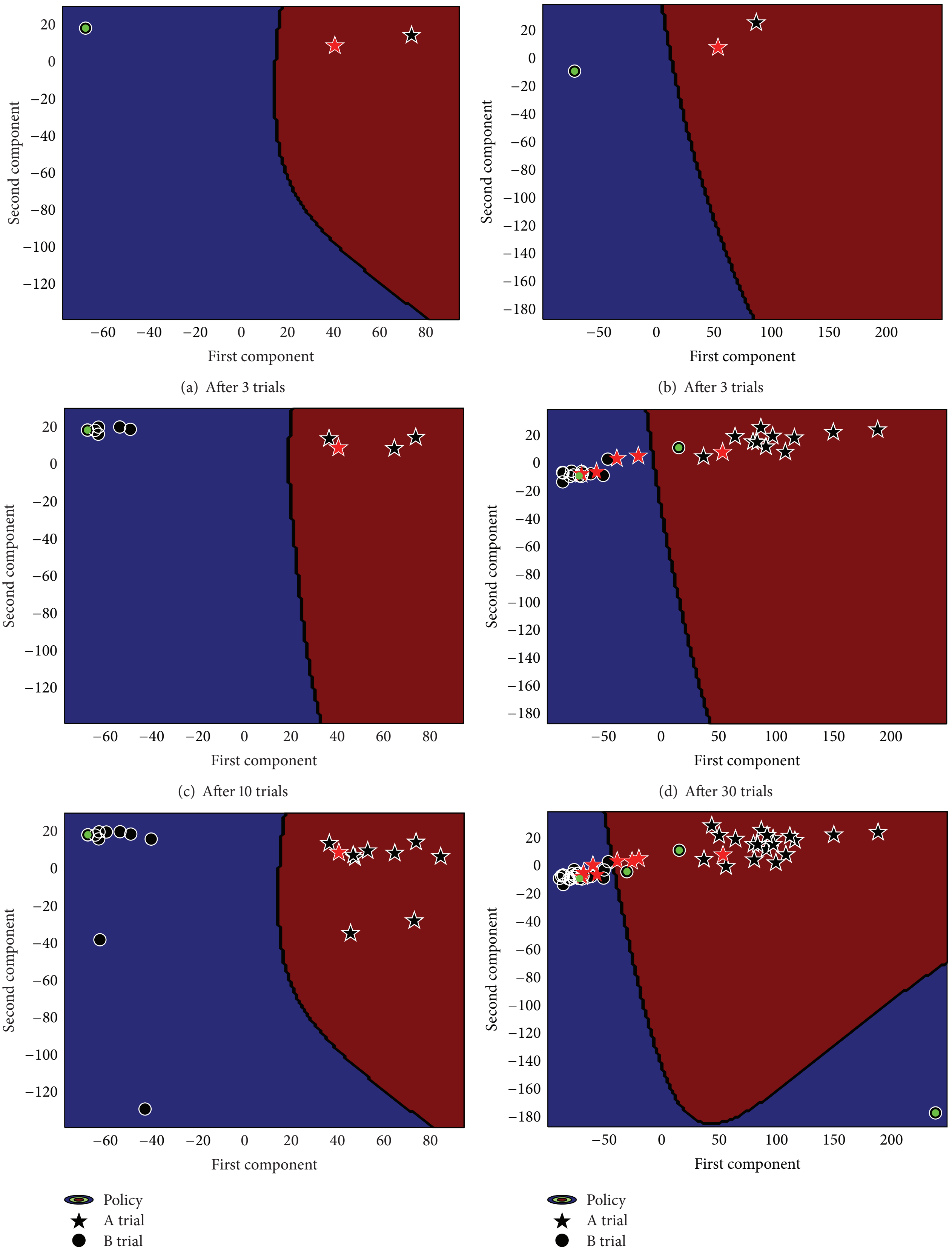
TABLE 2: Success rates of Q-KTD in closed-loop RLBMI.

	Total trial numbers (total A, B trial)	Success rates (%)
Day 1	20 (10, 10)	90.00
Day 2	32 (26, 26)	84.38
Day 3	53 (37, 36)	77.36
Day 4	52 (37, 35)	78.85

Note that at each day, the same experimental set up was utilized. The decoder was initialized in the same way at each day. We did not use pretrained parameters to initialize the system. To understand the variation of the success rates across days, we look at the performance of Day 1 and Day 3. Figure 14 shows the decoder performance for the 2 experiments.

Although the success rate for Day 3 is not as high as Day 1, both experiments show that the algorithm learns an appropriate neural state to action map. Even though there is variation among the neural states within each day, the decoder adapts well to minimize the TD error, and the Q-values converge to the desired values for each action. Because this is a single step task and the reward +1 is assigned for a successful trial, it is desired for the estimated action value $\widetilde{Q}$ to be close to +1.

It is observed that the TD error and Q-values oscillate. The drastic change of TD error or Q-value corresponds to the missed trials. The overall performance can be evaluated by checking whether the robot arm reaches the desired target



(a) After 3 trials

(b) After 3 trials

(c) After 10 trials

(d) After 30 trials

(e) After 20 trials

(f) After 57 trials

Figure 15: Estimated policy for the projected neural states from Day 1 (left) and Day 3 (right). The failed trials during the closed loop experiment are marked as red stars (missed A trials) and green dots (missed B trials).

(the top plots in Figure 14). However, this assessment does not show what causes the change in the system values. In addition, it is hard to know how the two separate intelligent systems interact during learning and how neural states affect the overall performance.

Under the coadaptation scenario in the RLBMI architecture, it is obvious that if one system does not perform properly, it will cause detrimental effects on the performance of the other system. If the BMI decoder does not give proper updates to the robotic device, it will confuse the user conducting the task, and if the user gives improper state information or the translation is wrong, the resulting update may fail even though the BMI decoder was able to find the optimal mapping function.

Using the proposed methodology introduced in [36], we can observe how the decoder effectively learns a good state to action mapping, and how neural states affect the prediction performance. Figure 15 shows how each participant (the agent and the user) influences the overall performance in both successful and missed trials, and how the agent adapts the environment. By applying principal component analysis (PCA), the high-dimensional neural states can be visualized in two dimensions using the first two largest principal components. In this two-dimensional space of projected neural states, we can visualize the estimated policy, as well.

We observe the behavior of two systems at the beginning, intermediate, and final stages of the experiment by using the neural states that have been observed as well as the learned decoder up to the given stage. It is evident that the decoder can predict nonlinear policies. Day 1 (left column in Figure 15) shows that the neural states from the two classes are well separable. It was noted during Day 3 that the monkey seemed less engaged in the task than in Day 1. This suggests the possibility that during some trials the monkey was distracted and may not have been producing a consistent set of neural outputs. We are also able to see this phenomenon from the plots (right column in Figure 15). We can see that most of the neural states that were misclassified appear to be closer to the states corresponding to the opposite target in the projected state space. However, the estimated policy shows that the system effectively learns. Note that the initially misclassified A trials (red stars in Figure 15(d) which are located near the estimated policy boundary) are assigned to the right direction when learning has been accomplished (Figure 15(f)). It is a remarkable fact that the system adapts to the environment online.

8. Conclusions

The advantages of KTD(λ) in neural decoding problems were observed. The key observations of this kernel-based learning algorithm are its capabilities for nonlinear function approximation and its convergence guarantees. We also examined the capability of the extended KTD algorithm (Q-KTD(λ)) in both open-loop and closed-loop reinforcement learning brain machine interface (RLBMI) experiments to perform reaching tasks.

In open-loop experiments, results showed that Q-KTD(λ) can effectively learn the brain state-action mapping and offer performance advantages over conventional nonlinear function approximation methods such as time-delay neural nets. We observed that Q-KTD(λ) overcomes main issues of conventional nonlinear function approximation methods such as local minima and proper initialization.

Results on closed-loop RLBMI experiments showed that the algorithm succeeds in finding a proper mapping between neural states and desired actions. Its advantages are that it does not depend on the initialization neither require any prior information about input states. Also, parameters can be chosen on the fly based on the observed input states. Moreover, we observed how the two intelligent systems coadaptively learn in an online reaching task. The results showed that KTD is powerful for practical applications due to its nonlinear approximation capabilities in online learning.

The observation and analysis of KTD(λ) give us a basic idea of how this algorithm behaves. However, in the case of Q-KTD(λ), the convergence analysis remains challenging since Q-learning contains both a learning policy and a greedy policy. For Q-KTD(λ), the convergence proof for Q-learning using temporal difference (TD)(λ) with linear function approximation in [37] can provide a basic intuition for the role of function approximation on the convergence of Q-learning.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is partially supported by DARPA Contract N66001-10-C-2008. The authors would like to thank Pratik Chhatbar and Brandi Marsh for collecting the center-out reaching task data for the open loop experiments.

References

[1] J. DiGiovanna, B. Mahmoudi, J. Fortes, J. C. Principe, and J. C. Sanchez, "Coadaptive brain-machine interface via reinforcement learning," *IEEE Transactions on Biomedical Engineering*, vol. 56, no. 1, pp. 54–64, 2009.

[2] B. Mahmoudi, *Integrating robotic action with biologic perception: a brain machine symbiosis theory [Ph.D. dissertation]*, University of Florida, Gainesville, Fla, USA, 2010.

[3] E. A. Pohlmeyer, B. Mahmoudi, S. Geng, N. W. Prins, and J. C. Sanchez, "Using reinforcement learning to provide stable brain-machine interface control despite neural input reorganization," *PLoS ONE*, vol. 9, no. 1, Article ID e87253, 2014.

[4] S. Matsuzaki, Y. Shiina, and Y. Wada, "Adaptive classification for brainmachine interface with reinforcement learning," in *Proceedings of the 18th International Conference on Neural Information Processing*, vol. 7062, pp. 360–369, Shanghai, China, November 2011.

[5] M. J. Bryan, S. A. Martin, W. Cheung, and R. P. N. Rao, "Probabilistic co-adaptive brain-computer interfacing," *Journal of Neural Engineering*, vol. 10, no. 6, Article ID 066008, 2013.

[6] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.

[7] J. A. Boyan, *Learning evaluation functions for global optimization [Ph.D. dissertation]*, Carnegie Mellon University, 1998.

[8] S. J. Bradtke and A. G. Barto, "Linear least-squares algorithms for temporal difference learning," *Machine Learning*, vol. 22, pp. 33–57, 1996.

[9] A. Geramifard, M. Bowling, M. Zinkevich, and R. S. Sutton, "ilstd: eligibility traces and convergence analysis," in *Advances in Neural Information Processing Systems*, pp. 441–448, 2007.

[10] R. S. Sutton, C. Szepesvári, and H. R. Maei, "A convergent O(n) algorithm for off-policy temporal-difference learning with linear function approximation," in *Proceedings of the 22nd Annual Conference on Neural Information Processing Systems (NIPS '08)*, pp. 1609–1616, MIT Press, December 2008.

[11] R. S. Sutton, H. R. Maei, D. Precup et al., "Fast gradient-descent methods for temporal-difference learning with linear function approximation," in *Proceeding of the 26th International Conference On Machine Learning (ICML '09)*, pp. 993–1000, June 2009.

[12] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.

[13] S. Haykin, *Neural Networks and Learning Machines*, Prentice Hall, 2009.

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

[15] Y. Engel, *Algorithms and representations for reinforcement learning [Ph.D. dissertation]*, Hebrew University, 2005.

[16] X. Xu, T. Xie, D. Hu, and X. Lu, "Kernel least-squares temporal difference learning," *International Journal of Information Technology*, vol. 11, no. 9, pp. 54–63, 2005.

[17] J. Bae, P. Chhatbar, J. T. Francis, J. C. Sanchez, and J. C. Principe, "Reinforcement learning via kernel temporal difference," in *Proceedings of the 33rd Annual International Conference of the IEEE on Engineering in Medicine and Biology Society (EMBC '11)*, pp. 5662–5665, 2011.

[18] S. Zhao, *From fixed to adaptive budget robust kernel adaptive filtering [Ph.D. dissertation]*, University of Florida, Gainesville, Fla, USA, 2012.

[19] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2275–2285, 2004.

[20] X. Xu, "A sparse kernel-based least-squares temporal difference algorithms for reinforcement learning," in *Proceedings of the 2nd International Conference on Natural Computation*, vol. 4221, pp. 47–56, 2006.

[21] B. Chen, S. Zhao, P. Zhu, and J. C. Principe, "Quantized kernel least mean square algorithm," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 1, pp. 22–32, 2012.

[22] J. Bae, L. S. Giraldo, P. Chhatbar, J. T. Francis, J. C. Sanchez, and J. C. Principe, "Stochastic kernel temporal difference for reinforcement learning," in *Proceedings of the 21st IEEE International Workshop on Machine Learning for Signal Processing (MLSP '11)*, pp. 1–6, IEEE, September 2011.

[23] X. Chen, Y. Gao, and R. Wang, "Online selective kernel-based temporal difference learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 12, pp. 1944–1956, 2013.

[24] R. S. Rao and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, New York, NY, USA, 1998.

[25] W. Liu, J. C. Principe, and S. Haykin, *Kernel Adaptive Filtering: A Comprehensive Introduction*, Wiley, 2010.

[26] J. Mercer, "Functions of positive and negative type, and their connection with the theory of integral equations," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 209, pp. 415–446, 1909.

[27] P. Dayan and T. J. Sejnowski, "TD(λ) converges with probability 1," *Machine Learning*, vol. 14, no. 3, pp. 295–301, 1994.

[28] H. J. Kushner and D. S. Clark, *Stochastic Approximation Methods for Constrained and Unconstrained Systems*, vol. 26 of *Applied Mathematical Sciences*, Springer, New York, NY, USA, 1978.

[29] C. J. C. H. Watkins, *Learning from delayed rewards [Ph.D. dissertation]*, King's College, London, UK, 1989.

[30] C. Szepesvari, *Algorithms for Reinforcement Learning*, edited by R. J. Branchman and T. Dietterich, Morgan & Slaypool, 2010.

[31] S. Zhao, B. Chen, P. Zhu, and J. C. Príncipe, "Fixed budget quantized kernel least-mean-square algorithm," *Signal Processing*, vol. 93, no. 9, pp. 2759–2770, 2013.

[32] W. Liu, I. Park, and J. C. Príncipe, "An information theoretic approach of designing sparse kernel adaptive filters," *IEEE Transactions on Neural Networks*, vol. 20, no. 12, pp. 1950–1961, 2009.

[33] J. A. Boyan, "Technical update: least-squares temporal difference learning," *Machine Learning*, vol. 49, pp. 233–246, 2002.

[34] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[35] J. C. Sanchez, A. Tarigoppula, J. S. Choi et al., "Control of a center-out reaching task using a reinforcement learning Brain-Machine Interface," in *Proceedings of the 5th International IEEE/EMBS Conference on Neural Engineering (NER '11)*, pp. 525–528, May 2011.

[36] J. Bae, L. G. Sanchez Giraldo, E. A. Pohlmeyer, J. C. Sanchez, and J. C. Principe, "A new method of concurrently visualizing states, values, and actions in reinforcement based brain machine interfaces," in *Proceedings of the 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC '13)*, pp. 5402–5405, July 2013.

[37] F. S. Melo, S. P. Meyn, and M. I. Ribeiro, "An analysis of reinforcement learning with function approximation," in *Proceedings of the 25th International Conference on Machine Learning*, pp. 664–671, July 2008.