Towards autonomous neuroprosthetic control using Hebbian reinforcement learning

This content has been downloaded from IOPscience. Please scroll down to see the full text.

View the table of contents for this issue, or go to the journal homepage for more

Download details:

IP Address: 129.171.150.150
This content was downloaded on 14/10/2013 at 15:54

Please note that terms and conditions apply.
Towards autonomous neuroprosthetic control using Hebbian reinforcement learning

Babak Mahmoudi\(^1,5\), Eric A Pohlmeyer\(^2\), Noeline W Prins\(^2\), Shijia Geng\(^2\) and Justin C Sanchez\(^2,3,4\)

\(^1\) Department of Neurosurgery, Emory University, Atlanta, GA, USA
\(^2\) Department of Biomedical Engineering, University of Miami, FL, USA
\(^3\) Department of Neuroscience, University of Miami, FL, USA
\(^4\) Miami Project to Cure Paralysis, University of Miami, FL, USA

E-mail: b.mahmoudi@emory.edu

Received 20 July 2013
Accepted for publication 16 September 2013
Published 8 October 2013
Online at stacks.iop.org/JNE/10/066005

Abstract

Objective. Our goal was to design an adaptive neuroprosthetic controller that could learn the mapping from neural states to prosthetic actions and automatically adjust adaptation using only a binary evaluative feedback as a measure of desirability/undesirability of performance.

Approach. Hebbian reinforcement learning (HRL) in a connectionist network was used for the design of the adaptive controller. The method combines the efficiency of supervised learning with the generality of reinforcement learning. The convergence properties of this approach were studied using both closed-loop control simulations and open-loop simulations that used primate neural data from robot-assisted reaching tasks.

Main results. The HRL controller was able to perform classification and regression tasks using its episodic and sequential learning modes, respectively. In our experiments, the HRL controller quickly achieved convergence to an effective control policy, followed by robust performance. The controller also automatically stopped adapting the parameters after converging to a satisfactory control policy. Additionally, when the input neural vector was reorganized, the controller resumed adaptation to maintain performance.

Significance. By estimating an evaluative feedback directly from the user, the HRL control algorithm may provide an efficient method for autonomous adaptation of neuroprosthetic systems. This method may enable the user to teach the controller the desired behavior using only a simple feedback signal.

(Some figures may appear in colour only in the online journal)

1. Introduction

Recent clinical trials have shown restoration of motor function in people living with paralysis using neuroprosthetics \([1, 2]\). At the heart of these neuroprosthetic systems, the controller is a key component that translates neural representation of user’s motor intent into the prosthetic actions or functional electrical stimulation of the paralyzed limb \([3–5]\). Neuroprosthetic control is a difficult challenge because the neural input to the controller is by nature non-stationary and the statistics of the data may change over time due to various reasons including neuroplasticity \([6]\), learning new tasks \([7]\) and the dynamics of the tissue–electrode interface \([8, 9]\). Previous studies have shown that the adaptation of the controller parameters will increase the performance of the neuroprosthetic systems \([10–17]\). However, in order to use neuroprosthetic systems as a practical tool in daily-life activities the controller should ideally be able to adapt and learn the optimal control policy autonomously. Here autonomy means adaptation without need

5 Author to whom any correspondence should be addressed.
for a copy of the desired response or manual calibrations by a caregiver.

From the machine learning perspective, the choice of learning paradigm (supervised, unsupervised and reinforcement learning) has important implications on the design of autonomous neuroprosthetic controllers. Supervised learning methods have been used widely to design adaptive controllers for neuroprosthetic systems. In spite of high efficiency and speed of adaptation, supervised learning techniques are not suitable for autonomous adaptation because they rely on the existence of a desired response (for example kinematics of arm movements in a reaching task). Unfortunately, in patients with motor disability, this information may not be available. There have been attempts to ‘infer’ [16, 17] a desired response; however, these approaches may not exactly match the users intent and may not be feasible in unstructured environments such as those encountered during daily living. Unsupervised learning is an alternative that only relies on the structure of the input neural data. These techniques can be used for neural decoding without any need for a reference signal [18]. However, for effective adaptation of the decoder a model of the user’s motor behavior is required. The decoder adaptation in this paradigm is then based on tracking the changes in the tuning parameters, which may not be feasible in the event of major input perturbations. Also, adaptation in unsupervised paradigm is often slow and less accurate [18]. Reinforcement learning (RL) is an interactive learning framework [19] that does not need a detailed desired signal (unlike supervised learning) nor a model of user’s motor behavior (such as in unsupervised learning) for adaptation. The RL controller actively modifies its behavior to optimize a scalar measure of performance (reward) through interaction with the environment [20].

By recording a scalar evaluative feedback from the user, RL techniques can be used for autonomous adaptation of the neuroprosthetic controllers. In our previous work [21], we developed a framework for autonomous adaptation of the neuroprosthetic controllers using the actor–critic method of RL [22–25]. In the actor–critic framework, the neuroprosthetic controller (actor) learned the neural state-to-action mapping based on the user’s evaluative feedback [26–28]. The important requirement for autonomous RL adaptation is that the evaluative feedback should be recorded from the user. The feedback could be recorded directly from the brain, based on the neural representation of reward expectation, or supplied by the user through other means, e.g. eye blink or muscle twitch. The role of the critic is to translate the user’s feedback into an explicit training signal that can be used by the actor for adaptation. Therefore, learning techniques that enable the actor to adapt efficiently using a parsimonious feedback are pivotal for development of autonomous neuroprosthetic systems. In this paper, we introduce a learning control approach that enables the actor to learn the state-to-action mapping efficiently, based on a binary evaluative feedback which is the simplest form of evaluative feedback that the critic may provide.

Associative RL defines a class of RL problems where the controller receives immediate reinforcement based on the most recent state–action pair [29, 30]. In these tasks, elements of gradient-based search in supervised learning and reinforcement-based optimization can be combined to design efficient controllers that can learn to adapt just using a scalar evaluative feedback. The feedback indicates whether the action of the controller was desirable or not. The controller always selects actions to maximize the gradient of the reward. We present the theory and testing of a neuroprosthetic controller that is able to learn and automatically produce a stable neural to motor mapping and respond to perturbations by readjusting its parameters in order to maintain the performance, all using a binary evaluative feedback. We investigated the performance of the controller in terms of speed of convergence, generalization, accuracy and recovery from perturbation using simulations and real neural data.

2. Methods

2.1. Hebbian reinforcement learning

Hebbian reinforcement learning (HRL) is a class of associative RL problems where the learning agent receives an immediate reinforcement feedback ($r$) after each decision and the control policy (mapping between the states and action) is parameterized by the synaptic weights, $ω$, of a connectionist network [31–33]. The goal of the learning process is to search within the network’s weight space ($W$) for an optimal set of parameters that maximizes the expected value of the reward $E(r)$ [34]. This can be accomplished by using policy gradient methods to estimate and optimize $\partial E(r|W)/\partial ω$ [35, 36]. REINFORCE is a particular class of policy gradient algorithms in which the learning agent finds the optimal solution without needing to explicitly compute the gradient [37]. Consider the stochastic node $j$ in a connectionist network with transfer function $f(.)$ that receives input $x_i$ from node $i$ through synaptic weight $ω_{ij}$ and generates output $x_j$. The probability mass function $g$ that node $j$ takes a certain value $ϕ$ can be written as

$$g(ϕ, ω_{ij}, x_i) = Pr(x_j = ϕ|ω_{ij}, x_i).$$

(1)

It can be proven [37] that this network with the following incremental update rule,

$$\Delta ω_{ij} = µ(r - b_{ij})\partial\ln g_j/\partial ω_{ij}$$

(2)

will climb the gradient of the expected reward where $µ$ is the learning rate and $b_{ij}$ is the reward baseline. In fact, $(r - b_{ij})\partial\ln g_j/\partial ω_{ij}$ represents an unbiased estimate of the $\partial E(r|W)/\partial ω$. This implies that the average weight update using equation (2) will converge to a local maximum of $r$. For the special case of $b_{ij} = 0$ and a stochastic binary node with logistic transfer function $f(.)$, the probability that the output state of the node $j$ is $x_j$, will be

$$P_j = f\left(\sum_ω ω_{jk}x_k\right),$$

(3)

and the update equation (2) will become

$$\Delta ω_{ij} = µ(r x_j - P_j)x_i.$$ 

(4)

The weight update algorithm using equation (4), which is known as the reward-inaction algorithm in adaptive control
Figure 1. The actor–critic model for autonomous adaptation of the neuroprosthetic systems. The actor plays the role of neuroprosthetic controller by mapping the neural motor commands into actions during goal-directed interaction of the user with the environment. The actions of the actor will modulate the reward expectation of the user. The critic will translate this reward expectation to an evaluative feedback that the actor will use for modifying its control policy.

Figure 2. The structure of the actor in the actor–critic decoding framework using (A) a feed-forward neural network with (B) binary nodes. The action corresponding to an output node with the maximum value among all the output nodes was selected (marked with W in panel (A)). Depending on the desirability of the action a binary (+1) evaluative feedback is projected by the critic to all of the nodes in the network and modulated the synaptic weight updates based on the local pre- and postsynaptic activity.

2.2. Neuroprosthetic control architecture and learning algorithm

The HRL algorithm was used for training the actor in the actor–critic control architecture (figure 1). The actor served as the neuroprosthetic controller by mapping the user’s motor neural states into neuroprosthetic actions. The role of the critic was to evaluate the actor’s performance and to provide an instantaneous binary evaluative feedback for the adaptation of the actor. The actor in figure 1 was parameterized by a fully connected neural network as shown in figure 2(A). The network consisted of a set of input nodes, a hidden layer and an output layer of nonlinear nodes. The input nodes received neural activity from the primary motor cortex and the hidden layer nodes formed a set of basis functions for dimensionality reduction and projecting the input neural vector to a feature space. Each output node computed the value of one action. Based on a greedy action selection policy [19], the action with the highest value from the output nodes was then selected and implemented.

theory [38], was expanded to the associative reward–penalty algorithm [39] by adding a penalty term to the update rule in equation (4):

\[
\Delta \omega_{ij} = \mu^+ r(x_j - P_j) x_i + \mu^- (1 - r)(1 - x_j - P_j) x_i, \tag{5}
\]

where the \( \mu^+ \) and \( \mu^- \) are separate learning rates for the reward and penalty components, respectively. In connectionist networks, the update rule in equation (5) captures the essence of HRL by correlating the local presynaptic and postsynaptic activity in the network with a global reinforcement signal. In other words, \( r \) evaluates the ‘appropriateness’ of the node’s output, \( x_j \), due to the input \( x_i \) [40].

The mapping between the neural activity and actions was parameterized using the synaptic weights (\( \omega_{ij} \)) of the network. The goal of the actor learning algorithm was to search for an optimal set of weights that maximized the expected value of the positive feedback. The actor parameters were adjusted using
equation (5) where $\mu^+ \gg \mu^- > 0$ were the learning rates and $r$ was the instantaneous evaluative feedback computed by the critic. In equation (5), the first term emphasizes positive feedback and the second term corresponds to negative feedback. The balance between these two terms is unique for the binary evaluative feedback where $r = \pm 1$ because only when $r = -1$ do both terms contribute to the weight update. In this way, the controller becomes more sensitive to the negative feedback and quickly responds to failures. Conversely, in the case of positive feedback, the second term simply becomes zero. In the limit when $P_j$ approaches $x_j$, the total weight update will approach zero. Consequently, once the controller has converged to a stable control policy, the weights stop changing automatically until a negative reinforcement is received. One of the important features of the HRL algorithm is that the controller learns the mapping between the neural states and actions in an on-line fashion (in machine learning terms) using equation (5). This means that as the neural data and the evaluative feedback are received, the controller adapts and adjusts the weights in real-time under closed-loop control conditions.

Learning through interaction with the environment is one of the fundamental features of the HRL controller [42]. We defined and tested two different learning modes for the HRL algorithm (sequential and episodic) that reflected different levels of interaction between the controller and the environment. The sequential mode is applicable to regression tasks in which the controller has to perform a sequence of actions over multiple steps in order to accomplish the goal of a trial. This means that there can be multiple sequences of actions that could eventually result in the end goal, and thus multiple solutions that could be learned by the HRL. After completing each of the individual actions, the controller receives a positive or negative evaluative feedback depending on whether the action increased the probability of achieving the end goal or decreased it. For example, in a reaching task the goal is navigating the prosthetic arm to a target. Movement direction at each time is the action. At a specific time-step if the movement direction is towards the target, that indicates the action increased the probability of success and that action will be reinforced with a positive evaluative feedback. Thus, in the sequential learning mode the controller continuously uses equation (5) to update its parameters. In the early stages of the learning, the actions will be exploratory until the controller learns the best mapping between neural states and actions.

In the episodic interaction mode, the controller can select only one action in each trial, and thus that action must achieve the goal for a successful trial. The episodic learning mode is suitable for classification tasks. Unlike supervised classification though, in which both input patterns and class labels are presented to train the classifier, for the HRL no class labels are available. Instead, upon action selection the controller will receive a $+1$ or $-1$ feedback depending on success or failure. Since achieving the trial goals requires only a single clear action, the HRL can be encouraged to learn more quickly in the episodic task compared to the sequential task. To reduce the number of trials necessary for the HRL controller to learn the episodic task, we were able to use a form of experience replay to speed learning in this paradigm [43, 44]. After each trial, the neural vector, selected action, and the action outcome were registered in a database. Whenever updating the parameters, the actor went through all the previous entries of the database and modified the control policy by re-adjusting the network weights. Table 1 summarizes the experience replay algorithm: $a_t$, $s_t$, $w_t$, and $r_t$ represent the action, neural state vector, weight matrices of the network, and the binary evaluative feedback at time $t$, respectively. $\pi$ is the state-action mapping and $R(\cdot)$ represents a function that was used to re-evaluate the actions during the experience replay phase. This was necessary, because in the episodic interactions the only reinforcement that could be known for certain was for the action chosen in real time, thus alternative reinforcement had to be generated when replay actions were different from the real-time actions.

2.3. Closed-loop simulation methods

In order to test the performance of the HRL controller in response to known neural states, we developed a simulation platform for neuroprosthetic reaching tasks in a 2D grid space to test both the sequential and episodic learning performance of the controller. The goal of the controller was to infer the movement direction from the overall activity of the neural ensemble. The simulator was composed of three main components: synthetic neural data generator, neuroprosthetic controller (actor) and the behavioral paradigm. Figure 3 shows the components of the closed-loop simulator and their interaction.

The user’s neuronal activity was simulated by five synthetic neural ensembles. The synthetic neural data was generated using a biologically realistic model of spiking neurons [45]. Neurons in four of the ensembles were each tuned to one action (i.e. moving left, right, up and down). The neurons in the fifth ensemble were not tuned to any action to simulate the uncorrelated neurons often observed in real experiments [46]. The goal of the user was to reach targets in a 2D grid space. Therefore, at each time-step the user generated a motor command by increasing the firing rate of the corresponding neural ensemble above the base-line activity. A vector of the firing rates of all the neurons over a 100 ms time window was used as the input feature vector to the controller. The four discrete actions (four principal directions of movement) available to the controller spanned the 2D grid space in Cartesian coordinates, thus we call these actions motor primitives in the grid space.

The controller’s task was to map the activity of the neural ensembles to appropriate actions in order to reach the target. We assumed the user was monitoring the performance of the controller and would generate an evaluative feedback that was modeled using a binary signal. If the action (movement) of the controller was desirable the user generated a $+1$ feedback otherwise the feedback was $-1$. In this paradigm the controller again had no specific information about the location of the target. The only information that was available to the controller was the user’s neural motor commands and the evaluative feedback that followed the execution of each action. In all
the experiments the controller was initialized with random weights.

The generalization performance of the HRL controller was tested using a pinball reaching task in which the controller was required to reach targets at random locations in the 2D space [16] by selecting actions (movement direction) sequentially over multiple steps (each step being the implementation of a movement action). Upon reaching to the target, a new target was presented and the previous target was regarded as the starting point for the new trial. A timeout period was defined within which if the task was not successful the trial was considered a failure and the location of the target changed to another random location. A constraint in the new target allocation ensured that the distance between the starting position and the target position was not less than 20 steps.

For the sequential learning mode (figure 3(A)), the user generated new neural commands at each step as it tried to move the prosthetic to the target. The controller’s task was to infer the user’s intended movements based on its interaction with the user and environment. The sign of the cosine of the angle between the desired direction, i.e. the movement that the user’s neural encoding was meant to produce, and the actual movement direction made by the HRL controller was fed back to the controller as the evaluative feedback. It is important to note that the controller did not have any direct information about either the desired direction or location of the target. Rather, the evaluative feedback used here provides a binary signal that simulates the discrepancy between what the user was expecting to happen, and what the actual prosthetic action was. The user continued to generate neural states in each trial until either the prosthetic reached the target (success) or the time limit expired (failure).

The episodic learning mode of the HRL algorithm was tested using the simulation setup in figure 3(B). The goal of this experiment was to test the capability of the HRL controller to learn the neural patterns associated with movements in the four directions of the 2D space, and then use those as motor primitives when generating more complicated movement trajectories during the pinball task. The synthetic neural data was generated the same way as in the sequential learning paradigm. However, in each trial the starting position was the same and the target was only one-step away, therefore the controller had only one opportunity to select the correct action. Depending on success or failure, the controller received a +1 or −1 feedback in each trial. After learning the neural signature of the motor primitives in the episodic mode, the weights were frozen. The generalization performance of the controller was tested in the pinball task in which the controller had to generate multiple step reach trajectories to achieve the target. The trial timing limitations, target movements and distances were the same as in the sequential mode pinball tests. Since the controller had to decode movement direction from the user’s neural pattern each step, these tests required the controller to use the previously learned motor primitives as building blocks to create reach trajectories.

### Table 1. Experience replay algorithm in episodic interaction mode.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Randomly initialize the actor parameters (w_i)</td>
</tr>
<tr>
<td>2</td>
<td>At time step t, execute the action a_t = π(s_t, w_i) and receive a binary evaluative feedback r_t ∈ [−1, 1]</td>
</tr>
<tr>
<td>3</td>
<td>Register the tuple s_t, a_t, r_t in the database</td>
</tr>
<tr>
<td>4</td>
<td>For each step contained in the database, adapt the weight matrices</td>
</tr>
<tr>
<td></td>
<td>Re-evaluate the policy using a′: π(s_t, w_i) and ŷ = R(a′, r_t)</td>
</tr>
<tr>
<td></td>
<td>a′ = a_t + 1, r_t = +1</td>
</tr>
<tr>
<td></td>
<td>a′ ≠ a_t, 0 ≤ r_t ≤ 1</td>
</tr>
<tr>
<td></td>
<td>In essence, when updating R(·), the same reinforcement as registered in the database was used if the same action was taken. Conversely, if a different action was taken from a previously rewarded action, indifferent feedback was offered, and if a different action from a previously punished action was taken, positive reinforcement was offered.</td>
</tr>
<tr>
<td>5</td>
<td>Update w_i (equations (5)–(8)) using ŷ and s_t</td>
</tr>
<tr>
<td></td>
<td>(5) return to step (2)</td>
</tr>
</tbody>
</table>

In order to validate the simulation results, we also tested the HRL controller using real neural data recorded from two nonhuman primates (PR and DU) during experiments in which they controlled reaching movements of a robot arm. During these experiments, the monkeys (common marmosets, *Callithrix jacchus*) utilized a Go/No Go motor task to move a robot arm to spatial targets on the monkeys’ left (A trial) or right (B trial) side. The monkeys started each trial by putting their hand on a touchpad and keeping it motionless for a random hold period (700–1200 ms). At this point, an audio ‘start’ tone was delivered, the robot arm moved to a start position facing the monkey, and LED lights for either the A or B target were illuminated. The monkeys received food rewards when they moved the robot to the illuminated target. To move the robot to the A (Go) target, the monkeys were required to reach and touch a target sensor within a 2 s time limit. To move the robot to the B target (NoGo), the monkeys had to keep their hand motionless on the touchpad for 2.5 s. The proportion of A and B trials were kept roughly equivalent, and they were presented in a pseudo-random order. Periodically, catch trials were used to ensure the monkeys understood the necessity of the robot movements. During the catch trials, the robot moved in the opposite direction commanded by the monkey and thus the monkey received no reward. No catch trial data was included in the offline analyses in this work.
Both monkeys had been implanted with two 16-channel microwire arrays (Tucker Davis Technologies, Alachua, FL) to record neural data. Neural data was acquired at 24,414 Hz using a Tucker Davis Technologies RZ2 system (bandpass filtered between 300 Hz and 5 kHz). One array was implanted in the motor cortex (M1), using stereotaxic coordinates and motor mapping to target hand and arm regions, and the other was implanted in the striatum. A skull screw served as ground and reference (although all signals for each array were also re-referenced using a common average reference to improve SNR). Only data from the M1 array was used in the offline HRL tests in this work. Specifically, for each monkey 20 M1 single and multiunit neural signals were isolated in real-time using the waveform shapes and manually defined thresholds. The firing rate over a 1 s time interval following the start of movement in A trials and start cue in B trials was used as the input neural vector for the offline HRL tests (see section 3.3 below for details of the decoding task). Data from three experimental sessions (taken across a week) were used from each monkey. Microwire implant surgeries were conducted under sterile conditions in a dedicated surgical suite using isoflurane (PR) or continuous ketamine infusion (DU) anesthesia, with cefazolin and buprenorphine administered postoperatively. All the procedures were 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.
Figure 4. Sequential neural decoding in the pinball task. The controller was initialized with random parameters at the beginning of the experiment. (A) Success rate during multi-step reaching (pinball) task, each bar indicates success. In each trial a new target at a random location in the 2D workspace was presented. The ratio of the number of steps used by the controller to the number of steps of the shortest path to the target was computed as the deviation from the optimal trajectory. The system had converged to the optimal path after eight trials. (B) The network weight trajectories at the hidden layer and output layer show the controller stopped changing the parameter after learning the optimal neural to motor mapping.

3. Results

3.1. Sequential learning performance

The structure of the actor network consisted of 25 nodes at the input (one for each synthetic neuron), 4 nodes at the output (one for each of the output actions), and 5 nodes at the hidden layer. This was the minimum number of nodes that yielded the optimal performance, i.e. we observed empirically that increasing the number of nodes did not increase the performance. A bias term was included at the hidden and output layers of the network. The network weights were initialized randomly at the beginning of the experiment, and were continuously updated during the task. Figure 4(A) shows
controller (figure 4) was able to learn a new control policy and reach the same level of performance as before perturbation.

3.2. Episodic learning performance

We tested the episodic learning performance of the HRL controller using the closed-loop simulation setup in figure 3(B). The experiment consisted of 100 trials. In each of the first 25 trials, one of the two targets (T1 and T2 spanning the horizontal 1D line) was presented randomly. The controller was initialized with random parameters at the beginning of each experiment. In each trial a new target at a random location in the 2D workspace was presented. Each bar shows success during multi-step reaching (pinball) task. The ratio of the number of steps that the controller by the number of steps in the shortest path to the target was computed as the deviation from the optimal trajectory. The horizontal solid red-line shows when the controller was following the optimal trajectory. Although the controller learned to complete the reaching task in two trials, it took 11 trials to learn the optimal mapping between the neural states and actions. The controller continued to follow the optimal control policy even without adaptation (after trial 15). After trial 25 the order of the neurons in the input neural vector was shuffled, rendering the previously learned control policy ineffective. From trial 25 to 35 the controller was not able to complete the task in any trial because there was no adaptation. By resuming adaptation (trial 35) the controller relearned to complete the task in one trial and converge to the optimal control policy after seven trials.

The performance of the controller both in terms of its success rate over time, and in regards to the optimality of the reach trajectories during a representative experiment consisting of 50 trials. The time to the target in each trial was computed based on the ratio of the minimum number of steps to the target (shortest reach trajectory) and the number of steps that the controller actually took to reach the target. In the pinball task target locations were selected randomly, therefore the number of steps to the target was different in each trial. In this experiment the shortest and longest reach trajectory were 25 and 80 steps, respectively (mean ± SD: 41.1 ± 13.9). The horizontal solid red-line is a reference that indicates whether the controller followed the optimal trajectory in a given trial or not. The controller had learned to complete the reaching task in three trials, and had found the optimal mapping between the neural states and actions in eight trials. It then followed the optimal control policy afterward, completing the task with 100% accuracy. The weight trajectory of the controller (figure 4(B)) shows that after learning the task the controller stopped changing the weights and the control policy was consolidated.

In order to test the generalization performance of the controller, we fixed the weights of the network after convergence and then introduced input perturbations. In these experiments the controller was again initialized using random weights. In figure 5 we can see the controller learned to complete the task in two trials and converged to the optimal control policy after ten trials. The controller continued to follow the optimal control policy after the parameters had been fixed (trial 15) and continued to complete trials successfully without need for adaptation until trial 25. Following trial 25, we perturbed the control policy by shuffling the tuning of the neurons. That means the mapping between the neural pattern and the desired actions was completely changed. Figure 5 shows that the controller did not complete any trials following the perturbation until we allowed the network to once again start adapting (trial 35). Once adaptation recommenced, it was able to learn a new control policy and reach the same level of performance as before perturbation.
the weight trajectories of the controller at the hidden and output layers (figure 6(C)): as the network converged to an effective control policy the weight trajectories plateaued. The dashed vertical lines show the point at which the control task was changed. At trial 25, the controller had no information about the task change (i.e. the presentation of T3 and T4 as new targets). The only information that led the controller to change its control policy was the evaluative feedback. The negative evaluative feedback caused an increase in the variance of the weight change magnitude in equation (5) and automatically forced the controller to search for a new control policy. In figure 6, this adaptation is reflected both in the action-value assignment and the weight update trajectories.

We tested the effect of memory size and the network size on the performance of the controller in the episodic learning mode using 100 Monte Carlo (MC) simulations [47]. Each simulation consisted of 50 trials and the controller was initialized with a different set of random weights drawn from the same uniform distribution. The convergence criterion was defined as at least 95% decoding accuracy over a block of 20 consecutive trials. The memory size was the number of the past trials stored in the replay database (see section 2.2). Figure 7(A) shows the effect of memory size on the learning properties and the generalization performance of the controller. When using the shortest memory (keeping only the previous trial), the controller was able to learn simpler tasks, T1-T2 horizontal and T2-T3 vertical, in 64% and 60% of the MCs, respectively. However, controller was only able to generalize to more complex tasks (4-target and pinball tasks) in less than 10% of the MCs. By increasing the memory size to 70 trials the generalization ability of the controller improved, but beyond that point it started to decline. We also tested the effect of network size (number of hidden layer nodes) on the learning and generalization performance of the controller. In this analysis, the network input size was 25 and the network had 4 output nodes. The empirical results in figure 7(B) show...
the optimal number of hidden layer nodes in this experiment was 5. The network with three hidden nodes was more sensitive to the initial condition of the parameters in terms of generalization and had more trouble learning the new task during adaptation. By increasing the number of hidden nodes to 10, the performance of the controller dropped both in terms of learning a new task and generalization to more complex tasks.

3.3. Primate neural decoding

We used neural data recorded from two monkeys over three sessions (for each monkey) to conduct offline analyses to further validate the decoding performance of the HRL algorithm. The decoding task was to reconstruct the robot arm trajectory (section 2.4) by sequential mapping the monkey’s neural firing rate (states) onto one of four movement directions (actions) over multiple steps. Each step in the input state-space was a vector of the firing rates of 20 M1 neurons in 100 ms bins. In each trial, one of the two fixed targets (T1 and T2) in a 2D grid space was presented to the controller correspond to the robot target location (A or B) during the original robot control experiment. Each step in the action space was moving to one of the four (left, right, up or down) adjacent points in the 2D grid-space. The controller received a +1 or −1 evaluative feedback after each action depending on whether the distance to the target was decreased or not. The direct path between the initial (center) position and each of the targets was four steps in the grid-space. The controller was free to move to any point in the grid-space; however, if the task was not completed in eight steps the trial was marked as failure and a new trial started.

Figure 7. The effect of (A) memory size and (B) number of hidden nodes in the network on the training and generalization performance of the controller, using 100 Monte Carlo simulations. The y-axis shows the percentage of Monte Carlos that were successful. The memory size corresponds to the number of past trials (input-action-feedback) that were logged for the experience replay algorithm during adaptation. In each plot the dark and light blue bars correspond to the adaptation phase in the episodic learning task (see figure 6(A)). The yellow and red bars show the generalization performance of the controller in the four-target classification and the pinball tasks. The optimal memory size for generalization performance in this task was 70, however by increasing the memory size, it became harder for the network to learn the new task (light blue, vertical 2-target task) during the adaptation phase. The optimal number of hidden layer nodes was five in a network with 25 inputs and four outputs. Increasing the number of hidden layer nodes had an adverse effect on the generalization performance as well as learning the new task during the adaptation phase.

The controller parameters were initialized randomly at the beginning of the experiment and the controller continuously adapted over each of the three sessions for each monkey. As demonstrated in figures 8(A) and (B), for both monkeys the controller converged after 20 trials and reached above 95% decoding accuracy. The order of target presentations was randomized throughout the offline experiments and it was comparable with the order of trials in the real experiments. After convergence, the HRL controller found the shortest path to the target in 95% and 87% of the trials in monkey PR and DU, respectively. The number of steps in each trial decreased over time as the controller learned an effective control policy based on the input neural states, and the controller often followed the optimal policy (direct path to the target).

The effect of day-to-day variability on the performance of the controller was tested by stopping the adaptation of the controller after day 1 and testing the performance in the data from days 2 and 3. In monkey DU, without adaptation, the performance dropped from 100% to 64% for day 2 and from 96% to 62% for day 3. In monkey PR, the decoding performance slightly dropped from 95% to 90% for day 2 but the performance drop was more profound in day 3 (from 98% to 67%). To further test the effect of adaptation, after fixing the parameters of the controller at the end of day 1 to prevent adaptation, a perturbation was introduced by shuffling the order of neurons in the input neural vector. As a result of this perturbation, figure 8(C) demonstrates that the decoding performance dropped to 54% and 48% in days 2 and 3, respectively, for monkey DU. Likewise for monkey PR, the decoding performance declined to 52% in day 2 and 55% in day 3 (figure 8(D)). The number of steps in the success
Figure 8. Sequential neural decoding performance of the HRL algorithm using neural data from two monkeys over three days. The decoding performance was quantified by the success rate (decoding accuracy) in reaching the target and the length of reach trajectory (number of steps to the target). In both monkeys the controller was initialized with random parameters at and the data from three experiment sessions was streamed sequentially to the controller. Plots (A) and (B) give the performance in the presence of continuous adaptation, and (C) and (D) show the effects of fixing the parameters of the controller and reorganizing the input by shuffling the order of neurons at the input at day 2.

In the presence of continuous adaptation, the controller was able to recover performance following perturbations on day 2 by readjusting its parameters and modifying the mapping between the neuronal activity and action values in both monkeys (figures 9(A) and (B)). The controllers were initialized at the beginning of day 1, and the smooth trajectories throughout the early trials in figures 9(C) and (D) indicate that the controller converged to an effective control policy and reached steady-state after about 65 iterations in monkey DU and 110 iterations in monkey PR. At the beginning of day 2, a perturbation was introduced to the controller by shuffling the order of the input neurons. In both monkeys the controller reorganized its parameters to adjust the mapping between the neural states and action values to maintain the performance. Depending on the operating point of the controller in the input space, the level of readjustment varied between the two monkeys. In monkey DU, the controller had a smooth readjustment and converged after 1 trial; however, in monkey PR the readjustment was more profound and it took longer for the controller to converge to its new control policy.

(4 steps) and failure trials (8 steps) following the perturbation demonstrate that the controller always took the direct path to target T2. This means that the controller was not able to distinguish between the T1 and T2 states after the perturbation, and so followed the same control policy for both neural states.
Figure 9. The decoding performance and parameter trajectories of the controller in response to the neural input reorganization (after day 1) during sequential adaptation in the multi-step reaching task. (A, B) The decoding performance was quantified by the success rate and the number of steps to the target in reaching tasks using the neural data from two monkeys. The input reorganization was introduced by randomly shuffling the order of neurons in the input neural vector at the end of day 1. (C, D) The weights of the network were initialized randomly at the beginning of the experiment. The variance of the weight trajectories decreases (at iteration 80 in monkey DU and 110 in monkey PR) as the network learns the effective mapping between the neural input and actions to maximize the positive evaluative feedback. By introducing a neural reorganization at the input (at the end of day 1), the optimal control policy changes and the controller readjusts its parameters to maintain performance. It is interesting that in monkey PR the controller underwent another readjustment at the beginning of day 3 without an external perturbation. There was a four-day gap between session 2 and 3 in monkey PR, and the reorganization could reflect the day to day variability of the input neural vector.

4. Discussion

The main motivation of the current work was to develop an adaptive controller that could learn the mapping between neural states and optimal actions without need for a detailed descriptive feedback (desired response). The HRL algorithm exhibited efficient decoding performance using a binary evaluative feedback with the assumption that such an evaluative feedback is readily available. For autonomous adaptation of the neuroprosthetic controllers the evaluative feedback should be recorded directly from the user. There are two approaches to this problem: either the user may consciously provide a binary evaluative feedback or the feedback could be recorded directly from the brain. In the former case, the feedback may be signaled by the user through eye-blink, muscle twitch or tongue movement \cite{48, 49} whenever the user is not satisfied with the neuroprosthetic performance. The lack of such user’s negative feedback could then be interpreted as a positive reinforcement by the controller. The HRL algorithm is especially suitable in this scenario because after convergence positive reinforcement will
not change the control policy. In other words, the controller and the user will reach to an equilibrium state.

Finding effective means of estimating the user’s feedback directly from the brain is a non-trivial problem which is beyond the scope of this paper, but the neural representation of reward expectation in multiple brain areas could prove suitable for this purpose [26, 50]. The striatum, cingulate, and orbitofrontal cortices are candidate structures for extracting the evaluative feedback from the brain [51]. Error-related potentials in EEG also may provide a source of information for recording evaluative feedback [52]. In the actor–critic theory of information processing in the basal ganglia (striatum) has been proposed as a structure that computes an internal evaluative feedback [24]. In particular, the role of the ventral striatum as the interface between the limbic and motor systems [53] makes this structure a suitable candidate for motor neuroprosthetic control applications. We are studying the neural representation of reward expectation in the striatum in response to the performance of the prosthetic function [54].

We introduced the HRL as an alternative to the supervised learning adaptation paradigms commonly used for neuroprosthetic systems. However, these two paradigms use similar mechanisms for adaptation. In both of the HRL and supervised learning paradigms the controller will use the instantaneous gradient of the performance to modify its behavior. It has been demonstrated that the HRL algorithm was equivalent to the error backpropagation in supervised learning for training a connectionist network [55]. The HRL algorithm provides a general learning framework for connectionist networks in associative tasks by combining elements of supervised classification with reinforcement-based optimization (learning automata). In fact, in connectionist networks with stochastic units, supervised learning may be viewed as an extreme case of the HRL where the output of each unit is binary and there is one correct output for each input [40].

The episodic and sequential modes of learning demonstrated that the HRL controller was able to handle both classification and regression tasks using simulated neuron populations. The results of off-line primate neural decoding indicated that the HRL algorithm was able to learn to complete the task by distinguishing between neural states and selecting appropriate actions in sequential learning mode. We have also completed basic tests of the episodic mode of the HRL algorithm during primate experiments for closed-loop neural control of a robotic arm in a binary choice task [56]. The basic behavior of the HRL algorithm in those closed-loop control experiments was generally consistent with the simulation results in this work (figure 6(A)). In the closed-loop experiments, the controller used a binary evaluative feedback to learn the mapping between the M1 neural states and robot actions after only a few trials (2–3), from a naive state, and the weight update trajectories plateaued after convergence.

Using the concept of motor primitives, our results highlighted the duality between the episodic and sequential learning modes in motor neuroprosthetic control. In motor control theory, motor primitives are considered as the building blocks of complex movements that have neural correlates in the central nervous system [57, 58]. The simulation results in this work demonstrated that the controller was able to decode movement trajectories either by learning to decode the motor primitives first or through direct interaction with the sequence of motor commands. In this work, we used movement direction and neural tuning as the motor primitives and their neural representation respectively. While this was a simple form of motor primitives, the HRL controller might be able to use complex motor primitives to decode more complex motor behavior from the brain neural activities.

One of the important features of the HRL controller is that it will adapt only if the performance is not satisfactory. If the controller converges to an effective control policy, the adaptation process will stop automatically in spite of continually receiving a positive evaluative feedback. In other words, the controller will adapt if the user is not satisfied with the performance. This feature of the controller is important from the perspective of consolidation of motor maps in neuroprosthetic control [59]. Although the negative evaluative feedback may trigger a change in the controller parameters, the results of our closed-loop experiments indicated that in presence of stable and repeatable neural states at the input the controller was robust to mislabeled feedback [56]. In those experiments the controller matched the behavior of the monkey even if the negative feedback was the result of the monkey occasionally not performing the task correctly.

The artificial neural network with binary nodes in this work implemented a deterministic control policy, however by using alternative processing elements in each node (e.g. stochastic Bernoulli nodes) stochastic policies could also be implemented. One of the benefits of using stochastic nodes in the network is that it provides an intrinsic exploration mechanism by adjusting the stochasticity of the nodes in the network. Another advantage of using a stochastic control policy would be decreasing the sensitivity of the controller to inaccuracies in the evaluative feedback and/or motor neural states. This would become important if using evaluative feedback estimated from the brain, because the acto performance will be less affected by any inaccuracies in the critic feedback decoding.

The four-action reaching task in a grid space in this work can be extended to a continuous domain by increasing the spatiotemporal resolution of the control task. By increasing the temporal resolution the input neural vector will be estimated using finer time bins, however there is a physiological limit in terms of estimating meaningful neural states by decreasing the size of time bins. To overcome this limitation, using function approximators that can operate directly in the space of spikes could be beneficial [32]. Another alternative is to increase the spatial resolution of movement by increasing the number of actions. The downside of this approach is that by increasing the number of actions, the size of the action space of the controller will increase, which will adversely affect the speed of search for the optimal solution. There is a tradeoff between the temporal resolution and spatial resolution in constructing the continuous trajectory. By using hierarchical or modular structures the search in the action space can be improved and the decoding approach can be expanded in the future [60, 61].
5. Conclusion

We presented a control architecture and learning algorithm for adaptive motor neuroprosthetic systems based on the concept of HRL. The decoding performance of this learning control method was evaluated in simulated closed-loop control and offline decoding of primate neural data. These results indicate that a neuroprosthetic user using the HRL controller could control neuroprosthetic actions not only based on the neural representation of motor states at the input of the controller but also using an evaluative feedback to adapt and maintain the system. Furthermore, the HRL adaptive controller can learn optimal control policies using only a binary evaluative feedback that indicates whether the actions of the controller increased the probability of achieving the goal or not. In this work, the evaluative feedback was simulated using the discrepancy between the user’s expected movement and prosthetic actions. If such feedback information could be obtained directly from the user (brain, eye blinks, muscle twitch, sip-puff, etc), this approach could be used to develop fully autonomous neuroprosthetic systems.

Acknowledgments

This work was supported by DARPA REPAIR Contract #N66001-10-C-2008.

References

[12] Li Z et al 2011 Adaptive decoding for brain-machine interfaces through Bayesian parameter updates Neural Comput. 23 3162–204
[22] Joel D, Niv Y and Ruppin E 2002 Actor-critic models of the basal ganglia: new anatomical and computational perspectives Neural Netw. 15 535–47
[36] Li Z et al 2011 Adaptive decoding for brain-machine interfaces through Bayesian parameter updates Neural Comput. 23 3162–204
[38] Rotermund D, Ernst U A and Pawelzik K R 2006 Towards on-line adaptation of neuro-prostheses with neuronal evaluation signals Biol. Cybern. 95 243–57
[48] Houk J C, Davis J L and Beiser D G 1995 Prediction in primate basal ganglia and frontal cortex Neuropharmacology 37 421–9
[52] Pennartz C M A 1997 Reinforcement learning by Hebbian synapses with adaptive thresholds Neuroscience 81 303–19
[59] Li Z et al 2011 Adaptive decoding for brain-machine interfaces through Bayesian parameter updates Neural Comput. 23 3162–204
Learn. 8 229–56
SMC-15 360–75
4 R1–13
4 237–85
Syst. Man Cybern. C 42 201–12
22 1484–97
Biomed. Eng. 51 943–53
[51] MacDonald A W et al 2000 Dissociating the role of the dorsolateral prefrontal and anterior cingulate cortex in cognitive control Science 288 1835–8
[61] Barto A G and Mahadevan S 2003 Recent advances in hierarchical reinforcement learning Discrete Event Dyn. Syst. 13 41–77