Towards Real-Time Distributed Signal Modeling for Brain-Machine Interfaces

Jack DiGiovanna[1], Loris Marchal[2], Prapaporn Rattanatamrong[2], Ming Zhao[2], Shalom Darmanjian[2], Babak Mahmoudi[1], Justin C. Sanchez[3], José C. Príncipe[2], Linda Hermer-Vazquez[4], Renato Figueiredo[2], and José A.B. Fortes[2]

[1] Dep. of Biomedical Engineering, [2] Dep. of Electrical and Computer Engineering,
[3] Dep. of Pediatrics, [4] Dep. of Psychology
University of Florida, Gainesville, Florida, USA
{fortes, renato, lindahv, principe, jcs77}@ufl.edu

Abstract. New architectures for Brain-Machine Interface communication and control use mixture models for expanding rehabilitation capabilities of disabled patients. Here we present and test a dynamic data-driven (BMI) Brain-Machine Interface architecture that relies on multiple pairs of forward-inverse models to predict, control, and learn the trajectories of a robotic arm in a real-time closed-loop system. A method of window-RLS was used to compute the forward-inverse model pairs in real-time and a model switching mechanism based on reinforcement learning was used to test the ability to map neural activity to elementary behaviors. The architectures were tested with *in vivo* data and implemented using remote computing resources.

Keywords: Brain-Machine Interface, forward-inverse models.

1 Introduction

Among other applications, Brain-machine interfaces (BMIs) for motor control can potentially enable patients with spinal injuries to regain autonomy as well as create new and revolutionary forms of man-machine interaction. Research is providing an increased understanding of how brains control and learn movement, how neuronal signals can be detected and processed, and how computers can be used to control artificial limbs in real time. This paper reports progress of our work in these fronts.

In [2] we described a BMI architecture that relies on pairs of forward-inverse models to predict, control and learn the trajectories of a robotic arm in a real-time closed-loop system. Section 2 extends it into a dynamically data-driven adaptive system (DDAS) where the choice and learning of computational models is determined by predicted model responsibilities and observed trajectory errors. The experimental setup is described in Section 3. The real-time computation and learning of the models require the matrix operations and adaptive algorithms discussed in Section 4. Experimental results and conclusions are reported in Sections 5 and 6.

Y. Shi et al. (Eds.): ICCS 2007, Part I, LNCS 4487, pp. 964–971, 2007.

2 A Mixed-Model DDDBMI Architecture

The basis of our design methodology for BMIs is rooted in a model-based approach to motor control proposed by Kawato and Wolpert [1, 3]. The major difference is that for BMIs the inputs are a mixture of neuronal outputs from motor cortex and feedback signals which are processed by computer hardware to generate an output to a prosthetic arm (Fig 1). The architecture uses the concept called the Multiple Paired Forward-Inverse Model (MPFIM) which consists of multiple pairs of models, each comprising a forward model (for movement planning) and an inverse model (for movement execution). Individual *model-pairs* or combinations of model-pairs control motion on the basis of feedback data (visual or proprioceptive).

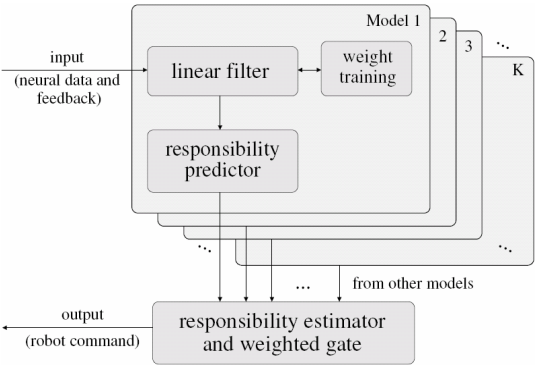

Fig. 1. Multiple paired models and the responsibility predictor for the DDDAS based BMI

In the DDDAS framework, the feedforward models are implemented with conventional optimal input-output mappers (adaptive FIR filters) using supervised learning [4]. Therefore, we include both the neural signal and musculoskeletal transformation of those signals to prosthetic joint actuations (lookup table) in our inverse model. The MPFIM architecture requires a Responsibility Predictor/Estimator (RP) to serve as the gating mechanism to select which local 'expert' (forward-inverse model pair) is most appropriate based on the current system inputs. In the original architecture, the RP uses 'contextual signal' such as 'sensory information and cognitive plans' to determine the responsibility of each local expert at the current time based on the state of the motor control system and environment [1]. Research suggests that the a representation of the state of the motor control system is present in the motor cortex [5]. Following this view, this work estimates the firing rates from the forelimb area of primary motor cortex [6] to create a contextual (state) signal.

The RP is trained to map neural state to the appropriate forward-inverse model pair. That model pair generates an elementary action (or moveme analogous to phoneme) for the prosthetic limb. To determine the model-pairs for each neural state, we use a modified version of reinforcement learning (RL), which provides a learning mechanism that is similar to those of a biological motor system [7]. Indeed, the learner is not told what actions to take but must discover which actions yield the most reward by trying them [8], very similar to operant conditioning in psychology.

Conceptually, RL teaches an agent a policy to interact with their environment in a fashion that maximizes reward over time [8]. RL requires 3 signals to try to learn the optimal policy: the 'state' of the environment, the 'reward' provided by the environment, and the 'action' taken by the agent. From the BMI perspective, 'reward' is achieved by reaching a targeted position. For RL, a positive reward signal is generated for task completion (i.e. reach-and-grasp); to encourage efficiency, the reward signal is negative for actions that do not immediately complete the task.

3 Experimental Design

To investigate and exploit the distributed representation of neurons in the rat motor cortex and implement them in a novel computational framework, we have developed a novel experimental design involving a "reach and grasp" task where a cue is given to initiate controlled movement of a robotic arm from a rest position to one of two targets. This paradigm is designed to be comparable to a real situation where the patient's neural signal and rewards are known but kinematic variables are unavailable. We developed a "testbed" behavioral task that rats could perform stereotypically across trials, that could be relatively easily modeled using our grid computing approach, that would facilitate rats' switching from arm control to brain control, and finally, that could be modified to introduce brain control of the MPFIM without the availability of a desired response, as is conventionally done in neurorobotic research. The animal training paradigm (see Fig. 2) was designed to gradually shape the rats into the desired reaching behavior. Neural firing rates from a rat were recorded using a Tucker-Davis (Alachua, Florida) Pentusa system during a go no-go behavioral task as described in [9]. The task was defined such that the animal initiates all trials (nose poke breaking an IR beam). The animal must press a lever in the behavioral box while a LED cue is present.

Additionally, a robotic arm moves in a 3-dimensional grid to target levers within the rat's field of vision [10] but in a discrete area (outside the cage) that the rat cannot reach. To earn a reward, the rat and robotic arm must press and hold their respective levers for 250 ms simultaneously. Performance of this task must exceed 80% before

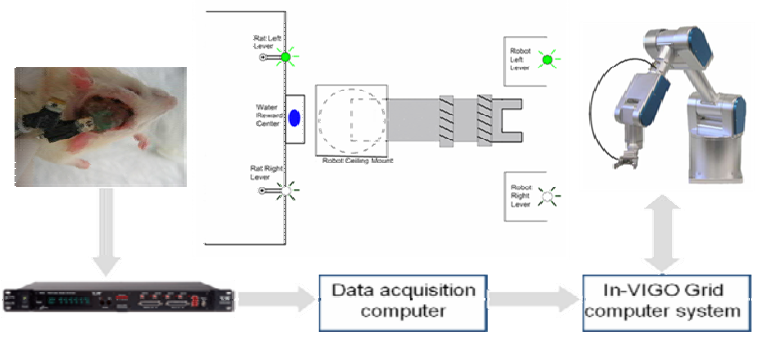

Fig. 2. Animal Experimental Paradigm. The rat is in the cage on the left side. In brain control, the rat levers are not extended, the robotic arm is the only way to earn rewards.

electrode implantation or the animal was excluded from the study. This criterion ensures that the animal is discriminating between cues rather than guessing. Video analysis confirmed (> 95% trials) that the animal presses with the paw contralateral to the lever; ispilateral neural firings reflect this difference in behavior. In this reach-and-grasp task all data is time synchronized to the Pentusa's clock (neural signal, reward times, and lever position) and sent to remote computers for analysis. However, we do not record the kinematic variables of the animal's arms.

4 Computational Framework

We now turn to the question of how to train the forward-inverse models pairs and the gating mechanism (i.e. the individual likelihood models and the RP) of Fig. 1.

4.1 Weight Training for Linear Models

The output of an optimal linear filter of order p is computed by $Y = W^T \times X$ where X is the vector of the last p input data and W is the matrix of weights. To train the weights, we are given the past values of X and Y on a large time window (also called the training window). A common method is then to find the weights W that gives the smallest sum of square errors on Y when applied on the X values (least squares method). When all data are available at once, the weights can be computed by $W = R_X^{-1} \times r_{XY}$ where R_X is the auto-correlation matrix of signal X, and r_{XY} the cross-correlation matrix between X and Y. Computing the auto-correlation matrix and its inverse requires a huge amount of computation, especially when the training window is large. This is the reason why we focus on recursive updates to compute the optimal weights, specifically the recursive least square (RLS) algorithm [11].

The principle of the recursive least square method is to obtain rank one updates of both for R_X^{-1} and r_{XY} at each time a new data vector is received. Many variants exist such as the sliding window-RLS [14] or the QR-RLS [15], which uses a QR factorization of the auto-correlation matrix. Because of the requirement of large training windows (up to 10,000 samples), we adopt the sliding window-RLS algorithm, which is based on the Woodbury formula [13]. We implemented a general algorithm, providing the following features: (1) multidimensional inputs and outputs and separated modules for auto- and cross-correlation recursion, so that the same signal can be used to train a lot of different models at low cost; (2) strict sliding window, the matrices are updated with new data and "downdated" with old ones; (3) ability to use a forgetting factor to increase the importance of recent data.

This algorithm is specifically designed for online training, by updating its internal data structure at each data received. It works on a special set of weights, called "training weights" and does not interfere with current model computations. When the user asks for it, or at predefined time-steps, the weights of the model computation are updated and replaced by the training weights. The RLS algorithm could also be used on offline data, to initialize a set of weights on data recorded in a past experiment. However, the RLS algorithm is not efficient for this purpose, so we also provide an implementation of the block least square (BLS) algorithm for offline training. Fig. 3 shows the raw neural data along with the lever press and a linear model output.

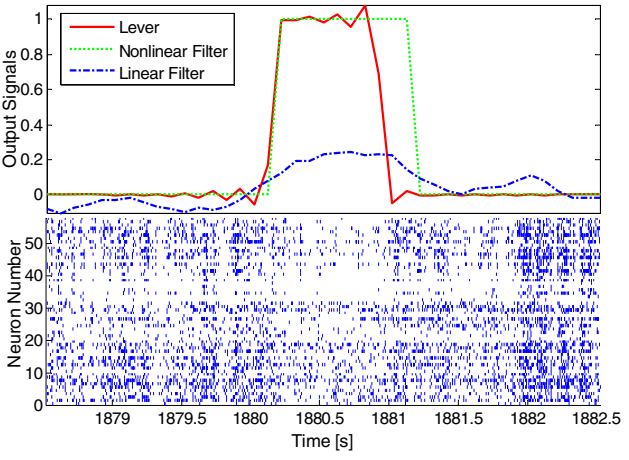

Fig. 3. Representative lever press, linear, and non-linear filter output for this paradigm [top]. Raster plot of recorded neural firing during the lever press [bottom].

4.2 Inverse Model Computation

In the initial testing of this DDDBMI architecture, the set of possible desired trajectories is restricted to six movemes (or simplified models of moving forward, back, up, down, left, and right) of equal length. The starting position of the robot prosthetic is also constant for all trials. This allows us to compute a lookup table of necessary joint actuations for any possible prosthetic orientation – desired trajectory pair in our workspace. This lookup table is only a placeholder for future inverse models where the initial positions may not be known a priori and the set of movemes may be modified through training.

These future inverse models would require an optimization using a biomechanical model of the prosthetic limb to be controlled. Such an optimization averages 24.5 function evaluations using Matlab's fmincon[1]. Computing a single inverse model (performed every 100 ms) requires, on average, 784 multiplications, 1054 additions, and 343 trigonometric function evaluations.

4.3 RL Computation

Here we train the switching mechanism through an application of RL. In the classical embodiment of RL, a model of the animal's brain would be considered an 'agent' which learns (through RL or other methods) to achieve a goal or reward. Here the states of the environment and rewards are used to drive new actions (i.e. which model pair to select). In a BMI experimental paradigm though, one has access to not only the real animal brain but can observe the spatio-temporal activation of brain states (indirectly related to the environment) as the animal seeks a goal or reward. Thus,

[1] fmincon is a function provided by the MATLAB Optimization Toolbox for constrained nonlinear optimization; it finds a constrained minimum of a scalar function of several variables starting at an initial estimate.

from the BMI point of view, neural activity is external to, and can not be directly modified by, the agent (algorithm); it must be considered part of the environment [8]. The BMI RL agent must use information in the neural signal to create movement commands for the robot; therefore the agent must learn the state-to-action mapping.

We devised a novel architecture where the animal's neural firing is part of the environment and defines the state, actions occur in a discrete physical space (a separate portion of the environment), and the RL algorithm serves as the agent. The animal can see the actions of the robot in the environment; however, actions do not directly influence the state of the rat brain. This is an important shift in RL architecture because states are decoupled from actions

5 Results

Presently, linear models are being studied; the software must support the following requirements. With about 100 neurons and a time window of 10 steps to compute a five-dimension output, each linear model roughly use 5000 weights. Thus the computing power needed for one model is small; however, we want to use hundreds of models and need to distribute this computation on the available resources. Additionally, each linear model is constantly training its weights as data is received, to improve its behavior. This training has to be done in parallel with the computation of the output, and may represent a huge amount of computation. The software for signal processing is composed of communicating processes, currently in MPI. One global process receives data from the Brain Institute and broadcasts them to a number of processes, each of them responsible for one model. For each model, a thread, running in background, is constantly training the weights. All the computation-intensive operations are implemented using function calls to BLAS (gotoBLAS) and LAPACK libraries. The timing results for a close-loop experiment on a 512MB Pentium 3 1.16GHz computing node using one model are the following:

Gathering data from the animal's brain	3.4 ms
Overall communication between both laboratories	0.87 ms
Computing the output of one model	0.20 ms
Inter-process communication and data management	23 ms
Robot Control	0.028 ms
Weight training	31.8 ms
From collection of brain data to robot control	28 ms

This shows that one closed-loop using only one model takes much less than the bound of 100ms, even when training the weights of the model on the same machine.

The MPFIM architecture tested here contains two fixed movemes (actions) and two target locations; thus, we can model the movements of the prosthetic in the environment to generate the reward signal for RL. The 'value' (responsibility in the MPFIM) of each moveme (forward-inverse model pair) is approximated by an artificial neural network (ANN) based on the animal's brain state. Watkin's Q(λ)

learning [8] was used to train the ANNs and decide which moveme is most appropriate given the current neural state. *In vivo* testing of the experimental paradigm and MPFIM architecture was performed in one dimensional space with two possible movemes (-1 or +1,i.e. left or right robotic movements). In this form of RL, we performed value function approximation using both single and multilayer perceptrons to test the performance differences for achieving the target. To earn a reward, the BMI needed to reach a target of -7 or +7 starting from position 0. There were 46 training trials (23 trials each for left and right targets) and 16 testing trials (8 targets for each direction). All trials contained 10 samples of neural data (1 s) which terminated in a water reward in the *in vivo* animal recordings. Three different ANN topologies were used to approximate the value of the neural state: single layer perceptron (SLP), multilayer perceptron (MLP-nonlinear output), and a multilayer perceptron with linear outputs.

After training the value function, the MPFIM architecture was tested on the 16 novel trials not used for training. Performance was evaluated as the percent of trials that the robot reached the targets. For this test, the null hypothesis is that the neural data contains no information about the movement intent of the animal and the MPFIM will learn regardless of neural state. To test this hypothesis, a surrogate data set was constructed by randomizing the temporal sequence of neural activity collected from all of the neurons. The results for real neural data and surrogate neural data are presented in table 1. The real neural data can hit the cued target between 81% and 94% of the trials. In contrast, the null hypothesis was shown to be incorrect through the performance of the surrogate data (hitting the cued target 31% of the time). Although we only present results from a simplified environment, we show that the RL does generalize to novel data and presents a viable technique for switching in the RP.

The training and testing performance of this architecture can also be measured in terms of its computational complexity. Temporal difference error [8] was used to train the networks where each error signal is applied to a history of past states (inputs). Therefore, the average number of weight updates per sample depends on the length of trials that the algorithm has to find a reward. For a 1s trial length, the algorithm averages 4.5 weight updates per step (100 ms). In animal experiments, we allow 6.5 s for the animal to earn a reward. This trial length averages 33 weight updates per step. For the best performing case (SLP), the weight training consists of multiplications on the order of $(4M+1)*A$ and adds on the order of $(M+1)*A$ for each update. Here M corresponds to the number of input neurons (ranging from 10s to 100s) and A is the number of actions. This example indicates that future architectures utilizing online training will require high-performance computing resources.

Table 1. MPFIM performance in 1D environment

	SLP	**MLP(NO)**	**MLP(LO)**
Real Data	93.7%	81.2%	81.2%
Surrogates	31.2%	31.2%	31.2%

[note: MLPs have 3 hidden PEs]

6 Conclusions

The DDAS BMI architecture can serve as a general-purpose neural–to-motor mapping framework with modular forward and inverse models with a semi-supervised gating mechanism. This architecture is well prepared to deal with the uncertainties in the nature and generation of motor commands. We showed how the capabilities of linear and nonlinear feedforward models can be implemented in real-time, simultaneously allowing one to determine the most appropriate functional form of the motor commands. Distributed processing provided the necessary computational power for performing *in vivo* experimentation on the timescales of the behaving animal. Implementation of the reinforcement learning for mapping to the elemental constructs of movement is a first step in the development of semi-supervised, goal-directed BMI techniques for paralyzed patients who are unable to generate a desired movement response. Future work will focus on scaling up the biomechanical modeling and behavioral response of the animal to 3-D robot control.

Acknowledgments

This work is supported in part by the NSF under Grant No. CNS-0540304 and by the BellSouth Foundation.

References

1. Kawato,M.,Wolpert, D. M.: Multiple paired forward inverse models for motor control. Neural Networks. vol. 11(1998) 1317-1329
2. Fortes, J., Figueiredo, R., Hermer-Vazquez, L., Principe, J., Sanchez, J.: A New Architecture for Deriving Dynamic Brain-Machine Interfaces. In: ICCS-DDDAS(2006)
3. Erhan Oztop, D. W., Kawato,M.: Mental state inference using visual control parameters Cognitive Brain Research.vol. 22(2004)129-151
4. Carmena, J., et al.: Learning to control a brain–machine interface for reaching and grasping by primates. PLoS Biology. vol. 1(2003) 1-16
5. Doya, K.:What are the computations of the cerebellum, the basal ganglia and the cerebral cortex?. Neural Networks.vol. 12(1999) 961-974
6. Nicolelis, M. ,et al.:Reconstructing the engram: simultaneous, multisite, many single neuron recordings. Neuron. vol. 18(1997)529-537
7. Bower, G. :Theories of Learning, 5th ed. Englewood Cliffs: Prentice-Hall, Inc.(1981)
8. Sutton, R., Barto,A.: Reinforcement learning: an introduction. MIT Press (1998)
9. DiGiovanna, J., Sanchez, J., Principe, J.: Improved Linear BMI Systems via Population Averaging.In: IEEE Conf. Eng. in Medicine and Biology Society (2006)
10. Whishaw, I.:*The behavior of the laboratory rat*. NY: Oxford University Press (2005).
11. Haykin, S.: *Adaptive Filter Theory*: Prentice Hall (2002)
12. Golub, G. H. and Van Loan, C. F.: Matrix computations. Johns Hopkins (1989).
13. Liu, H. and He, Z.: A sliding-exponential window rls adaptive filtering algorithm: properties and applications. Signal Process., 45(3):357-368 (1995).
14. Sakai, H. and Nakaoka, H.: A fast sliding window QRD-RLS algorithm. Signal Process., 78(3):309-319 (1999).