UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

PREDICTING ARM MOTION FROM CORTICAL ACTIVITY

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

DAVID IAN GOLDBERG
Norman, Oklahoma
2007

PREDICTING ARM MOTION FROM CORTICAL ACTIVITY

A THESIS APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

By

Andrew H. Fagg, Chair

Nicholas G. Hatsopoulos

Amy McGovern

Robert L. Rennaker II

Dedication

To my parents, Jan and Jennifer Goldberg, my Uncle, Michael Kaswan, and my Grandfather, Ed Kaswan. Also, to Dr. Andrew Fagg for enabling me to conduct this research.

Acknowledgments

Special thanks to Dr. Andrew Fagg for mentoring, teaching, guiding, and encouraging this endeavor. This work has been done in collaboration with Dr. Nicholas Hatsopoulos, Jake Reimer, Dawn Paulsen, and Dennis Tkach at the University of Chicago (Neural Data), Dr. Lee Eugene Miller, Northwestern University, Dr. Gregory Ojakangas, Drury University, and Shamim Nemati, University of Oklahoma. This work has been supported in part by the University of Oklahoma, the National Institutes of Health (Grant #R01 NSO48845), and the National Institute of Neurological Disorders and Stroke (Grant #R01 NS45853-01).

Contents

List Of Figures

Abstract

From injured soldiers returning from war, to patients suffering from peripheral vascular disease, there are a large number of people worldwide who benefit from prosthetic arms. However, amputees using current prosthetic technology require a long training period to use their prosthetic arm effectively. One reason is that amputees must learn arbitrary methods for actuating a prosthetic arm. In addition, they must rely on alternative sensory information (particularly vision) to monitor their arm movement, since they no longer receive proprioceptive feedback. One approach to addressing these difficulties is to command a prosthetic arm using some of the information that was used to originally command the intact arm. One of the major sources of this command information is the primary motor cortex. In this thesis, I examine the problem of how to predict intended arm motion from the recent activation of motor cortical neurons. Specifically, I compare three methods. In the method of Georgopolous' *Population Vectors*, one creates a model of discharge rate for each individual neuron as a function of movement direction. The set of cells then "vote" for direction based upon their firing rate. I also examine the Pseudo-Inverse solution to the Wiener filter, a linear method that minimizes mean-squared error on the training set model. Finally, I examine Least-Squared Kernel Regression with the set of polynomial kernels. I find that the Wiener filter predicts arm motion more accurately than Population Vectors in all cases. Equally important, I demonstrate a significant improvement of prediction performance in most data sets using Least Squares Kernel Regression with a quadratic kernel compared to the Pseudo-Inverse solution to the Wiener filter.

Chapter 1

Introduction

From injured soldiers returning from war, to patients suffering from peripheral vascular disease, there are a large number of people worldwide who benefit from prosthetic arms. However, amputees using current prosthetic technology require a long training period to use their prosthetic arm effectively. One reason is that amputees must learn arbitrary methods for actuating a prosthetic arm. In addition, they must rely on alternative sensory information (particularly vision) to monitor their arm movement, since they no longer receive proprioceptive feedback. One approach to addressing these difficulties is to command a prosthetic arm using some of the information that was used to originally command the intact arm. However, there are many challenges in this approach. The first challenge is how to represent the intended arm movement in the context of interpreting neural activity. A continuing argument in the neuroscience literature is how movement is encoded in the primary motor cortex. One of the key debates is whether this area encodes movement in terms of extrinsic parameters or in terms of how the musculature is recruited to implement the intended movement. In this thesis, I studied this question by comparing whether one can predict movement in terms of an extrinsic coordinate frame (such as Cartesian hand position) or an intrinsic frame of reference (such as shoulder and elbow torque) more accurately than the other.

Another challenge is identifying the kind of mathematical transformation from cortical activity to arm motion that yields the closest reconstruction to the intended movement. It stands to reason that the more accurate reconstruction is, the less time the patient will have to spend learning how to use the prosthetic device. One of the standard ways to make the transformation from cortical activity to arm motion

is *Population Vectors*, an intuitive method in widespread use (Georgopoulos *et al.*, 1982; Reina *et al.*, 2001; Schwartz *et al.*, 1988). For this method, one creates a model of discharge rate of a particular neuron as a function of movement direction. Given these models for a population of neurons, one then estimates the current movement direction of the arm based on the firing pattern of the population.

Alternatively, one can use the Pseudo-Inverse solution to the Wiener filter, a linear method that minimizes mean-squared error on the training data (Serruya *et al.*, 2002; Serruya *et al.*, 2003). In addition, this method can individually weight neurons in terms of their importance in making predictions about intended movement. To help identify the most effective unstructured mathematical model for reconstructing arm movement, I compare predictions of two linear models - population vectors and the Pseudo-Inverse solution to the Wiener filter.

We also know that the transformation from neural activation patterns to arm motion is inherently non-linear, since the dynamics and kinematics are non-linear functions of joint state. Therefore, it stands to reason that a non-linear model could achieve higher performance. Consequently, I evaluate the non-linear method of Least-Squared Kernel Regression (LSKR) with a set of polynomial kernels. Polynomial kernels capture the co-occurence of cell activity or specific temporal patterns. I compare the performance of LSKR with that of the Pseudo-Inverse solution to the Wiener filter.

Finally, once we predict arm movement, what metric most effectively compares the performance of different models? This metric should capture the ability of the model to predict the intended motion of the subject. In addition, since we are comparing predictions from extrinisic coordinates to intrinsic coordinates, we need a unitless metric. I compare the coefficient of determination, a metric widely used in the literature, to the Fraction of Variance Accounted For (FVAF). The coefficient of determination measures how linearly correlated the prediction is to the desired movement. The FVAF, a modification of the coefficient of determination, accounts for deviations between the desired motion and the predicted arm motion, as well as cancelling the measured units. Because the coefficient of determination can mask some types of prediction errors, we argue that the FVAF is a more realistic measure of arm movement prediction.

I examine each of these questions using six independent data sets (three macaque monkeys in total) of simultaneously recorded arm movement and neural spike rates. I find that Pseudo-Inverse predicts arm motion more accurately than the Population Vector approach in all cases. Equally important, I demonstrate a significant improvement of prediction performance in most data sets using Least Squares Kernel Regression with a quadratic kernel as compared to the Pseudo-Inverse solution to the Wiener filter.

Chapter 2

Background

The primary motor cortex gives rise to a large number of the corticospinal projections in primates. We therefore expect a relationship between cell activation and movement. Georgopoulos *et al.* (1982) show that motor cortical activation rates are related to arm movement direction. Figure 2.1 shows the behavior of a single neuron as a monkey moved her arm in eight, equidistant directions, all in the same plane. In each direction, there are five horizontal lines of action potentials, each corresponding to a separate trial. Time 0 represents the onset of movement. For this experiment, the neuron fired maximally prior to and during movement when the monkey moved its arm in the 135° direction. As the movement direction changed from 135°, the spike rate of this example neuron prior to and during movement decreased. Since this neuron responded differentially to a range of movement directions, Georgopoulos *et al.* (1982) described it as "directionally tuned." Moreover, one can model firing rate as a function of movement direction. Specifically, Georgopoulos *et al.* (1982) modeled firing rate as a linear function of the cosine of the angular difference between preferred and actual direction. In this case, the peak of the cosine was at 161°.

Based on this neural behavior, Georgopoulos *et al.* proposed that neurons have a "preferred direction," which was defined as the direction of movement for which the cell exhibits maximal activity. Further, they reasoned that, within a population of neurons, a subset of cells encode information about a given movement. Given the activity of a set of cells, one can use their measured preferred direction, in turn, to predict the direction of movement. In essence, each cell "votes" for a motion in its preferred direction, with the strength of its vote corresponding to the task-related

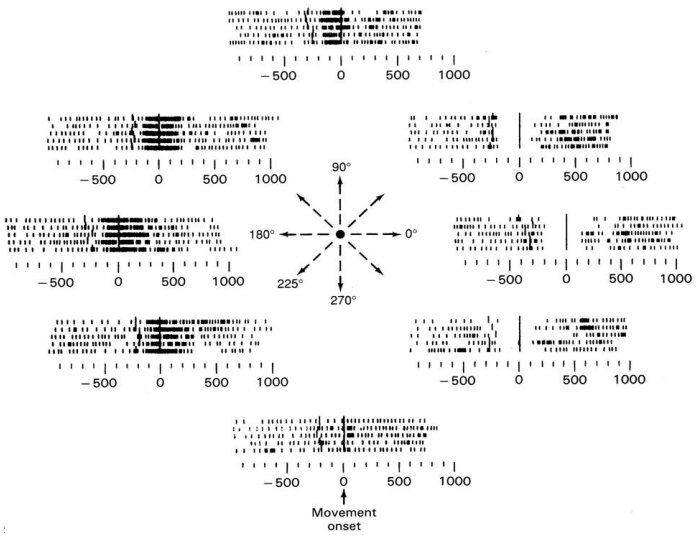


Figure 2.1: Cell Activity in a Center-Out Task (reprinted with permission; Georgopoulos *et al.*, 1982, Figure 4). Action potentials for one neuron as a function of time. The eight sections correspond to the spikes as the monkey makes planar arm movements in 8 equidistant directions. Each line in a section represents a different trial. Movement onset is at time 0.

cell activity. In addition, they suggested that direction is measured in terms of an extrinsic coordinate frame (specifically, Cartesian coordinates).

Anatomically, we know that some M1 cells project down the spinal cord to motorneurons, which in turn influence the activation of muscles. Thus, it stands to reason that M1 cells might best be thought of as encoding the dynamics of movement in an intrinsic coordinate frame – rather than an extrinsic, Cartesian frame. In order to investigate this further, Shadmehr and Mussa-Ivaldi (1993) examined whether humans make use of an intrinsic or extrinsic-oriented internal model when planning movement. Subjects performed center-out movements towards one of eight equidistant points of an outer circle, while holding a free-moving manipulandum. Morasso (1981) and Flash and Hogan (1985) had previously shown that subjects required to make movements with medium accuracy tended to make straight-line, smooth movements. The subjects in Shadmehr and Mussa-Ivaldi's experiments also exhibited this behavior. Shadmehr and Mussa-Ivaldi then imposed an artificial force field using a set of actuators exerting forces on the manipulandum. There were two

types of force fields, "E" and "I", as well as two different workspace regions where the subjects performed the center-out task, Left and Right. Field E was a function of the subject's hand velocity, and, from an extrinsic point of view, remained exactly the same in both the Left and Right regions of movement. On the other hand, field I was based on the velocity of the subject's shoulder and elbow joints. Therefore, field I exerted the same torques about the joints, in intrinsic terms, in both the left and right regions. The constant in the function of field I was carefully chosen so that both field E and I exerted nearly identical forces in the Right region. However, in extrinsic terms, field I was nearly orthogonal to E in the Left region. Subjects first learned to perform the task in the Right region, while subjected to field E. Following training, subjects repeated the task in the Left region. Half of the subjects were presented with field E, while the other half were presented with I. Shadmehr and Mussa-Ivaldi found that the subjects moved almost directly to the target in field I, while there was a "hook" in the movement in field E (Fig. 2.2). The hook was presumably due to on-line corrections made toward the end of the reaching movements.

Thus, after training with field E in the Right region, the subjects "expected" the intrinsic equal of it, field I, in the Left region. On the other hand, they did not anticipate the extrinsic equal of the force field E, since they did not perform straight-line movements under this field. A likely explanation for this behavior is that the subject's internal model was based in an intrinsic coordinate frame.

Scott and Kalaska (1997) examined the changes in discharge behavior of a monkey's motor cortex as a function of arm posture. If MI performs more abstract processing, encoding movement in an extrinsic, hand position frame of reference, then neither the discharge rate nor the preferred direction of an MI cell should vary based on arm posture. However, if MI encodes movement in an intrinsic frame of reference, then cortical activity should change significantly. Scott and Kalaska (1997) trained two monkeys to perform planar, center-out tasks in two separate postures. In the first posture, the subject's hand rested on a manipulandum in front of its body, positioned about one inch below its shoulders. The subject's elbow, pointed away from its side, rested on a piece of plexiglass that was positioned in a horizontal plane just below the subject's shoulder. In the second posture, the subject's hand

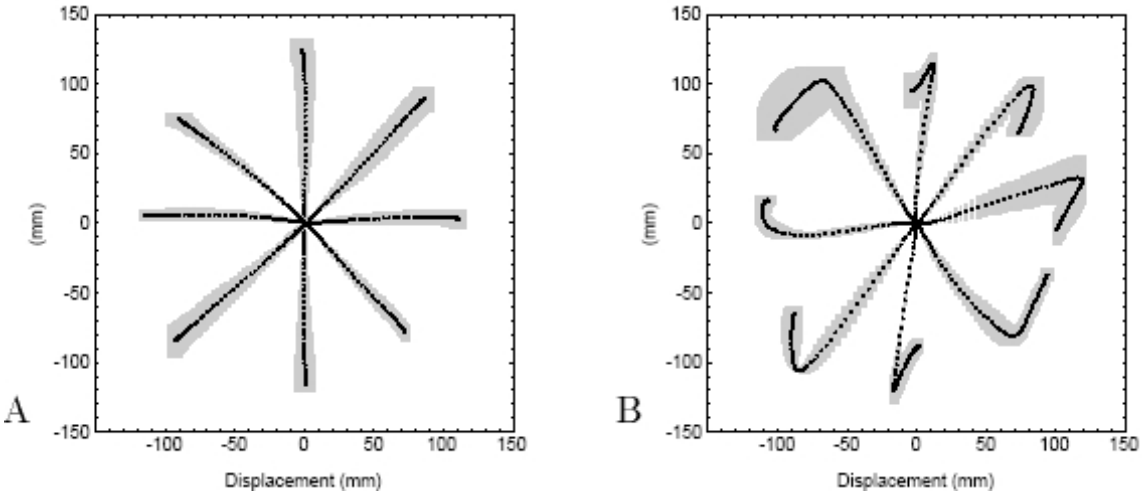


Figure 2.2: Movements in Different Fields in a Separate Region of Movement (reprinted with permission; Shadmehr and Mussa-Ivaldi, 1994, figure 15). Subjects initially performed center-out movements in an region of movement, called the Right region, under force field, E. This figure shows the averages ± SD of hand trajectories during movement in the Left region. Dots are samples taken at 10 ms intervals. *A* shows trajectories of subjects under the influence of the same force field, E. *B* shows trajectories of subjects under the influence of the force field I, whose forces in the Left region are orthogonal to E's forces in the Right region, from an instrinsic, joint angle point of view.

remained in the same position. However, the elbow and hand formed a vertical line parallel to the monkey's body, with the elbow in front of the subject's stomach.

Scott and Kalaska (1997) found that the activity level changed significantly between the two arm orientations for 95% of the cells. Specifically, more than 50% of cortical cells showed significant changes in overall discharge rates between the different arm postures during the initial hold time before movement, during reaching, and when the subject held her hand at the target after movement. Morever, 70% of the directionally tuned cells showed a change in their directional relationship to movement. These cells either altered their range of discharge during movement in their preferred direction, or had a different preferred direction altogether. This suggests that MI encodes movement in an intrinsic frame of reference.

Hocherman and Wise (1991) trained subjects to perform either straight, clockwise curved, or counter clockwise curved reaching movements towards a target. Hocherman and Wise (1991) classified the cells based on the firing rate during the different types of movements and found that the vast majority of cells fired maximally during the curved movements. Ajemian *et al.* (2000) performed a simulation based on the experiment of Hocherman and Wise (1991) to examine the implications of assuming each of three different direction encodings – joint angle, Cartesian, or extrinsic polar. Ajemian *et al.* (2000) distinguished *internal pd* and *external pd*. An external pd is the preferred direction of a cell in terms of an extrinsic, Cartesian coordinate frame. An internal pd is the preferred direction of a cell in terms of the putative coordinate frame in which the cell encodes movement. This could be Cartesian, joint angle, extrinsic polar, or any other frame of reference. Ajemian *et al.* (2000) transformed the external pds of the cells into Cartesian, extrinsic polar, and joint angle internal pds. Using each of these internal pds, they built three different vector fields, each over the entire workspace. The direction of the vectors in each field corresponded to the extrinsic direction that would result from a particular intrinsic pd. Of course, this transformation was identity for a Cartesian internal pd; however, for a joint angle or extrinsic polar internal pd, the directions of vectors varied in the corresponding Cartesian-based field as the hand moved from its initial position in the workspace. Applying these three vector fields to the path of the monkey's hand, Ajemian *et al.* (2000) examined whether a cell would fire maximally during a clockwise, counterclockwise, or straight movement. They found that only a joint

angle internal pd could make similar predictions to Hocherman and Wise (1991). They concluded that, of the three candidates, the most likely frame of reference for the cells was joint angle.

2.1 Solving Linear Models with the Moore-Penrose Pseudo-Inverse

The general "learning" problem is to make a prediction employing some function learned from the training data. One can employ a variety of approaches to solve this problem. One of the simplest models is a linear combination of input:

$$\hat{y_j} = w_{0j} + \sum_{i=1}^{N} w_{ij} x_i, \tag{2.1}$$

where $\mathbf{w}$ is a set of parameters indexed by input (i) and predicted quantity (j), $\mathbf{x}$ is an N-dimensional input, and $\hat{y_j}$ is the predicted quantity. Without loss of generality, the bias term can be incorporated into the sum:

$$\hat{y_j} = \sum_{i=0}^{N} w_{ij} x_i, \tag{2.2}$$

where $x_0 = 1$.

Given a *training set* of P tuples ($< y_p, x_p >$) and J predicted quantities, we would like to find w so as to minimize the error function E:

$$\begin{aligned} E &= \frac{1}{2} \sum_{j=1}^{J} \sum_{p=1}^{P} (y_{jp} - \hat{y}_{jp})^2 \\ &= \frac{1}{2} \sum_{j=1}^{J} \sum_{p=1}^{P} \left(y_{jp} - \sum_{i=0}^{N} w_{ji} x_{ip} \right)^2, \end{aligned} \tag{2.3}$$

where y_{jp} is the actual value and $\hat{y}_{jp}$ is the predicted value.

We want to minimize the error E with respect to the parameter matrix W. The extreme value of E must satisfy $\frac{\partial E}{\partial w_{JI}} = 0$. The derivative of E with respect to w_{JI} is:

$$\frac{\partial E}{\partial w_{JI}} = \sum_{p=1}^{P} \left[\left(y_{Jp} - \sum_{i=0}^{N} w_{Ji} x_{ip} \right) (-x_{Ip}) \right] \tag{2.4}$$
$$= -\sum_{p=1}^{P} y_{Jp} x_{Ip} + \sum_{i=0}^{N} \sum_{p=1}^{P} w_{Ji} x_{ip} x_{Ip}.$$

Setting the derivative equal to zero and rearranging gives:

$$\sum_{p=1}^{P} y_{Jp} x_{Ip} = \sum_{i=0}^{N} \sum_{p=1}^{P} w_{Ji} x_{ip} x_{Ip} \tag{2.5}$$

Combining the results for each w_{JI} and expressing all P equations of 2.5 in a vector-matrix form gives:

$$\mathbf{Y}\mathbf{X}^T = \mathbf{W}\mathbf{X}\mathbf{X}^T, \tag{2.6}$$

where $\mathbf{Y}$ is a $J \times P$ matrix, and $\mathbf{X}$ is an $I \times P$ matrix. Finally we solve for $\mathbf{W}$:

$$\mathbf{W} = \mathbf{Y}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}, \tag{2.7}$$

where $\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}$ is the Moore-Penrose Pseudo-Inverse of $\mathbf{X}$ (Penrose, R., 1955; Serruya *et al.*, 2003).

2.2 Least Squares Kernel Regression

Linear models can perform well under certain conditions, but ultimately one wants to express non-linear transformations. At the same time, linear models have no local minima and a minimum can often be unique. We would prefer that our non-linear formulation also preserve this property. Additionally, the non-linear approach should still perform robustly with respect to overfitting and computational complexity. One such formulation, the kernel-based approach, involves generating a large set of non-linear features over the original feature set and then defining a tunable linear function over the features (Schölkopf and Smola, 2002). Specifically, a nonlinear model can be expressed as follows:

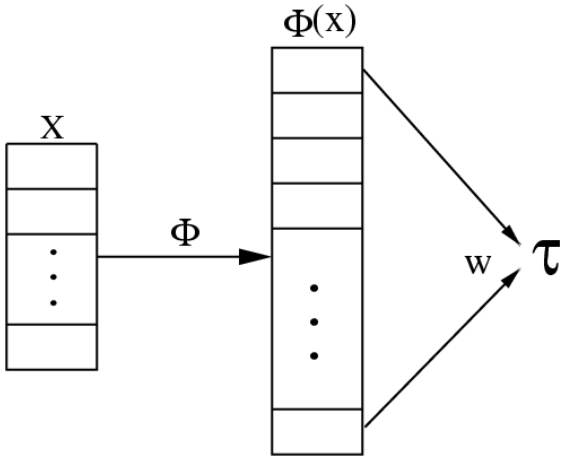


Figure 2.3: Non-Linear Map. A non-linear map, $\phi(\mathbf{x})$, transforms the input vector $\mathbf{X}$ into a larger, more information-rich "feature space." Then, a linear model $\mathbf{w}$ is built on top of this new "feature space" to predict torque or other arm motion descriptor.

$$\hat{\mathbf{y}} = \mathbf{w}^T \phi(\mathbf{x}) + b. \tag{2.8}$$

The mapping function, ϕ, generates the set of non-linear features, while the rest of the equation is the standard linear function given in Equation 2.1 (although the size of the vector $\mathbf{w}$ has changed). Figure 2.3 illustrates the linear function on the mapped input.

There are many possible mapping functions. One such family of functions is the *polynomial map* (Schölkopf and Smola, 2002). Here, each feature of $\phi(\mathbf{x})$ is a product of two or more features in $\mathbf{x}$. If M is the number of features in $\mathbf{x}$, then the number of all possible d-wise combinations of these features is M^d. When $d = 1$, we refer to the kernel as being linear; d=2 is a *quadratic* kernel function.

In order to solve for our weight vector $\mathbf{w}$, we bring the problem into a constrained optimization form by introducing a set of slack variables (Suykens *et al.*, 2000). In particular, we minimize:

$$E(\mathbf{w}, \mathbf{e}, b) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{\gamma}{2}\sum_{i=1}^{N} e_i^2, \tag{2.9}$$

subject to:

$$y_i = \mathbf{w}^T \phi(\mathbf{x}_i) + b + e_i \tag{2.10}$$

where $\mathbf{w}$ is the regression parameter vector, $\mathbf{e}$ is the prediction error, and b is the offset. Also, γ is a regularization constant that specifies the relative importance of minimizing error versus retaining small values in the parameter vector. Unless the training data encodes otherwise, the individual elements of $\mathbf{w}$ are chosen to be very small. This combats overfitting to small variations in the training data.

The above constrained optimization problem can be re-cast into an unconstrained problem through the introduction of a set of Lagrangian multipliers. Through this approach, in addition to including the original cost function (equation 2.9), we encode the equality constraints from equation 2.10 within the third set of terms. This gives the Lagrangian:

$$L(\mathbf{w}, \mathbf{e}, \alpha, b) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{\gamma}{2}\sum_{i=1}^{N} e_i^2 - \sum_{i=1}^{N} \alpha_i \left(\mathbf{w}^T \phi(\mathbf{x}_i) + b + e_i - y_i\right). \tag{2.11}$$

Note that y_i is a scalar (without loss of generality). The derivative of L with respect to the free variables is as follows:

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x}_i), \tag{2.12}$$

$$\frac{\partial L}{\partial e_i} = \gamma e_i - \alpha_i, \tag{2.13}$$

$$\frac{\partial L}{\partial \alpha_i} = -\mathbf{w}^T \phi(\mathbf{x}_i) - b - e_i + y_i, \text{ and} \tag{2.14}$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{N} \alpha_i. \tag{2.15}$$

At the extreme point of L, each of these derivatives must be zero. Setting equation 2.14 equal to 0 recovers each of the equality constraints of equation 2.10. When we hold the α_i's constant at this extremum, E (equation 2.9) and L (equation

2.11) are the same (given any choice of $\mathbf{w}, \mathbf{e}$, or $\mathbf{b}$). This means that under this choice of the α's the extreme point of L corresponds to the extreme point of E.

To solve this unconstrained problem, we set equations 2.12-2.15 equal to zero. Setting 2.12 to zero yields:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x_i}), \tag{2.16}$$

and setting 2.13 to zero results in:

$$e_i = \frac{\alpha_i}{\gamma}. \tag{2.17}$$

Likewise, setting 2.14 to zero and substituting 2.16 and 2.17 yields:

$$y_i = \sum_{j=1}^{N} \alpha_j \phi(\mathbf{x_j})^T \phi(\mathbf{x_i}) + b + \frac{\alpha_i}{\gamma}. \tag{2.18}$$

This gives us a set of N equations and N+1 unknowns (α's and b). The remaining equation comes from setting 2.15 to zero:

$$0 = \sum_{i=1}^{N} \alpha_i. \tag{2.19}$$

This set of equations can be expressed as a vector-matrix equation:

$$\begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & & & \\ \vdots & & \left[\ \mathbf{K} + \frac{1}{\gamma}\mathbf{I}\ \right] & \\ 1 & & & \end{bmatrix} \begin{bmatrix} b \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix} = \begin{bmatrix} 0 \\ y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \tag{2.20}$$

where:

$$K_{ij} = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j), \tag{2.21}$$

i and j range from 1 to N, and $\mathbf{I}$ is the identity matrix. We can solve for the unknowns by inverting the $(N+1) \times (N+1)$ matrix.

Furthermore, equation 2.12 means that we can express equation 2.8 in terms of a sum over the training set:

$$\hat{y_i} = \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b. \tag{2.22}$$

This implies that the function can be represented as a weighted sum of inner products between the query point and the training set elements. However, as stated, such a query requires $N \times M^d$ multiplications. We will show in the next section that this computation can be reduced to $N \times M$ multiplications.

2.3 Kernel Reduction

In the general case, a kernel, k, maps from an M-dimensional input space to the inner product of two vectors in the expanded feature space:

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}').$$

For a polynomial kernel, the feature space ϕ forms the set of all possible products of an input dimension with $d-1$ other input dimensions. The calculation of the kernel could easily become computationally intractable. Fortunately, the polynomial kernel is also equivalent to an inner product over the space of size M. An element of the feature space, $\phi_j(\mathbf{x})$, is the following combination of d elements of $\mathbf{x}$:

$$\begin{aligned} \phi_j(\mathbf{x}) &= x_{j \bmod M} \times x_{(j/M) \bmod M} \times x_{(j/M^2) \bmod M} \times ... \times x_{(j/M^{d-1}) \bmod M} \\ &= \Pi_{k=0}^{d-1} x_{(j/M^k) \bmod M}. \end{aligned} \tag{2.23}$$

Given the definition of the mapping function, consider the inner product of two mapped vectors:

$$\begin{aligned}
\phi(\mathbf{x})^T\phi(\mathbf{x}') &= \sum_{j=0}^{M^d-1} \phi_j(\mathbf{x})\phi_j(\mathbf{x}') \\
&= \sum_{j=0}^{M^d-1} (\Pi_{k=0}^{d-1} x_{(j/M^k) \bmod M})(\Pi_{k=0}^{d-1} x'_{(j/M^k) \bmod M}) \\
&= \sum_{j=0}^{M^d-1} \Pi_{k=0}^{d-1} x_{(j/M^k) \bmod M} \times x'_{(j/M^k) \bmod M} \\
&= \sum_{j=0}^{M^d-1} (x_{j \bmod M} \times x'_{j \bmod M} \times x_{(j/M) \bmod M} \times x'_{(j/M) \bmod M} \times ... \\
&\qquad \times x_{(j/(M^{d-1})) \bmod M} \times x'_{(j/(M^{d-1})) \bmod M}) \\
&= (x_0x'_0 + x_1x'_1 + x_2x'_2 + ... + x_{M-1}x'_{M-1})^d \\
&= \left(\sum_{i=0}^{M-1} x_ix'_i\right)^d \\
&= (\mathbf{x}^T\mathbf{x}')^d
\end{aligned} \tag{2.24}$$

This implies that the computation of K_{ij} does not require M^d products, but instead only requires M. However, in order to solve 2.21, we still need to perform $O(N^2)$ of these inner products. Through the use of the kernel reduction, we significantly reduce the computation time of this non-linear approach, while retaining the property of no local minima.

Chapter 3

Brain Machine Interface Problem

To create a Brain Machine Interface that improves current prosthetic technology, we are developing techniques that will allow the use of cortical recordings to directly command the movement of a prosthetic or paralyzed arm. We use simultaneously recorded cortical activity and arm movement from monkeys performing a random target pursuit task. First we formulate this problem in terms of prediction: given the recent cell firing behavior, we would like to predict the subsequent motion of the monkey's arm. If we can make accurate predictions, then it becomes possible to artificially drive the monkey's arm (or a robotic arm) given the predicted motion.

The focus of our study is on a monkey performing a random-walk, two-dimensional (planar) reaching task (Hatsopoulos, 2005). The monkey's arm is placed in a KINARM (Scott, 1999), an exoskeleton robot arm that records the arm position at the same time that a sample of 50-135 motor cortical cells is recorded. A target appears on the screen at random places in his reachable workspace, and the monkey moves her arm to the target in a specified amount of time. Once the monkey's hand arrives at the target, the target will appear at another random point in the plane. The monkey performs approximately 200 trials through the course of an experiment. For each trial, the monkey makes 6-7 pointing movements, with an average length of 9 inches.

Samples of arm position are taken every 2 msec. The differentiation required to estimate angular velocity and acceleration amplifies high frequency noise. Therefore, the arm position signals are filtered using a 6-pole Butterworth filter with a 6 Hz cutoff frequency. Velocity and acceleration are then estimated using differencing.

Next, we use an inverse dynamics model of the kinarm and monkey's arm to estimate joint torques from observed arm motion (Fagg *et al.*, Submitted).

In the pre-processing step, we discard cells that fall below a .5 Hz average firing rate over the entire data set. Then, we bin the neural spike data into 50 msec intervals and count the number of spikes that occur in each interval. The number of spikes in a single bin represents a single feature.

The process of transforming a representation of recent cell activity into a prediction of arm motion is illustrated in Figure 3.1. Each of the horizontal lines of spikes represents a single neuron, while the dots depict the action potential of the corresponding neuron with respect to time. In order to make a prediction of arm motion at time t, a decoder is given access to the recent cell activity as described by the spike counts of the 20 previous bins, for a total of one second of history. This information is represented using a vector of length $20 \times$ N, where N is the number of cells. The lower half of the graph represents the observed arm state (in this case shoulder) described in terms of the joint angle, velocity, acceleration, and torque, shown from top to bottom.

The lower part of the figure represents the transformation of the neural data into a prediction of arm motion. $\mathbf{X}$ represents the larger vector of spike counts that we use as input into a model, while the arm motion, $f(x)$, is the prediction. Throughout the thesis, we use several different models, ranging from linear to non-linear. Finally, we compare the observed and the predicted arm motion in order to measure the performance of the model.

3.1 Describing Arm Motion

We approach the question of whether neural spike rates encode motion in an extrinsic or intrinsic coordinate frame by constructing predictors of each and comparing their relative performance. Given the recent evidence for intrinsic representation of movement in MI and related areas, we hypothesize that predictors of intrinsic (and, in particular, dynamic) representations of movement should outperform predictors of extrinsic representations of movement.

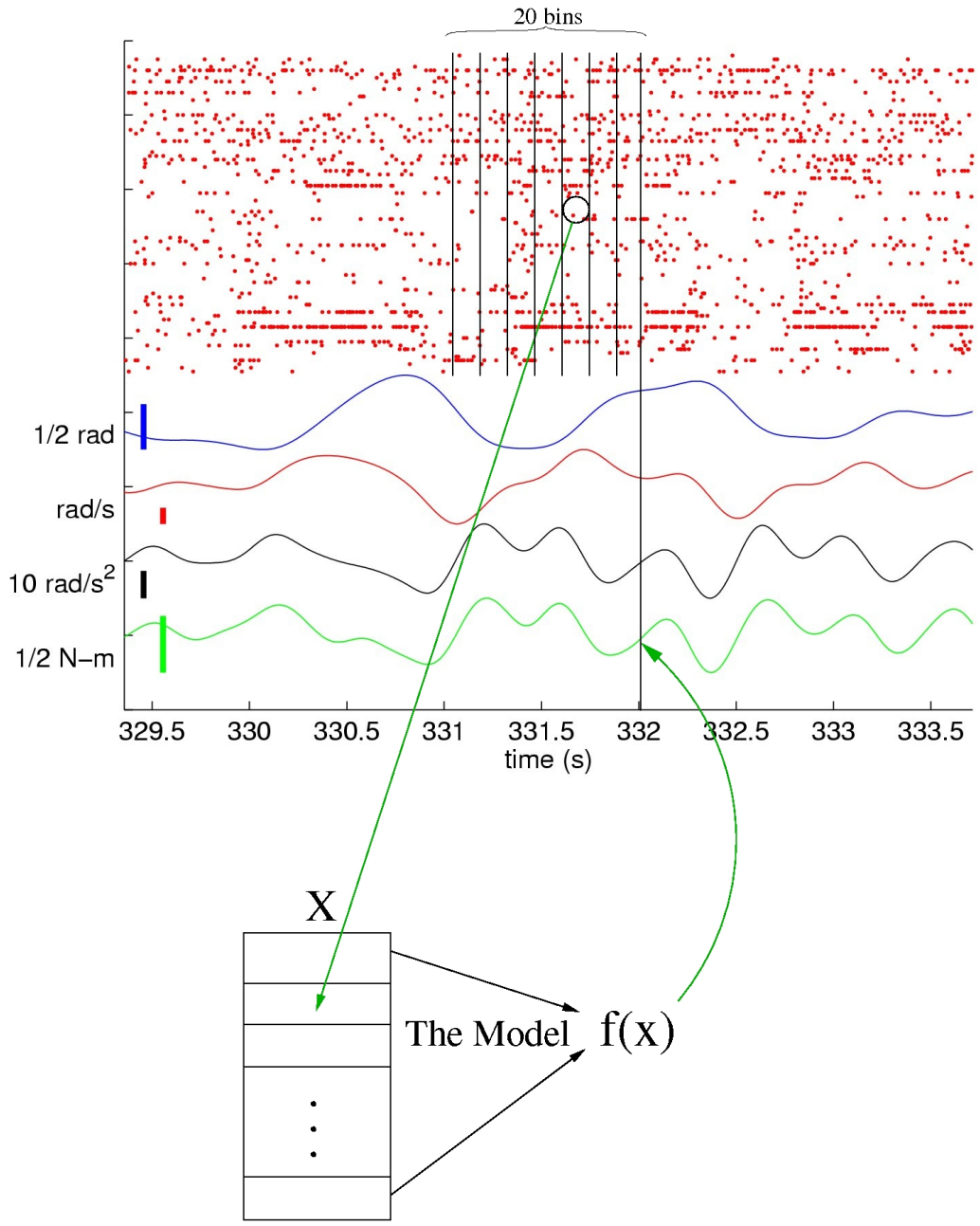


Figure 3.1: Prediction Method. Each row of dots represents a different neuron's spikes with respect to time. The color lines represent the simultaneously recorded joint angle, joint velocity, joint acceleration, and torque, from top to bottom. For a prediction at time t, a vector is formed from each of the cell's spike counts over the previous second, separated into 20 equal bins. For example, if there are 50 neurons used (a typical case), this requires a vector of length 1000 (shown in the bottom of the figure). Next, a model transforms the input vector into a prediction.

3.2 Measuring Prediction Accuracy

There are two important considerations when selecting a performance measure for a Brain Machine Interface (BMI). First, since intrinsic and extrinsic coordinates are expressed in different units, it is difficult to compare the performance of the two models directly. Therefore, one needs a unitless metric. Second, the subject will want the prediction to match her desired movement, not simply be linearly correlated with it. Therefore, the performance measure should reflect this requirement as well.

The BMI community traditionally uses the Coefficient of Determination (e.g., Reina *et al.*, 2001; Carmena *et al.*, 2003; Shoham *et al.*, 2005; Wu *et al.*, 2004):

$$R^2 = 1 - \frac{\sum_{i=1}^{N} (y_i - a\hat{y}_i - b)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2}, \tag{3.1}$$

where y_i is the observed value at time step i, $\hat{y}_i$ is the predicted value, $\bar{y}$ is the mean of the observed values, and a, b are constants that are chosen to maximize R^2. The Coefficient of Determination is the proportion of linear correlation between the predicted and actual value. Note that R^2 varies from 0, when the predicted value is not correlated with the actual value, to 1, when the predicted value is perfectly correlated to the actual value. Since both the numerator and the denominator contain the measured unit, R^2 is a unitless metric. However, the transformation from the predicted to actual value (which is determined by parameters a and b) means that the R^2 measure will not penalize a prediction value that is different than the observed value, yet linearly correlated with it. For example, if $\hat{y} = 2y+3$, R^2 will be unity. However, we are concerned with the desired path of the subject's hand, not just a path that is correlated with it. Consequently, this measure is less useful when measuring BMI performance. The other objection to the R^2 metric is that one selects a and b using the data set whose performance is being measured. This means that even when we are using a "test" data set, we are actually using it as part of the training process (which violates the fact that we are calling it a test set).

We follow Serruya *et al.* (2003) in using a modification of R^2, which we term the Fraction of Variance Accounted For (FVAF). Here, we require that $a = 1$ and $b = 0$:

$$FVAF = 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \bar{y})^2}. \tag{3.2}$$

This stricter measure accounts for deviations between the desired motion and the predicted arm motion, as well as cancelling the measured units. FVAF is interpreted in the following way:

$FVAF = 1$	The model is a perfect one.
$0 < FVAF < 1$	The model predicts some of the variance that is present in the predicted quantity.
$FVAF < 0$	The model injects more variance than exists in the predicted quantity. In practice, this case happens when the model is lacking some critical data and is forced to extrapolate inappropriately.

Note that given a linear model of $\hat{y}$ and a MSE metric, R^2 = FVAF for the training set (the optimal choices for a and b are 1 and 0, respectively). We will only see a difference with an independently drawn test or validation data set.

In the context of our experiment, we regard a model with an FVAF of 1 as reconstructing arm movement exactly as the monkey would expect. For example, suppose we can construct a decoder for which FVAF=1. Now, assume that we can replace the monkey's muscle-generated movement with the decoder-driven movement of the robotic exoskeleton. Since this is a perfect predictor, the arm behaves as the monkey would expect. Thus, the monkey will not require any training to perform reaching tasks and performance on those tasks should be the same as before the switch occurred. We assume that with a less than perfect predictor (for which FVAF < 1), the movement of the robotic arm will deviate from what is expected by the monkey. When we use this imperfect decoder, the difference between the generated and expected motion will require the monkey to correct the motion of the robot.

3.3 Model Evaluation

In reporting results, we use a standard procedure for evaluating models, N-fold cross-validation (Browne, 2000). For the first step, we divide the data into N independent folds, with an equal number of trials. Then, we build N different models. For the first model, we use folds 1, 2, ..., N-2 to train the model and the Nth fold as an independent test set. Some models require validation - testing on a sample data set in order to decide when to cease training or to select the appropriate parameter settings. We reserve the $N-1^{th}$ fold for use as this validation data set. For the second model, we use folds {1, 2, ..., N-3, N} to train the model and evaluate the resulting model using fold N-1, with N-2 for validation. This procedure results in N different test set performance measures. We report the mean and standard deviation of these measures (Browne, 2000). For our data sets, we use N=20. Using 20 folds allows enough different models to test, while allowing an adequate size for each training set and test set.

This method of reporting is important for robust, reliable results. If, for example, we only reported the results of the test set on the first model, it would be difficult to measure the performance of the algorithm. On the one hand, it could be that, by chance, all the cases of the chosen test set are well-encapsulated in training data. This would result in an inflated accuracy measure. On the other hand, it could be that, for the same data, the training data does a poor job of covering the cases in the chosen test set, resulting in an unfairly low accuracy measure. By averaging the results of all twenty cases, however, one can report an accurate performance metric.

Chapter 4

Evaluating Population Vectors for Predicting Arm Motion from Cortical Activity

4.1 Introduction

Well established in the literature, the *Population Vector* approach is commonly used today to make predictions of arm motion (Georgopoulos *et al.*, 1986; Reina *et al.*, 2001; Schwartz *et al.*, 1988). In this method, one first creates a linear model of discharge rate of a particular neuron as a function of movement direction. Georgopoulos *et al.* (1982) found that the rate will peak at a particular Cartesian direction of arm movement, which they called the *preferred direction*. Given these models for a population of neurons, one can then estimate the current movement direction of the arm based on the firing pattern of the population. In this chapter, we compare the prediction accuracy of Population Vectors to the Pseudo Inverse solution to the linear model, whose solution minimizes training set sum squared prediction error. We also compare the Population Vector approach to linear models involving multiple time delays.

4.2 Population Vector Background

In their Population Vector experiments, Reina *et al.* (2001) used data from monkeys performing controlled, center-out tasks, each in one of eight discrete directions.

Reina *et al.* (2001)'s formulation for the model of the discharge rate of a particular neuron i, adapted for 2D, is:

$$\bar{D}_i^* = b_{i,0} + b_{i,x}\bar{\dot{x}} + b_{i,y}\bar{\dot{y}}, \tag{4.1}$$

where $\bar{D}^*$ is the neural spike rate, b_o, b_x, and b_y are linear coefficients, $\bar{\dot{x}}$ is lateral hand velocity, $\bar{\dot{y}}$ is anterior/posterior hand velocity, i corresponds to neuron i, and the bar over the variables indicates an average over the reaction and movement periods. They derived the linear coefficients by performing a regression over the observed spike rates and hand velocities. The discharge rate is maximized when the movement direction is the same as $b_{i,x}/b_{i,y}$.

Our reconstructions are based on experiments which require the monkey to move through a sequence of targets randomly selected from a uniform distribution over the workspace (Hatsopoulos, 2005; Fagg *et al.*, Submitted). Rather than addressing the question of when individual movements begin and end, and assuming that these movements are straight, we instead choose to model discharge rate for a given 50 ms bin as a function of movement velocity. In order to account for signal propagation delays, we assume that discharge occurs Δt prior to the observed velocity (this parameter will be selected below).

Equation 4.1 tells us how to derive/predict cell activity given motion. Given activity of a *set* of cells, now we want to estimate motion. Reina *et al.* (2001) use the method of Population Vectors. A Population Vector is a weighted mean of preferred directions, where the weights are determined by the current discharge rate of each cell. In equation form, this is:

$$PV_{j,t} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{D_{i,t}^* - \bar{D}_i^*}{D_{i_{max}}^* - \bar{D}_i^*} \cdot \frac{b_{i,j}}{\sqrt{\sum_{j=1}^{2} b_{i,j}^2}}\right), \tag{4.2}$$

where $PV_{j,t}$ is a prediction for Cartesian coordinate j at time t, i represents neuron i of N, $D_{i,t}^*$ is the spike rate at time t, $\bar{D}_i^*$ and $D_{i_{max}}^*$ are the mean and maximum spike rate, respectively, during the entire experiment, and $b_{i,j}$ is the directional weighting of the neuron on Cartesian coordinate j. The left-hand side of the product in Equation 4.2 represents how rapidly an individual cell fires compared to its peak firing rate.

The right-hand side of the product is the representation of the preferred direction of the cell. Thus, the overall equation represents the sum of each neuron's preferred direction, with each direction scaled by its relative firing rate. This sum is taken as the prediction of movement for time t. It is important to note that the calculation of Population Vectors in Equation 4.2 is linear in the observed spike rates.

In order to maximize the prediction accuracy of Population Vectors, we found that two modifications were necessary. First, the maximum statistic is very sensitive to spurious measures of spike rates. The use of the 99^{th} percentile, in practice, gave more stable and repeatable results. Finally, the depth of modulation of each neuron may vary significantly, yet Population Vectors treat the contribution from each neuron equally. In other words, a neuron which is only slightly directionally tuned will have the same amount of vote for the reconstructed direction as a neuron which is much more directionally tuned. To reduce this interference and improve reconstruction performance, we eliminated the 10% of cells with the lowest task-related activity.

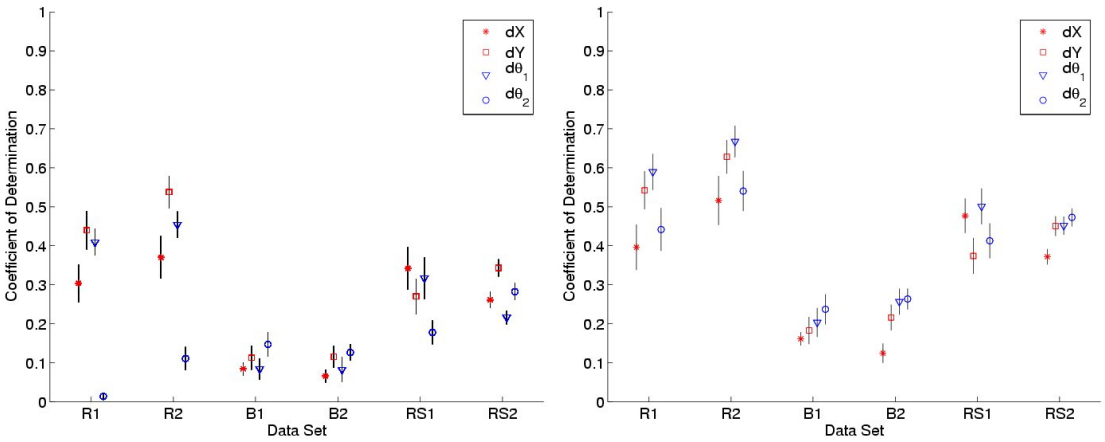
Finally, we compare the exstrinsic coordinate frame predictions (Cartesian hand velocity) with intrinsic predictions. For intrinsic predictions we construct a neuron discharge model as a function of joint position, as Reina *et al.* (2001) write:

$$\bar{D}_i^* = b_{i,0} + b_{i,1}\bar{\dot{\theta}}_1 + b_{i,2}\bar{\dot{\theta}}_2, \tag{4.3}$$

where θ_1 is internal/external rotation of the shoulder, and θ_2 is elbow flexion/extension. The rest of the variables and symbols, as well as modifications for our experiments, are the same as in Equation 4.1. Reina *et al.* (2001) also include additional joint angle information in this model, such as shoulder adduction/abduction and flexion/extension, radial pronation/supination, wrist flexion/extension and abduction/adduction.

4.3 Results

For our experiments, we chose the Δt parameter, the delay between $\bar{D}^*$ and velocity in Equation 4.1 and 4.3, for the Population Vector model by selecting the value from a range of time delays that were likely to include information encoding movement



(a) Population Vectors

(b) Pseudo Inverse

Figure 4.1: For six independent data sets, the mean of the test set R^2 performance is shown. For each data set, Cartesian X velocity is on the left, followed by Cartesian Y, shoulder, and elbow velocity.

at time t. Since Figure 7 in Georgopoulos *et al.* (1982) shows that the majority of increased neuronal discharge occurs up to 200 ms before the onset of movement, we built models with Δt's of 50, 70, 100, 150, and 200 ms. We selected 150 ms, since the corresponding model had the highest prediction accuracy on the validation set.

Figure 4.1(a) displays the mean and standard deviation of the test set R^2 performance of Population Vectors for six independent data sets. The figure shows R^2 for Cartesian X and Y hand velocity, shoulder, and elbow velocity. The B data sets have the fewest number of cells.

Now, we compare these results to the approach of using Pseudo Inverse to derive the coefficients of the linear model from discharge rate of the set of cells to the predicted quantities. We implement Pseudo Inverse using one feature per time bin for each cell. In the Pseudo Inverse reconstruction in Figure 4.2, the observed hand X-coordinate velocity is shown in red, while the prediction is in blue. Note that the prediction accuracy for Population Vector, as measured in terms of R^2, is not as good as the reconstruction using Pseudo Inverse.

Although the method of Population Vectors is intuitive, it heuristically assumes that each cell contributes equally to the estimate of movement velocity. In particular, Population Vectors do not take into account the fact that some cells might not have

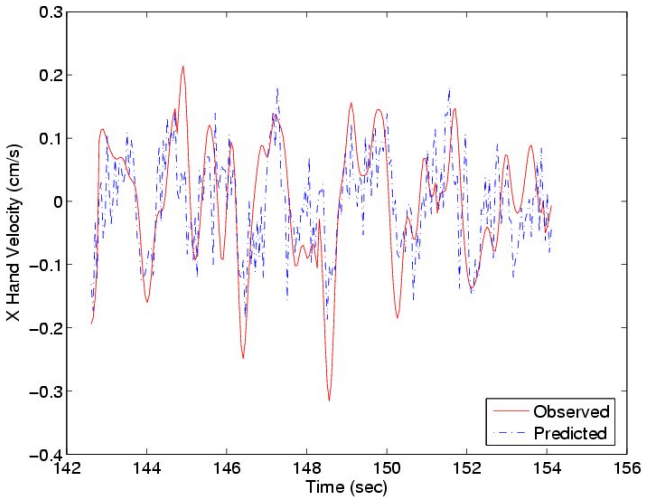
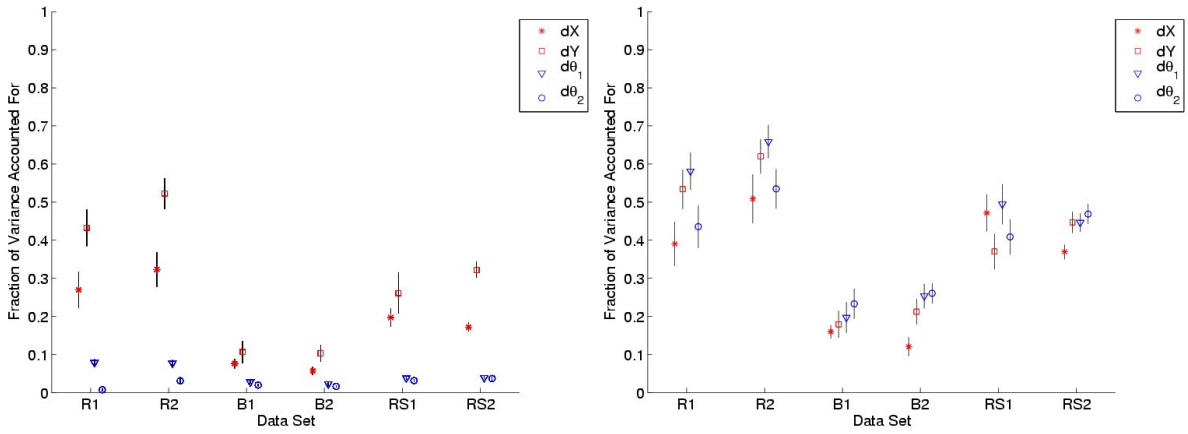


Figure 4.2: One Bin Pseudo-Inverse X Hand Velocity Reconstruction for a representative test set and trial on data set R1.

significant information content with respect to velocity, or that some cells may be redundant in their coding of velocity. Figure 4.1(b) shows Pseudo Inverse results for all six data sets. We used the same input data as Population Vectors (one, 50 ms bin of spike count centered at a time delay d before movement). Both Population Vectors and Pseudo Inverse are linear methods and can represent the same class of functions; however, Population Vector parameters are chosen using heuristics, while the parameters of Pseudo Inverse minimize the mean squared error of the training set. Clearly, Pseudo Inverse approach is superior to Population Vectors, as measured by R^2 (difference in means of .254 R^2, p $< 10^{-78}$ from a paired t-test across all data sets and motion descriptor prediction performances).

R^2 measures the correlation between observed and predicted variables. However, it does not capture the absolute accuracy of prediction. Figure 4.3(a) examines the performance of the Population Vector approach using the Fraction of Variance Accounted For (FVAF). Using FVAF, one sees a much lower performance for joint angle velocity than for Cartesian velocity. This large disparity in performance did not exist when using R^2 for model evaluation. This disparity means that population vectors are making a prediction for joint angle velocity that is somewhat correlated with the desired path, but is not nearly as close to the observed values as for Cartesian velocity. Thus, Population Vectors seem to be much less useful for joint angle



(a) Population Vectors

(b) Pseudo Inverse

Figure 4.3: For six independent data sets, the mean of the test set FVAF performance is shown. For each data set, Cartesian X velocity is on the left, followed by Cartesian Y, shoulder, and elbow velocity.

velocity predictions than for Cartesian predictions. More importantly, this highlights the difference between using R^2 and the stricter performance measure, FVAF, in evaluating the prediction performance of models.

Notably, measuring the reconstruction performance of Pseudo Inverse by FVAF and R^2 gives virtually equivalent prediction accuracy (Figures 4.1(b) and 4.3(b)). This is because Pseudo Inverse already minimizes the mean squared error of the training set, and the predictions on the test set are performed with comparable accuracy. As a result, the linear regression performed by the R^2 computation from predicted to actual velocity results in essentially a null transformation.

Note that with the Population Vector approach to a random-walk task, only one time interval of MI activity can be used to calculate the preferred direction. In our case, we chose to look at the spike rate in 50 ms bins. Reina *et al.* (2001) averaged the spike rate over the entire arm movement (about 300-500 ms). This was possible because the task was structured such that they could reasonably expect consistently straight-line movements (or at least this is what they assumed). Given that the models only have access to 50 neurons, they have a very limited view of the activity of the motor cortex as a whole. One way to improve on this is to allow the model access to cell activity over a range of time delays. This type of model is referred to

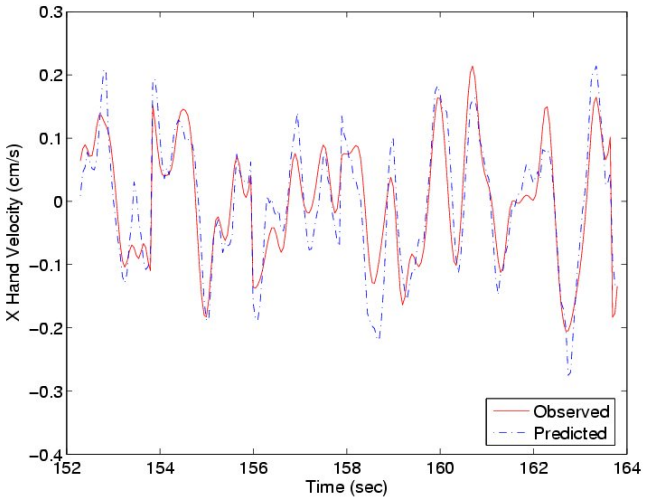


Figure 4.4: 20 Bin Wiener filter reconstruction of X hand velocity for a representative test set and trial on data set R1.

as a *Wiener filter* (Serruya *et al.*, 2003; Warland *et al.*, 1997). In the experiments whose results are shown below, we use the Pseudo Inverse approach to compute the parameters of the Wiener filter. Specifically, we include the spike rate measures from delays of 0 ms to 1 s, in 50 ms bins, for a total of 20 separately weighted time delays.

Figure 4.4 shows a sample of observed versus predicted hand X-coordinate velocity using this approach. Note the qualitative improvement in prediction accuracy compared to Figure 4.2, which has only one bin per prediction. Quantitatively, we compare Figure 4.3(b) to Figure 4.5 and see that accounting for multiple time delays allows for an enormous improvement in prediction performance (mean difference is .31 FVAF, $p < 10^{-20}$ for a paired t-test across all data sets and motion descriptor prediction performances).

4.4 Discussion

We solve for a linear model using two different methods. One method, Population Vectors, explicitly computes a representation of preferred direction for each cell (which allows for analysis of what individual cells might be contributing). The other method, Pseudo Inverse, selects an optimal set of parameters given a training set.

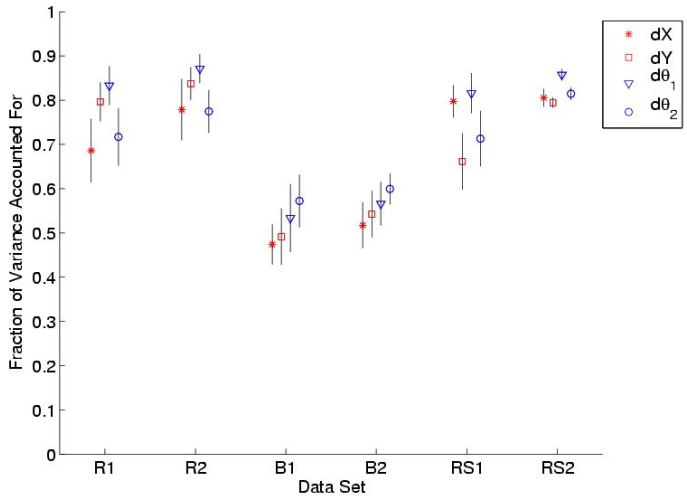


Figure 4.5: 20 Bin Pseudo-Inverse FVAF Summary. For six independent data sets, the mean of the test set performance, measured in FVAF, is shown for a prediction using Pseudo-Inverse.

Across six independent data sets, we find that Pseudo Inverse far outperforms Population Vectors in every case, using both FVAF (see Measuring Prediction Accuracy) and the coefficient of determination used in Reina *et al.* (2001). This stands to reason, since Population Vectors are a linear model with parameters selected by the "preferred direction," instead of optimized to reduce error on the training set. Therefore, this experiment provides a benchmark measuring the effectiveness of Pseudo Inverse compared to a wide-spread prediction method, Population Vectors. In addition, we find that the stricter peformance metric, FVAF, yields different results than R^2. We prefer the stricter measure since we would like to use these models to ultimately control a prosthetic arm. Using the stricter method, we find that the Pseudo Inverse solution to the model dramatically outperforms the population vector solution. Finally, we find that arm motion predictions using the Wiener filter approach are substantially more accurate than predictions using one time delay.

Chapter 5

A Kernel-Based Approach to Predicting Arm Motion from Cortical Activity

5.1 Background

The transformation from activation patterns to arm motion is inherently nonlinear since the dynamics and kinematics are non-linear functions of joint state. Thus, predicting arm motion from cortical activity warrants the use of a non-linear model. Since we can only record neural activity and arm movement from a monkey for short periods of time, we also need an approach that excels in predictions on sparse data sets. Least Squares Kernel Regression (LSKR) enables the introduction of nonlinearities in the represented function while providing a regularization method that addresses the problem of overfitting that can occur in sparse data sets.

LSKR requires a regularization parameter, a mapping function, and a kernel function. The regularization parameter, γ, is a constant in the error function that specifies the trade-off between function complexity and the degree of fit to the training set. The mapping function transforms the input into a higher dimensional feature space, where the regression is performed. This enables an otherwise nonlinear function to be expressed as a linear function in the expanded feature space. Finally, the kernel function measures similarity between input patterns. Polynomial kernels capture the co-occurence of cell activity or specific temporal patterns. Yet, due to the kernel reduction for polynomial kernels, we do not have to perform a computationally expensive regression in the expanded feature space. For these reasons, we

assess Least Squares Kernel Regression (LSKR) with polynomial kernels for offline prediction.

5.2 Results

We used a modified version of LS-SVMlab1.5 (Suykens *et al.*, 2002). Since the amount of training data is directly correlated with the time commitment required from the subject (and the subject will likely have to record training data periodically for recalibration), a critical question is how much training data is required for a model to perform well. Therefore, we measured the performance with a varied amount of training data and varying γ to understand how they affect prediction accuracy.

To help ensure reliable result reporting, all of the graphs represent the 20 fold cross-validated mean and standard deviation of FVAF of reconstruction performance on an independent test set. In addition, all differences in the mean are statistically verified for significance. The p-values reported in the results correspond to a two-tailed, paired t-test. When multiple comparisons are made, we apply a Bonferroni correction to this test (Jensen and Cohen, 2000).

5.2.1 Linear Kernel

In this section, we apply LSKR with a linear kernel to the movement prediction problem. A linear kernel transforms the input space into a feature space that is exactly the same as the input space (c.f. Section 2.2). Therefore, one can compare these results of LSKR with a linear kernel to a pseudo-inverse solution. Note that the only difference between the two is that LSKR also minimizes the length of the weight vector.

Two key questions are how the learning approaches deal with varying amounts of training data and how to set the regularization parameter. First, we vary the amount of training data and measure prediction performance of an LSKR predictor of torque (shoulder and elbow) and Cartesian position. Figure 5.1 shows these results, with dots representing the mean and whiskers the standard deviation of the test set performance. Except for shoulder torque, increasing the size of the training

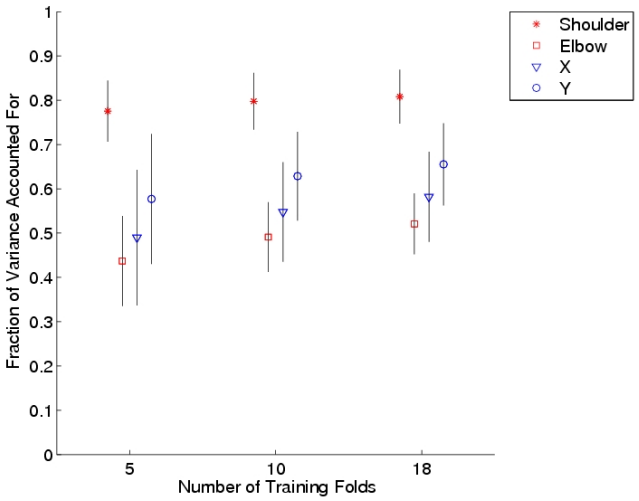


Figure 5.1: Prediction performance as a function of training set size. Results are for a linear kernel LSKR predictor of torque (red) and Cartesian (blue) on the R1 data set.

set improves performance on the R1 data set ($p < 0.02$ for all elbow-elbow, X-X, and Y-Y comparisons from 5 to 10 and 10 to 18).

Note that the shoulder prediction is signficantly more accurate than the elbow prediction. There are two possible explanations:

1) the shoulder is subject to higher inertial load, and therefore might be driven at a lower frequency than the elbow. Thus, it is easier to predict. Or,

2) the cells that are being recorded from may primarily encode shoulder movements. Since we see less of a difference in three out of the six data sets in Figure 5.8a, this explanation is less likely.

The regularization parameter determines the relative importance of explaining the training data and having a "flat" model. Larger γ's result in placing more weight on explaining the training data. In Figure 5.2, we show FVAF for torque and Cartesian predictors as γ varies on data set R1.

A properly selected γ can improve prediction accuracy. There is an average .16 FVAF increase in mean validation set performance comparing $\gamma = 0.00001$ and 10 for torque and Cartesian predictions ($p < 2 \times 10^{-8}$, paired t-test). Since the variation in performance is small amongst the rest of the gamma values, we choose a gamma of 10 for the remaining experiments.

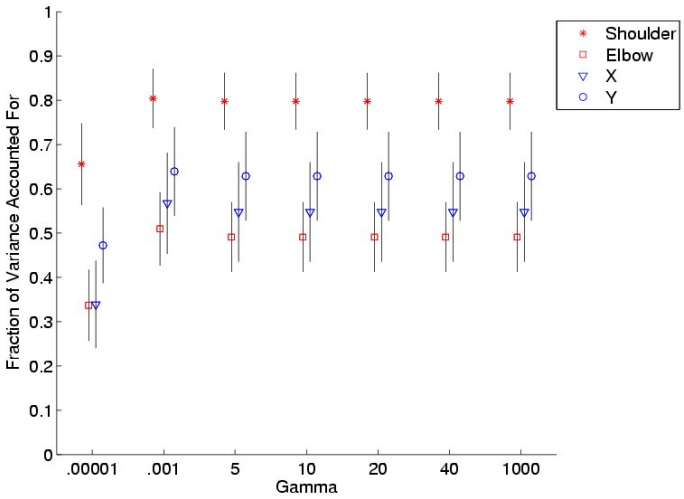


Figure 5.2: Prediction performance for LSKR as a function of γ, using 10 training folds and a linear kernel. Results are given for torque (red) and Cartesian (blue) on the R1 data set.

5.2.2 Quadratic Kernel

Polynomial kernels allow us to capture the co-occurence of cell activity or specific temporal patterns (for an explanation of the quadratic kernel see section 2.2).

Figure 5.3 shows the Fraction of Variance Accounted For for LSKR (quadratic kernel) on the R1 data set predicting torque and Cartesian position as number of training folds varies. There is a trend of increasing accuracy with an increasing number of training folds. Also notice that the shoulder torque performance improves less than elbow torque or Cartesian position as the size of the training set increases. This may be because of a ceiling effect. Figure 5.4 shows the effect of varying γ on prediction performance of R1 with 10 training folds. Changing γ makes a small but significant difference on prediction performance for both torque and Cartesian position (mean difference of 0.013 FVAF comparing $\gamma = 0.00001$ and 10, $p < 6.134 \times 10^{-4}$).

In the kernel reduction, the quadratic kernel is calculated as $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\mathbf{T}\mathbf{y}+t^2)^2$, where $\mathbf{x}$ and $\mathbf{y}$ are the two training instances and t is the offset. The intuition behind the offset is that it is identical to adding a t to the end of each vector. When $t = 0$, only terms of polynomial degree d are represented. When t is not 0, then terms corresponding to all polynomial degrees (from 0 to d) are represented in the feature

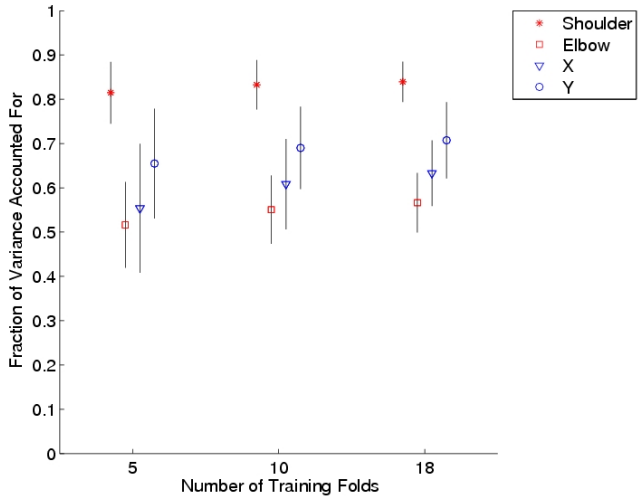


Figure 5.3: Measuring prediction performance as a function of training set size using FVAF. Results are shown for a quadratic kernel LSKR predictor of torque (red) and Cartesian (blue) on the R1 data set.

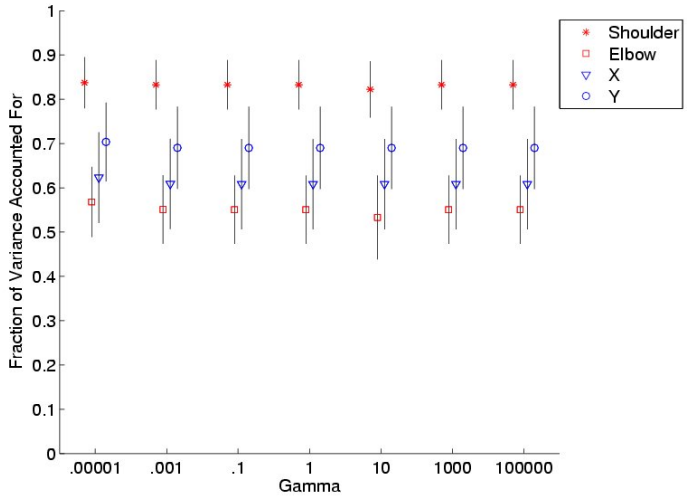


Figure 5.4: Prediction performance as a function of γ, the complexity trade-off, for LSKR with a quadratic kernel. Results are shown for torque and Cartesian position (red and blue, respectively) using 10 training folds of the R1 data set.

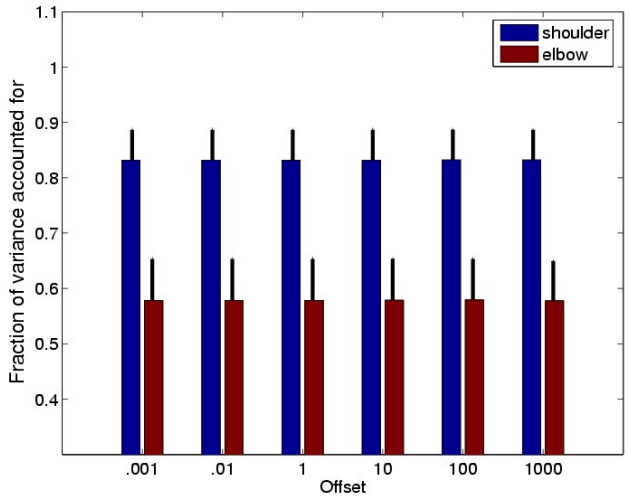


Figure 5.5: Prediction performance as a function of offset t for LSKR with a quadratic kernel. Results are given for torque (blue=shoulder, red=elbow) using 10 training folds on the R1 data set.

set. Moreover, features of various degrees will have different weights. Figure 5.5 shows LSKR with a quadratic kernel predicting torque on the R1 data set, as we vary the offset t. This shows that varying the offset of the polynomial kernel does not affect the performance for arm torque ($p < .95$ comparing offset = 0.001 and 1000). This result may be due to a change in the α's that compensates for this change in offset.

5.2.3 Higher Degree Kernels

Figure 5.6 shows the results of varying the degree of the polynomial kernel on LSKR predicting torque on the R1 data set. Neither third nor fourth degree polynomial kernels predict both shoulder and elbow torque significantly better than a quadratic kernel on data set R1 with 10 training folds and a γ of 10 ($p > 0.051$ for quadratic versus third degree, $p > 0.99$ for quadratic versus fourth degree elbow comparison). Moreover, eighth degree polynomial kernels have drastically lower performance than any aforementioned kernel (mean difference of shoulder test set performance = 0.64, elbow = 0.75 FVAF, $p < 0.003$). This drop in performance is due to the fact that individual $k(x, y)$'s take on exceedingly large values as d increases. The result is that

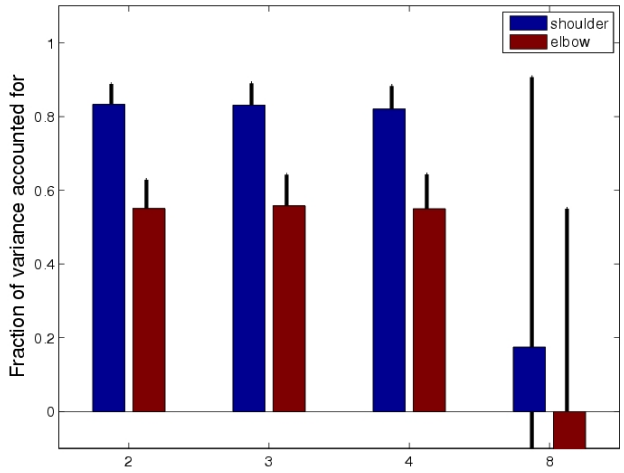


Figure 5.6: Prediction performance as a function of polynomial kernel degree for LSKR on the R1 data set. Results are given for torque (blue=shoulder, red=elbow) using 10 training folds.

the matrix in Equation 2.20 is not well conditioned for inversion. In turn, this leads to the construction of ill-formed models.

5.2.4 Pseudo-Inverse Versus Least-Squares Kernel Regression

Since Pseudo-Inverse was the highest performer of any prediction method previously implemented on this data set, we compare it to the current LSKR results (Goldberg and Fagg, 2005). The test set performance for both torque and Cartesian position predictors on data set R1, depicted in Figure 5.7, shows that Pseudo-Inverse performs similarly to LSKR with a linear kernel ($p > 0.15$). Since LSKR and Pseudo-Inverse both capture linear functions over the input space, they have the same representational ability. Thus, we expect similar results from them. One difference, however, is LSKR's use of a complexity term to help with overfitting. Nonetheless, with 18 folds of training data, the Pseudo-Inverse method does not suffer from difficulties with overfitting.

Furthermore, as Figure 5.7 shows, the quadratic kernel SVM predicts torque more accurately than Pseudo-Inverse ($p < 8.6 \times 10^{-4}$). In fact, the quadratic kernel improves the prediction 0.031 FVAF for shoulder and 0.045 FVAF for elbow torque

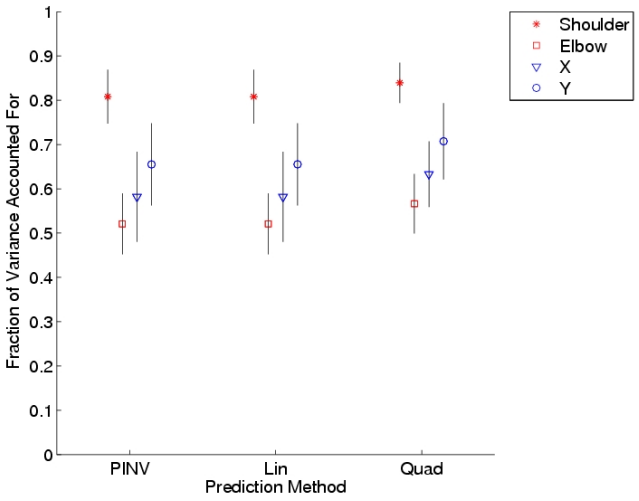


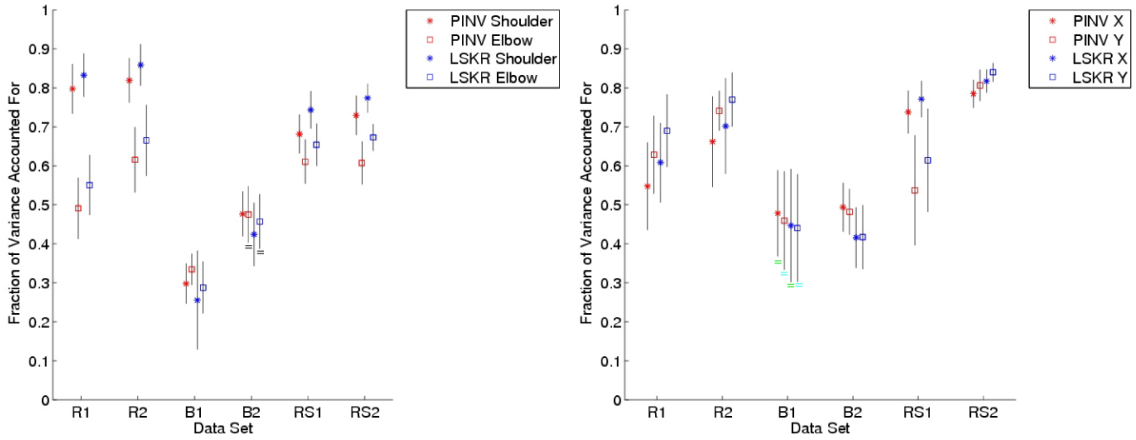
Figure 5.7: Prediction performance as a function of prediction method on data set R1 using 18 training folds. Torque (red) and Cartesian position (blue) results are given for pseudo-inverse, linear kernel LSKR, and quadratic kernel LSKR.

on data set R1. As Figure 5.7 also shows, the quadratic kernel SVM predicts Cartesian position more accurately than Pseudo-Inverse ($p < 8.5 \times 10^{-4}$). The quadratic kernel improves the prediction 0.05 FVAF for both X and Y hand position.

Figure 5.8 expands this comparison to six, independent data sets. Results with like-colored equals signs underneath them are considered equivalent ($p > .05$). Analyzing the data from Figure 5.8(a) with a paired t-test shows that LSKR with a quadratic kernel significantly outperforms Pseudo-Inverse predictions of torque in eight out of the twelve shoulder and elbow comparisions ($p < 6.7 \times 10^{-6}$). In three of the comparisons, Pseudo-Inverse performs better ($p < 0.045$), while in one there is not a significant difference between the two methods ($p > 0.095$).

Figure 5.8(b) shows that LSKR with a quadratic kernel significantly outperforms Pseudo-Inverse predictions of Cartesian position in eight out of the twelve X and Y comparisions ($p < 0.0013$). In two of the comparisons, Pseudo-Inverse performs better ($p < 0.0004$), while in two others there is not a significant difference between the two methods ($p > 0.058$).

It is important to address why LSKR with a quadratic kernel might perform better. Each dimension in the input corresponds to a neuronal firing rate at a given time. Via multiplication, the quadratic kernel effectively "ands" together the firing rates of two dimensions (for the feature to be non-zero, both cells would have to be



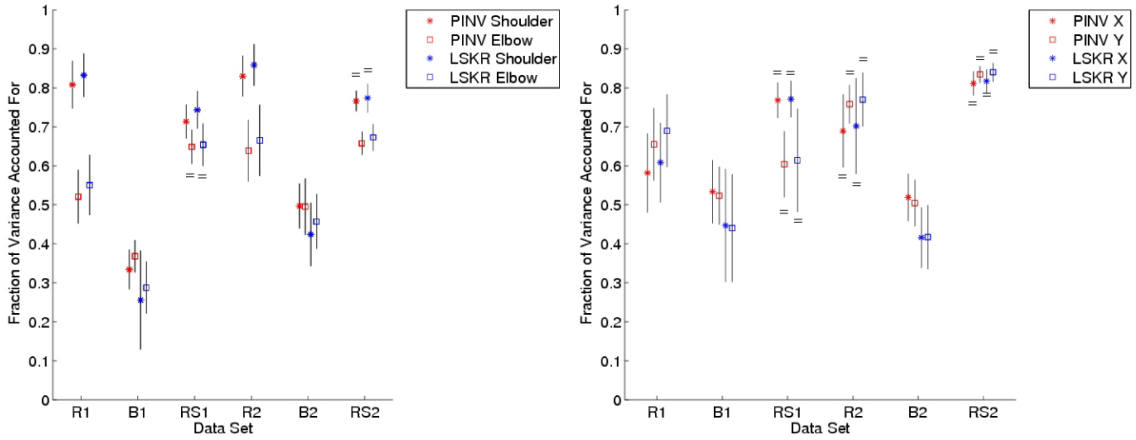
(a) Torque

(b) Cartesian Position

Figure 5.8: Comparison of prediction performance for Pseudo-Inverse (red) to LSKR with a quadratic kernel (blue) for six independent data sets. Equal signs above or below Pseudo-Inverse versus LSKR comparisons mean there is no significant difference between them ($p > .05$).

firing to some degree). In addition to predicting from a particular cell's firing at a time t before movement, the polynomial features enable the learned function to capture situations where 1) the same cell fires at two different times before movement, 2) two different cells fire at the same time, and 3) two different cells fire at different times before movement. It seems that this additional information, transforming the number of dimensions from approximately 1100 to more than a half million, is useful in the prediction process.

Thus, SVM's might have potential for use in Brain Machine Interfaces. However, execution time is currently a major drawback. Pseudo-Inverse requires at most 20 minutes for learning will full size data sets, while LSKR currently requires at least 6 days (Intel Xeon processor, 3GHz clock speed, 6GB RAM). Thus, a practical question is whether LSKR, with less training data, can still outperform Pseudo-Inverse. Using 10 training folds on data set R1 takes a more reasonable amount of computation time (about 1.5 days). However, each data set consists of a different number of samples. We selected the number of training folds for the five remaining data sets so as to approximate the number of samples in 10 training folds of R1. In



(a) Torque

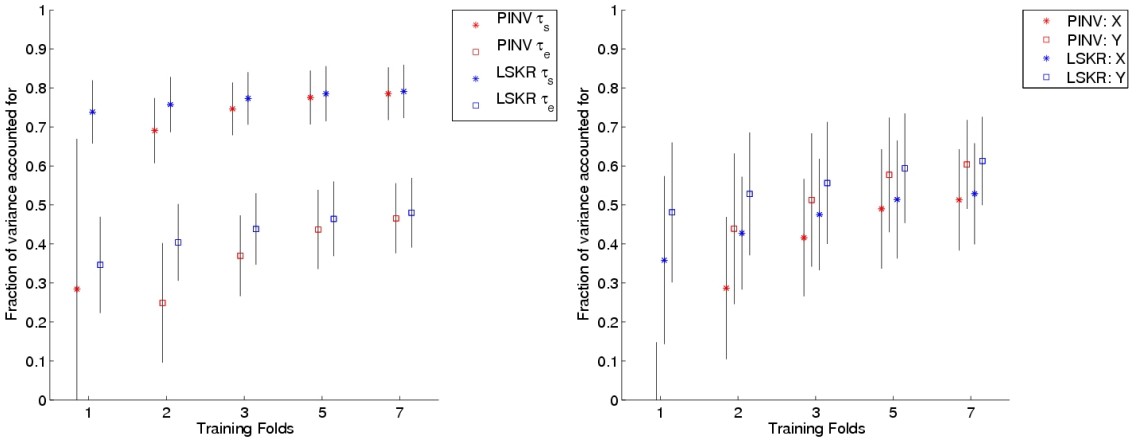
(b) Cartesian Position

Figure 5.9: Comparison of prediction performance for Pseudo-Inverse (red) with the full 18 training folds to quadratic kernel LSKR (blue) with a reduced amount of training data. Equal signs above or below Pseudo-Inverse vs LSKR comparisons mean there is no significant difference between them ($p > 0.05$).

Figure 5.9, we contrast the results of quadratic kernel LSKR, trained on the reduced amount of data, with Pseudo-Inverse trained on the full 18 training folds of data.

A paired t-test shows that LSKR with a quadratic kernel on a reduced size training set significantly outperforms Pseudo-Inverse predictions of torque on an 18 fold training set in six out of the twelve shoulder and elbow comparisions ($p < 0.014$). In four of the comparisons, Pseudo-Inverse performs better ($p < 0.003$), while in two there is not a significant difference between the two methods ($p > 0.18$). For Cartesian predictions on a reduced training set size, LSKR significantly outperforms Pseudo-Inverse in two out of the twelve comparisions ($p < 0.0320$). In four of the comparisons, Pseudo-Inverse performs better ($p < 0.007$), while in six others there is not a significant difference between the two methods ($p > 0.10$). If LSKR can be made more efficient, there is some hope for improvement in performance (on the larger data sets) compared to Pseudo-Inverse, even on a reduced training set size. Nevertheless, the benefit is rather small.

We contrasted prediction performance with linear kernel LSKR to PINV as a function of the number of training folds. Figure 5.10 shows torque and Cartesian position prediction performance on data set R1 for Pseudo-Inverse and linear kernel



(a) Torque

(b) Cartesian Position

Figure 5.10: Comparison of prediction performance of torque and Cartesian position for Pseudo-Inverse (red) to LSKR with a linear kernel (blue) with a small, varying training set size. For one training fold, the mean difference over torque and Cartesian is 0.89 FVAF ($p < 10^{-19}$). For seven training folds, the mean difference is 0.01 FVAF ($p < 10^{-17}$).

LSKR. The mean difference in torque and Cartesian position prediction performance between Pseudo-Inverse and LSKR with one training fold is 0.89 FVAF ($p < 10^{-19}$). For seven training folds, the difference decreases to 0.01 FVAF ($p < 10^{-17}$). The trend is the same for data sets R1, RS1, and RS2, but not for B1 or B2. Since the only difference between the Pseudo-Inverse algorithm and linear kernel LSKR is the complexity term in the minimization function, this term must be reducing overfitting for LSKR when there is a small amount of training data.

5.2.5 Adding Proprioceptive Feedback

Feedback information describing the state of the limb is made available to motor control circuitry at a variety of levels, including the spinal cord and the primary motor cortex. Moreover, Georgopoulos *et al.* (1982) reported an average delay of approximately 100 ms from the time cells begin to discharge to the initiation of movement. The activity of MI cells could therefore be affected by the delayed state of the limb. One simple approach to augment the linear model with this information

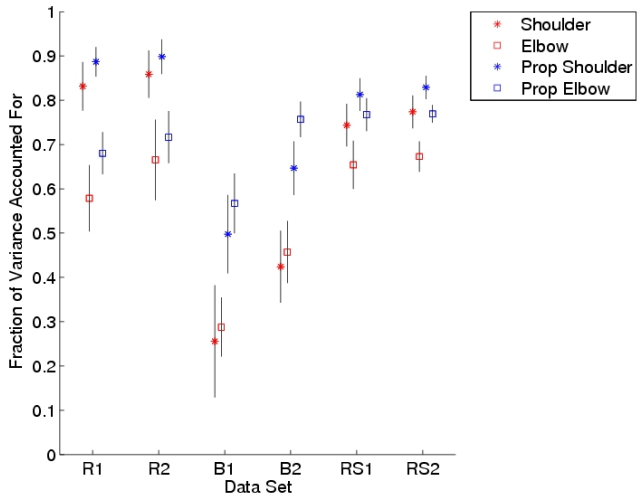


Figure 5.11: Comparison of torque prediction performance of LSKR with a quadratic kernel (red) to prediction performance including proprioceptive feedback (joint angle and velocity) at a delay of 100 ms (blue). Results are shown for six data sets on the reduced amount of training data.

is to add features that describe this delayed limb state. For predicting joint torque, we add features that correspond to the delayed (100 ms) joint position and velocity. These features are treated the same as those describing the cell activity at a given time: each contributes linearly to the torque predictions.

Figure 5.11 contrasts the Fraction of Variance Accounted For for torque predictions without linear proprioceptive feedback to predictions with it. LSKR with a quadratic kernel is depicted. In all six data sets, adding non-linear proprioceptive feedback significantly increases the prediction accuracy ($p < 2.49 \times 10^{-6}$). Therefore, including proprioceptive feedback into the model gives a notable boost to prediction performance.

Moreover, figure 5.12 compares torque prediction performance from LSKR with a quadratic kernel to Pseudo-Inverse, both including linear proprioceptive feedback. In this case, LSKR significantly outperforms Pseudo-Inverse in six out of the twelve shoulder and elbow comparisions ($p < 10^{-10}$). In three of the comparisons, Pseudo-Inverse performs better ($p < 0.033$), while in three others there is not a significant difference between the two methods ($p > 0.18$).

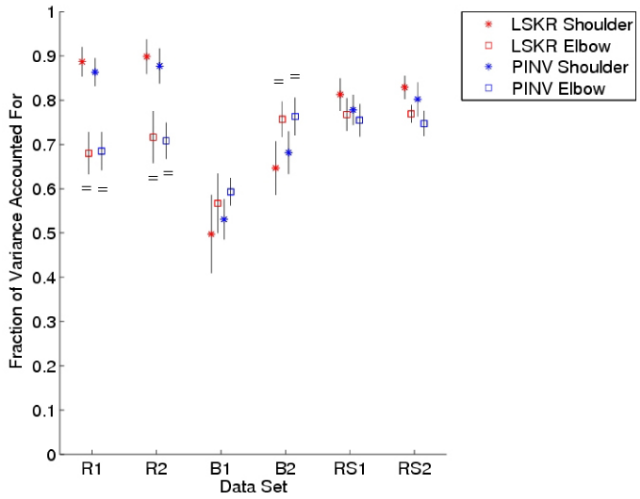


Figure 5.12: Comparison of torque prediction performance for LSKR with a quadratic kernel (red) to Pseudo-Inverse (blue) over six data sets on the reduced amount of training data. The models are built using linear proprioceptive feedback (joint angle and joint velocity). Equal signs above or below LSKR vs Pseudo-Inverse comparisons mean there is no significant difference between them ($p > 0.18$)

5.2.6 Comparing Torque and Cartesian Position Predictions

Whether one can predict intrinsic or extrinsic motion better than the other is a widely debated argument in the literature (Shadmehr and Mussa-Ivaldi, 1993; Ajemian *et al.*, 2000; Scott and Kalaska, 1997; Schwartz *et al.*, 1988). We examined this question using our predictions of torque and Cartesian position on the quadratic LSKR-derived models. Since those predictors are in different units, we must be able to predict both shoulder and elbow more accurately than both X position and Y position, or vice versa, in order to claim a significant difference in prediction accuracy.

Figure 5.13 shows the results of LSKR with a quadratic kernel predicting torque and Cartesian position for six independent data sets. The null hypothesis that Cartesian position and torque performance predictions are drawn from the same distribution can only be rejected in the cases of B1 and RS2 (mean difference of 0.172 FVAF for B1, 0.105 FVAF for RS2, $p < 0.001$). In both of these cases, a Cartesian position predictor outperforms a torque predictor. When providing linear proprioceptive feedback to the torque model, however, torque predictions are significantly more accurate than Cartesian position in data sets B1 and B2 (mean difference of 0.089 FVAF for B1, 0.29 FVAF for B2, $p < 0.04$). With or without

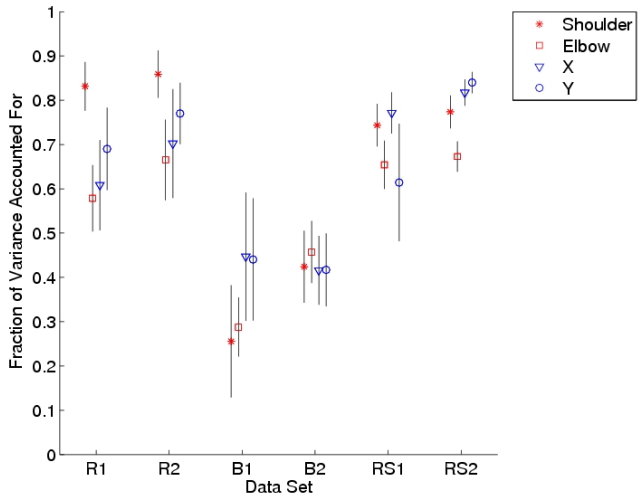


Figure 5.13: Comparison of prediction performance for torque (red) and Cartesian position (blue) using LSKR with a quadratic kernel over six independent data sets.

linear proprioceptive feedback, there was no significant difference in four out of the six data sets. Consistent with Wu and Hatsopoulos (2006), these findings do not add weight to the argument that most cells in M1 only encode arm motion in terms of only one type of variable (in this case, Cartesian position or torque).

Chapter 6

Conclusions

Some of the challenges of predicting arm motion from the recent history of activity of large numbers of cells are how to represent the models, how to infer the model parameters from a training set, and how to measure the performance of the model.

One of the traditions in the BMI domain is to measure performance using the R^2 metric. However, we are concerned with the desired path of the subject's hand, not just a path correlated with it. I argue for using Fraction of Variance Accounted For (FVAF), an alternative, but related measure.

The Population Vector approach is one of the traditional means of representing and training the transformation from cell activity to arm motion. Although intuitive, the approach is fundamentally heuristic. Alternatively, the parameters can be selected to minimize the mean-squared prediction error on a training set through the Pseudo-Inverse approach. We find that although the two approaches perform comparably with respect to the R^2 measure, the population vector approach suffers dramatically when the FVAF measure is used. This is the case because although the population vector approach produces predictions that are correlated with the ground truth, the predictions tend to be far from this truth. In addition, with the Pseudo-Inverse solution to the Wiener filter, we can express models that individually weight M1 action potentials from a range of time delays before the predicted movement. This property has the effect of increasing prediction accuracy further.

Moreover, we found that linear kernel Least Squares Kernel Regression (LSKR) outperforms Pseudo-Inverse when there is a small amount of training data. Since the only practical difference between linear kernel LSKR and Pseudo-Inverse is the complexity term in the cost function, this performance gain must be due to the

fact that the complexity term is reducing overfitting. However, since Pseudo-Inverse reconstructions require a fraction of the time of LSKR, one next step is to create a singularity-robust Pseudo-Inverse algorithm that will incorporate this complexity term in its minimization function.

Furthermore, we have shown that a non-linear, kernel-based method (LSKR with a quadratic kernel) often outperforms a linear method. This is logical since we know there are non-linear transformations between the spinal cord and the arm. Moreover, the quadratic kernel allows us to include the coincidence of firing of cells at different times. It stands to reason that such coincidences within the central nervous system would encode information that is critical to arm motion. However, the time required for model learning and querying is a major drawback.

Finally, we found that in the majority of the data sets there was no significant difference between prediction performance of intrinsic or extrinsic motion. This may be be due to the fact that many different variables are represented in MI or due to the conditions of the test. In particular, the tasks involved planar, two degree-of-freedom movements. Under these conditions, it may be difficult to differentiate between extrinsic and intrinsic codings (Shadmehr and Mussa-Ivaldi, 1993).

The fact that pure linear models do so well and the nonlinear models only perform a little better might mean that there is not much more to be gained in terms of performance. However, one of the advantages of LSKR is that custom kernels can be created that transform the data into particularly useful feature spaces. In equation 2.22, the $\phi(\mathbf{x_i})^T\phi(\mathbf{x})$ term is a measure of similarity between a training set point and the query point. In this work, this similarity is based on the sequence of spike rate estimates for a set of 50 ms bins. By grouping spikes into 50 ms bins, it is possible that a substantial amount of information is lost or that this information is corrupted in some way. For example, this method does not differentiate the occurrence of spikes at the beginning or end of a bin. Furthermore, the measure can be affected by subtle alignment shifts between the spike patterns and the bin boundaries. Alternatively, one could begin to define the similarity of a training set point and query point in terms of how well the two sequences of spikes line up with one another. This is the essence of a set of kernels referred to as *spike kernels* (Chi *et al.*, 2007). Thus, a next step will be learning how to create these custom kernels, such as spike kernels.

What's more, implementing an ϵ-insensitive loss function might help reduce overfitting. When building the support vectors from the training set, an ϵ-insensitive loss function ignores prediction errors within the ϵ-error bound. Thus, the model does not adjust for very small changes in prediction performance, and therefore tends to capture the general trends in the functions. Moreover, this loss function allows us to eliminate support vectors for which α is 0 (and ϵ-insensitive error is 0), reducing the cost of the query.

Bibliography

Ajemian, R., Bullock D., Grossberg, S. (2000). "Kinematic Coordinates in which Motor Cortical Cells Encode Movement Direction." *Journal of Neurophysiology*, 84:2191-2203.

Browne, M.W. (2000). "Cross-validation methods." *Journal of Mathematical Psychology*, 44:108-132.

Carmena, J., Lebedev, M., Crist, R., O'Doherty, J., Santucci, D., Dimitrov, D, Patil, P., Henriquez, C, Nicolelis, M. (2003). "Learning to Control a Brain-Machine Interface for Reaching and Grasping by Primates." *Public Library of Science Biology*, 1(2):193-208.

Chi, Z., Wu, W., Haga, Z., Hatsopoulos, N., Margoliash, D. (2007). "Template-Based Spike Pattern Identification With Linear Convolution and Dynamic Time Warping." *Journal of Neurophysiology*, 97:1221-1235.

Georgopoulos, A. P., Kalaska, J.F., Caminiti, R., and Massey, J.T. (1982). "On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex." *Journal of Neuroscience*, 2:1527-1537.

Georgopoulos, A. P., Schwartz, A. B., and Kettner, R. E. (1986). "Neuronal population coding of movement direction." *Science*, 233:1416-1419.

Goldberg, D., and Fagg, A. (2005) "Predicting Arm Motion from Cortical Activity." CS Research Conference, University of Oklahoma.

Goldberg, D., and Fagg, A. (2006) "Predicting Arm Motion From Cortical Activity: A Support Vector Approach."

Fagg, A.H., Ojakangas, G., Hatsopoulos, N.G., and Miller, L. "Kinetic trajectory decoding using motor cortical ensembles." (Submitted)

Flash, T., and Hogan, N. (1985). "The coordination of arm movements: An experimentally confirmed mathematical model." *Journal of Neuroscience*, 5:1688-1703.

Hatsopoulos, N. (2005). Personal Communication. University of Chicago.

Hocherman, S. and Wise, S.P. (1991). "Effects of hand movement path on motor cortical activity in awake, behaving rhesus monkeys. *Experimental Brain Research*, 83:285-302.

Houk, J. C., Fagg, A. H., Barto, A. G. (2002). "Fractional Power Damping Model of Joint Motion," In Latash, M., editor, *Progress in Motor Control: Structure-Function Relations in Voluntary Movements.* 2:147-178.

Jensen, D., and Cohen, P. (2000). "Multiple Comparisons in Induction Algorithms." *Machine Learning*, 38(3):309-338.

Morasso, P. (1981). "Spatial control of arm movements." *Experimental Brain Research*, 42:223-227.

Penrose, R. (1955) "A generalized inverse for matrices" *Proceedings of the Cambridge Philosophical Society*, 51:406-413.

Reina, G.A., Moran, D., and Schwartz, A. (2001) "On the Relationship Between Joint Angular Velocity and Motor Cortical Discharge During Reaching." *Journal of Neurophysiology*, 85:2576-2589.

Reina, G.A., Schwartz, A.B. (2003) "Eye-hand coupling during closed-loop drawing: Evidence of shared motor planning?" *Human Movement Science*, 22:137-152.

Schölkopf, B. and Smola, A. (2002). *Learning With Kernels.* Cambridge, Massachusetts: MIT Press.

Scott, Stephen H. and Kalaska, John F. (1997). "Reaching Movements with Similar Hand Paths but Different Arm Orientations. I. Activity of Individual Cells in Motor Cortex." *Journal of Neurophysiology*, 77:826-852.

Scott, S.H. (1999) "Apparatus for measuring and perturbing shoulder and elbow joint positions and torques during reaching." *Journal of Neuroscience Methods*, 89:119-27.

Schwartz, A.B., Kettner, R.E., and Georgopoulos, A.P. (1988). "Primate Motor Cortex and Free Arm Movements to Visual Targets in Three-Dimensional Space. I. Relations Between Single Cell Discharge and Direction of Movement." *Journal of Neuroscience*, 8(8):2913-2927.

Serruya, M., Hatsopoulos, N., Paninski, L., Fellows, M., and Donoghue, J. (2002). "Instant Neural Control of a Movement Signal." *Nature*, 416:141-142.

Serruya, M., Hatsopoulos, N., Fellows, M., Paninski, L., and Donoghue, J. (2003). "Robustness of neuroprosthetic decoding algorithms." *Biological Cybernetics*, 88(3):219-228.

Shadmehr, R. and Mussa-Ivaldi, F.A. (1993). "Adaptive Representation of Dynamics during Learning of a Motor Task." *Journal of Neuroscience*, 14(5):3208-3224.

Shoham, S., Paninski, L., Fellows, M., Hatsopoulos, N., Donoghue, J., and Normann, R. (2005). "Statistical Encoding Model for a Primary Motor Cortical Brain-Machine Interface." *IEEE Transactions on Biomedical Engineering*, 52(7):1312-1322.

Suykens J.A.K., Lukas L., Vandewalle J. (2000). "Sparse approximation using least squares support vector machines." *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2000)*. Geneva, Switzerland, May 2000, II757-II760.

Suykens, J., Van Gestel, T., De Brabanter,J., De Moor, B., and Vandewalle, J. (2002). "LS-SVMlab : A Matlab/C toolbox for Least Squares Support Vector Machines." http://www.esat.kuleuven.ac.be/sista/lssvmlab/. ESAT-SISTA, K.U.Leuven (Leuven, Belgium)

Warland, D., Reinagel, P., and Meister, M. (1997). "Decoding visual information from a population of retinal ganglion cells." *Journal of Neurophysiology*, 78(5):2336-2350.

Wu, W., Black, M., Mumford, D., Gao, Y., Bienenstock, E., and Donoghue, J. (2004). "Modeling and Decoding Motor Cortical Activity Using a Switching Kalman Filter." *IEEE Transactions on Biomedical Engineering*, 51(6):933-942.

Wu, W., and Hatsopoulos, N. (2006). "Evidence against a single coordinate system representation in the motor cortex." *Experimental Brain Research*, 175:197-210.