Journal of Machine Learning Research 9 (2008) 1535-1558 Submitted 12/07; Revised 6/08; Published 7/08

Learning to Combine Motor Primitives Via Greedy Additive Regression

Manu Chhabra MCHHABRA@CS.ROCHESTER.EDU
Department of Computer Science
University of Rochester
Rochester, NY 14627, USA

Robert A. Jacobs ROBBIE@BCS.ROCHESTER.EDU
Department of Brain & Cognitive Sciences
University of Rochester
Rochester, NY 14627, USA

Editor: Peter Dayan

Abstract

The computational complexities arising in motor control can be ameliorated through the use of a library of motor synergies. We present a new model, referred to as the Greedy Additive Regression (GAR) model, for learning a library of torque sequences, and for learning the coefficients of a linear combination of sequences minimizing a cost function. From the perspective of numerical optimization, the GAR model is interesting because it creates a library of "local features"—each sequence in the library is a solution to a single training task—and learns to combine these sequences using a local optimization procedure, namely, additive regression. We speculate that learners with local representational primitives and local optimization procedures will show good performance on nonlinear tasks. The GAR model is also interesting from the perspective of motor control because it outperforms several competing models. Results using a simulated two-joint arm suggest that the GAR model consistently shows excellent performance in the sense that it rapidly learns to perform novel, complex motor tasks. Moreover, its library is overcomplete and sparse, meaning that only a small fraction of the stored torque sequences are used when learning a new movement. The library is also robust in the sense that, after an initial training period, nearly all novel movements can be learned as additive combinations of sequences in the library, and in the sense that it shows good generalization when an arm's dynamics are altered between training and test conditions, such as when a payload is added to the arm. Lastly, the GAR model works well regardless of whether motor tasks are specified in joint space or Cartesian space. We conclude that learning techniques using local primitives and optimization procedures are viable and potentially important methods for motor control and possibly other domains, and that these techniques deserve further examination by the artificial intelligence and cognitive science communities.

Keywords: additive regression, motor primitives, sparse representations

1. Introduction

To appreciate why motor control is difficult, it is useful to quantify its computational complexity. Consider, for example, an agent whose goal is to apply torques to each joint of a two-joint arm so that the endpoint of the arm moves from an initial location to a target location in 100 time steps. Also suppose that torques are discretized to one of ten possible values. In this case, the agent needs

to choose one sequence of torques from a set of 10^{200} possible sequences. Searching this set of possible sequences is clearly a computationally intractable problem.

To ameliorate the computational challenges arising in motor control, it has been hypothesized that biological organisms use "motor synergies" (Bernstein, 1967). A motor synergy is a dependency among the dimensions or parameters of a motor system. For example, a coupling of the torques applied at the shoulder and elbow joints would be a motor synergy. Motor synergies are useful because they reduce the number of parameters that must be independently controlled, thereby making motor control significantly easier (Bernstein, 1967). Moreover, synergies are often regarded as "motor primitives". For our current purposes, we focus here on frameworks in which an agent with a library of motor synergies quickly learns to perform complex motor tasks by linearly combining its synergies. This idea has motivated a great deal of neuroscientific research. For example, Mussa-Ivaldi, Giszter, and Bizzi (1994) identified frogs' motor synergies by stimulating sites in the frogs' spinal cords. Importantly, these authors verified that stimulation of two sites leads to the vector summation of the forces generated by stimulating each site separately.

The idea of characterizing complex movements as linear combinations of motor synergies has also been influential in the fields of artificial intelligence and cognitive science. In these fields, an important research question is how to build an agent with a useful set of synergies or, alternatively, how an agent can learn a useful set of synergies. Approaches to these issues are typically based on techniques from the machine learning literature. Some researchers have developed theories motivated by kernel-based techniques. For example, Thoroughman and Shadmehr (2000) studied the errors in people's reaching movements and concluded that humans learn the dynamics of reaching movements by combining primitives that have Gaussian-like tuning functions. Other researchers have speculated that motor primitives can be learned using dimensionality-reduction techniques. For example, Sanger (1995) analyzed people's cursive handwriting using principal component analysis (PCA) to discover their motor synergies. He showed that linear combinations of these synergies closely reconstructed human handwriting. Other examples using dimensionality-reduction to learn motor primitives include Chhabra and Jacobs (2006), d'Avella, Saltiel, and Bizzi (2003), Fod, Matarić, and Jenkins (2002), Jenkins and Matarić (2004), Safanova, Hodgins, and Pollard (2004), Sanger (1994), and Todorov and Ghahramani (2003, 2004).[1]

The fact that novel motor tasks can often be performed by linearly combining motor synergies is a surprising result. To see why this result is unexpected, consider the case in which an agent needs to control a two-joint arm to perform a motor task. Suppose that a task is defined as a sequence of desired joint angles (i.e., desired angles for the shoulder and elbow joints at each time step of a movement), that a cost function is defined as the sum of squared error between the desired and actual joint angles, and that the agent has a library of motor synergies where a synergy is a sequence of torques (i.e., torques for the shoulder and elbow joints at each time step of a movement). The agent attempts to perform the motor task by finding a set of coefficients for a linear combination of synergies minimizing the cost function. (This optimization might be performed, for example, using a gradient descent procedure, known as policy gradient, in which an agent uses the gradient of the cost function with respect to the coefficients, Sutton, McAllester, Singh, and Mansour, 1999;

1. Our review focuses on frameworks in which complex movements are expressed as linear combinations of motor primitives. There are, of course, frameworks that use motor primitives in other ways. A reader interested in this topic may want to see Bentivegna (2004), Ijspeert, Nakanishi, and Schaal (2003), Lau and Kuffner (2005), Lee, Chai, Reitsma, Hodgins, and Pollard (2002), Peters and Schaal (2006), and Stolle and Atkeson (2006), among other articles.

Williams, 1992.) Should we expect that a good set of linear coefficients—a set that leads to a near-zero value of the cost function—will exist and, if so, be easy to find? We believe that the answer is no. Recall that synergies are defined in terms of torques, the agent **linearly** combines synergies, the cost function is defined in terms of joint angles, and there is a **nonlinear** relationship between torques and joint angles. Finding a good set of linear coefficients should be a difficult optimization problem because the nonlinear function relating coefficient values to values of the cost function will contain many local minima. As a matter of terminology, we refer to this as the "Motor Basis Optimization Problem".

The Motor Basis Optimization Problem motivates the need to think about good ways of constructing a library of synergies, and good ways of learning to linearly combine the synergies to perform novel motor tasks. In this paper, we propose a new learning model that learns a sparse and overcomplete representation of the space of potentially useful motor commands, and learns to linearly combine elements of this representation using a "greedy additive regression" procedure. At a high level of abstraction, our procedure closely resembles the use of greedy additive schemes for feature selection in recent machine learning systems (e.g., Perkins, Lacker, and Theiler, 2003; Viola and Jones, 2004). For example, Viola and Jones (2004) used AdaBoost (Freund and Schapire, 1997; Schapire, 1990) to create a fast and robust classifier for detecting faces in visual images. They started by creating a large library of image feature detectors. They then constructed a classifier in an additive manner. At each iteration, a new classifier was created by adding a feature detector to the old classifier. The feature detector that was added was the one whose use reduced the error of the old classifier by the largest amount. The end result after several iterations was a successful classifier with a sparse representation in the sense that it used relatively few feature detectors from the library.

This paper introduces a new learning model for motor control referred to as the Greedy Additive Regression (GAR) model. The GAR model maintains a library of torque sequences (i.e., motor synergies). If possible, the GAR model learns new movements by additively combining sequences in this library. If not possible, new movements are learned by other means (e.g., via feedback error learning). The torque sequences for these new movements are then added to the library. (Unlike Viola and Jones, we do not construct a library of primitives by hand. Instead, we learn the primitives in this library using the set of training tasks.)

We present results comparing the performances of the GAR model with those of another model, referred to as the PCA model, that can be regarded as a generic example from a large class of approaches commonly used in the artificial intelligence and cognitive science literatures. The PCA model learns a library of motor primitives using PCA, and finds coefficients for linearly combining the primitives using gradient descent. Whereas the PCA model often yields poor results, the GAR model consistently shows excellent performance. We find that the acquisition of new movements by the GAR model is rapid when the library is used. Moreover, the library is overcomplete and also sparse, meaning that only a small fraction of the stored torque sequences are used when learning a novel movement. The library is also robust in at least two different ways. First, after an initial training period, nearly all novel movements can be learned as additive combinations of sequences in the library. Consequently, learning from scratch via, for example, feedback error learning becomes rarer over time. Second, the library is also robust in the sense that it shows good generalization when an arm's dynamics are altered between training and testing conditions. If, for example, an arm is suddenly required to carry a payload during testing, torque sequences in the library can still be additively combined to rapidly learn new movements with this altered arm. We also demonstrate that the model works well regardless of whether motor tasks are specified in joint space or Cartesian

space. Based on these results, we believe that the GAR model contains several desirable properties, including a library which maintains a sparse and overcomplete representation of the space of potentially useful motor commands, and an additive regression optimization procedure which is fast and robust.

This article is organized as follows. Section 2 describes the two-joint arm that we simulated, and Section 3 describes the motor tasks that we used. Section 4 describes the Greedy Additive Regression model. Section 5 reports the simulation results comparing the performances of the GAR and PCA models under a variety of conditions. In Section 6, we briefly consider the performances of these models when the system to be controlled is a linear system with redundancy. Section 7 states our conclusions and directions for future research.

2. Simulated Two-Joint Arm

We simulated a two-joint arm that coarsely resembles a human arm (Li and Todorov, 2004). The arm can be written as a second-order nonlinear dynamical system (Hollerbach and Flash, 1982):

$$\mathcal{M}(\theta)\ddot{\theta} + \mathcal{C}(\theta,\dot{\theta}) + \mathcal{B}\dot{\theta} = \tau$$

where τ is a vector of torques, $\theta, \dot{\theta}$, and $\ddot{\theta}$ are vectors of joint angle positions, velocities, and accelerations, respectively, $\mathcal{M}(\theta)$ is an inertial matrix, $\mathcal{C}(\theta,\dot{\theta})$ is a vector of Coriolis forces, and $\mathcal{B}$ is a joint friction matrix. The mathematical forms of these variables are as follows:

$$\begin{aligned}
\mathcal{M}(\theta) &= \begin{pmatrix} a_1 + 2a_2\cos\theta_2 & a_3 + a_2\cos\theta_2, \\ a_3 + a_2\cos\theta_2 & a_3 \end{pmatrix}, \\
\mathcal{C}(\theta,\dot{\theta}) &= \begin{pmatrix} -\dot{\theta_2}(2\dot{\theta_1} + \dot{\theta_2}), \\ \dot{\theta_1}^2 \end{pmatrix} a_2\sin\theta_2, \\
\mathcal{B}(\theta) &= \begin{pmatrix} b_{11} & b_{12}, \\ b_{21} & b_{22} \end{pmatrix}, \\
a_1 &= I_1 + I_2 + m_2 l_1^2, \\
a_2 &= m_2 l_1 s_2, \\
a_3 &= I_2
\end{aligned}$$

where I_1 and I_2 are the moments of inertia of the two links, m_1 and m_2 are the masses of the two links, and s_1 and s_2 are the distances from the joints to the links' center of masses. We used the same parameter values for the arm as Li and Todorov (2004). These values are given in Table 1.

3. Motor Tasks

A motor task is to apply torques to the arm so that it follows a desired trajectory defined in joint space. A desired trajectory is specified by a sequence of joint angles written as a 2×50 matrix of 2 joint angles over 50 time steps, where each time step corresponds to 7 ms of simulation. This trajectory is created in several stages (see Figure 1). First, we generate a trajectory in Cartesian space. To generate this trajectory, an initial position for the end-effector of the arm is chosen by randomly sampling each joint angle from a uniform distribution between 0 and $\pi/2$. Then a final position for the end-effector is chosen as the end point of a vector v of length d at an angle ψ starting

Constant	Value	Constant	Value
b_{11}	0.05 kgm^2s^{-1}	b_{22}	0.05 kgm^2s^{-1}
b_{21}	0.025 kgm^2s^{-1}	b_{12}	0.025 kgm^2s^{-1}
m_1	1.4 kg	m_2	1.0 kg
l_1	0.30 m	l_2	0.33 m
s_1	0.11 m	s_2	0.16 m
I_1	0.025 kgm^2	I_2	0.045 kgm^2

Table 1: Values of constants used in the simulation of a two-joint arm.

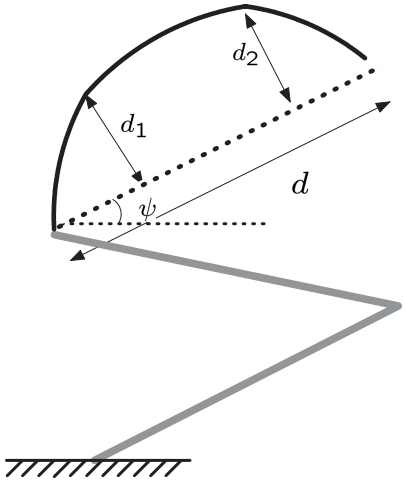

Figure 1: Schematic drawing depicting how a motor task is generated.

at the end-effector's initial position, where d and ψ are chosen uniformly at random between 10 cm and 30 cm, and between 0 and 2π, respectively. Next, two via points are chosen at distances d_1 and d_2 perpendicularly away from the vector v at locations $d/3$ and $2d/3$. Both d_1 and d_2 are drawn uniformly at random between -10 cm and 10 cm. Finally, a trajectory is generated by fitting a smooth cubic spline between the initial position, the two via points, and the final position. The Cartesian-space trajectory is converted to a joint-space trajectory by solving the robot arm's inverse-kinematics using the MATLAB robotics toolbox (Corke, 1996). The duration of movement is set to 350 ms, and the resulting joint-space trajectory is sampled at 7 ms intervals to get the 2×50 matrix defining a desired trajectory.

Given a desired joint-space trajectory, a motor task is to apply a time-varying torque to the arm so that the arm follows the desired trajectory. Torques are sampled every 7 ms, meaning that torques can be written as a 2×50 matrix. The cost function corresponding to the motor task is the sum of

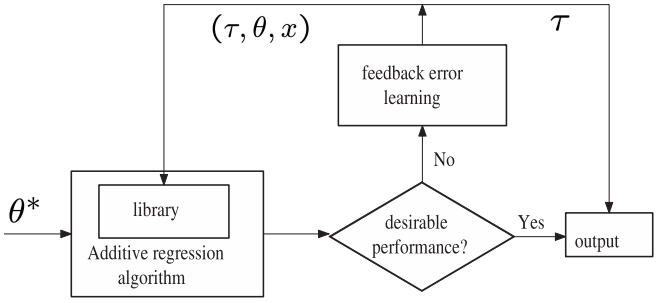

Figure 2: A schematic description of the Greedy Additive Regression(GAR) model. A desired trajectory θ^* is given as an input to the model. An additive regression algorithm is then used to construct a torque sequence by linearly combining sequences from the library. If this algorithm fails to find a linear combination yielding good performance, the model acquires a new torque sequence by other means (e.g., via feedback error learning), and then adds this new sequence to the library.

squared error between the desired and actual joint positions:

$$J = \sum_{t=1}^{50} \sum_{i=1}^{2} (\theta_i^*(t) - \theta_i(t))^2 \tag{1}$$

where $\theta_i^*(t)$ and $\theta_i(t)$ are the desired and actual angles of joint i at time t, respectively.

The motor tasks defined here are more complex than tasks often used in the literature in at least two respects. First, the desired Cartesian-space trajectories used here are typically highly curved, as opposed to straight-line reaching movements which are commonly used in experimental and computational studies of motor control. Second, our tasks specify desired joint angles at every time step. These tasks are more constrained than tasks that specify initial and final desired joint angles but allow an arm to have any joint angles at intermediate time steps.

4. The Greedy Additive Regression Model

We propose a model of motor learning called the Greedy Additive Regression (GAR) model. This model rapidly learns new motor tasks using a library of torque sequences. A schematic description of the model is given in Figure 2.

When a new motor task arrives, the model first checks whether a linear combination of sequences from the library achieves good performance on this task. Good performance is defined as a cost J less than ε (we set $\varepsilon = 0.05$ in our simulations). A potentially good linear combination is found via the additive regression algorithm which is described below. If a linear combination with good performance can be found, then this linear combination is used and nothing else needs to be done. If, however, such a linear combination is not found, then the model needs to learn a new torque sequence by other means. In the simulations reported in Section 5, we used feedback error learning to learn this new torque sequence (Kawato, Furukawa and Suzuki, 1987; see also Atkeson

and Reinkensmeyer, 1990, and Miller, Glanz, and Kraft, 1987).[2] The new torque sequence is then added to the library. Because a library has a fixed size of K, the addition of a new sequence may require the removal of an old sequence. Intuitively, the model removes the torque sequence that has been least used during the motor tasks that it has performed. Let n be an index over motor tasks, k be an index over sequences in the library, and $|\rho^k(n)|$ be the absolute value of the linear coefficient $\rho^k(n)$ assigned to sequence k on task n by the additive regression algorithm. The percent of the model's "total activation" that sequence j accounts for, denoted a^j, is defined as:

$$a^j = \frac{\sum_n |\rho^j(n)|}{\sum_n \sum_k |\rho^k(n)|} \times 100.$$

A sequence with large coefficients (based on magnitude, not sign) on many tasks would account for a large percent of the model's total activation, whereas a sequence with near-zero coefficients would account for a small percent. The model removes the torque sequence that accounts for the smallest percent of its total activation.

To complete the description of the GAR model, we need to describe the additive regression algorithm for finding potentially good linear combinations of torque sequences from the library for a motor task. As mentioned above, this algorithm is motivated by recent machine learning systems that have used greedy additive procedures for feature selection (Perkins, Lacker, and Theiler, 2003; Viola and Jones, 2004).

The additive regression algorithm is an iterative procedure. At iteration t, the algorithm maintains an aggregate torque sequence $F^{(t)}$ to perform a motor task such that:

$$F^{(t)} = \sum_{j=1}^{t} \rho_j f_j \tag{2}$$

where f_j is a sequence in the library and ρ_j is its corresponding coefficient. Note that the aggregate sequence $F^{(t)}$ is a weighted sum of t sequences from the library, but these sequences are not necessarily distinct. It is possible that the same sequence appears more than once in the summation in Equation 2. At each iteration of the algorithm, a sequence from the library is selected (with replacement), and a weighted version of this sequence is added to $F^{(t)}$ in order to create $F^{(t+1)}$. That is,

$$F^{(t+1)} = F^{(t)} + \rho_{t+1} f_{t+1} \tag{3}$$

where f_{t+1} is the library sequence selected to be added and ρ_{t+1} is its corresponding coefficient.

How does the algorithm choose f_{t+1} and ρ_{t+1}? Each torque sequence in the library is associated with a trajectory of joint angles. For computational convenience, the algorithm sets this trajectory to

2. In brief, feedback error learning proceeds as follows. An adaptive feedforward controller is used in conjunction with a fixed feedback controller. At each moment in time, the feedforward controller receives the desired joint positions, velocities, and accelerations, and produces a feedforward torque vector. The feedback controller receives the current and desired joint positions and velocities and produces a feedback torque vector. The sum of the feedforward and feedback torque vectors is applied to the arm, and the resulting joint accelerations are observed. During the learning portion of the time step, the inputs to the feedforward controller are set to the current joint positions, velocities, and accelerations, and the target output is set to the torque vector that was applied to the arm. This controller's parameters are then adapted so that it better approximates the mapping from the inputs to the target output in the future. Early in training, the outputs of the feedforward controller are near zero and most of the torques are supplied by the feedback controller. As training progresses, the feedforward controller better approximates the arm's inverse dynamics, and it supplies most of the torques. Feedback error learning is an attractive learning procedure because it is unsupervised; it does not require an external teacher but only a simple feedback controller.

a "prototypical" trajectory in the following sense. The position of the arm is initialized so that each joint angle is at its average initial value (i.e., each joint angle is initialized to $\pi/4$). The joint-angle trajectory associated with a torque sequence is then found by applying the sequence to the arm. A sequence is evaluated by correlating its joint-angle trajectory with $\partial J/\partial F^{(t)}$, the gradient of the cost function J with respect to the current aggregate torque sequence. This gradient indicates how the aggregate sequence should be modified so as to reduce the cost. In our simulations, it was obtained by numerically computing the partial derivative of the cost function with respect to each element of the aggregate sequence $F^{(t)}$.[3] The torque sequence whose trajectory is maximally correlated with this gradient, denoted f_{t+1}, is selected. To find the best coefficient ρ_{t+1} corresponding to this sequence, the algorithm performs a line search, meaning that the algorithm searches for the value of ρ_{t+1} that minimizes the cost $J(F^{(t)} + \rho_{t+1} f_{t+1})$ (we implemented a golden section line search; see Press, Teukolsky, Vetterling, and Flannery, 1992, for details). $F^{(t+1)}$ is then generated according to Equation 3, and the optimization proceeds to the next iteration. This process is continued until the value of the cost function converges (see Algorithm 1).

There are several possible perspectives on the additive regression algorithm. The idea of greedily selecting the next primitive from a library has also been explored in the feature selection literature. For example, Perkins, Lacker, and Theiler (2003) used a gradient-based heuristic at each iteration of their learning procedure to select the best feature from a set of features to add to a classifier. Our work differs from their work in many details because the domain of motor control forces us to confront the complexities inherent in learning to control a dynamical system (see also Tassa, Erez, and Smart, 2008). In addition, an appealing aspect of our work is that we use the solutions from prior tasks to create a library of primitives. We find that this practice leads to an overcomplete representation of the control space. Overcomplete representations have been shown to be useful in a wide range of applications (e.g., Lewicki and Sejnowski, 2000; Smith and Lewicki, 2006). In addition, the additive regression algorithm can be seen as performing gradient descent where the direction of the gradient at each iteration is projected onto the library sequence whose trajectory is maximally correlated with this gradient. The algorithm then minimizes the cost function by optimizing the coefficient corresponding to this sequence. The algorithm can also be seen as performing a type of "functional gradient descent" via boosting (readers interested in this perspective should see Bühlmann, 2003, or Friedman, 2001). Lastly, the algorithm can be seen as using "matching pursuit" to identify the next library sequence to add to the aggregate sequence at each iteration (see Mallat and Zhang, 1993, for details).

5. Simulation Results

This section reports a number of results using the GAR model. We compare the performances of the GAR model with those of another model, referred to as the PCA model, that can be regarded as a generic example from a large class of approaches commonly used in the artificial intelligence and cognitive science literatures. The PCA model performs dimensionality-reduction via PCA to learn a library of motor primitives. When given a novel motor task, the PCA model learns to perform the

3. $F^{(t)}$ is a 2×50 matrix. The partial derivative of the cost function with respect to element (j,k) of $F^{(t)}$ was computed by evaluating the cost of $F_+^{(t)}$ and $F_-^{(t)}$, where $F_+^{(t)}$ is the same as $F^{(t)}$ except that its $(j,k)^{\text{th}}$ element is set to $F^{(t)}(j,k) + \delta$ (similarly, $F_-^{(t)}$ is set to $F^{(t)}(j,k) - \delta$; we set $\delta = 0.01$.) The partial derivative was then approximated by $\frac{J(F_+^{(t)}) - J(F_-^{(t)})}{2\delta}$.

input : A desired trajectory θ^*
assume : A library $\mathcal{L} = \{(f^k, \theta^k)\}$ of torque sequences and their corresponding trajectories
output : An aggregate torque sequence F that minimizes cost J
$t \leftarrow 0$; $F \leftarrow 0$;
repeat
  $t \leftarrow t+1$
  numerically compute $\bigtriangledown J = \frac{\partial J}{\partial F}$
  From the library $\mathcal{L}$, pick a sequence f^k such that $\bigtriangledown J$ and θ^k are maximally correlated
  $f_{t+1} \leftarrow f^k$
  do a line search to find ρ_{t+1} that minimizes $J(F + \rho_{t+1} f_{t+1})$
  $F \leftarrow F + \rho_{t+1} f_{t+1}$
until J converges
output F

Algorithm 1: Additive regression algorithm for finding a linear combination of torque sequences from the library.

task using a policy gradient optimization procedure (Sutton, McAllester, Singh, and Mansour, 1999; Williams 1992) to learn a set of coefficients for linearly combining the motor primitives. (We regard the PCA model as generic because we regard PCA and gradient descent as generic dimensionality-reduction and optimization procedures, respectively.)

5.1 GAR versus PCA

In the PCA model, the library of motor synergies was created as follows. We first generated 3000 motor tasks as described in Section 3, and then used feedback error learning to learn a torque sequence for each task. This gave us 3000 sequences, each defined by a matrix of size 2×50. We re-stacked the rows of each matrix to form a vector of size 1×100. This gave us 3000 vectors (or data points) lying in a 100-dimensional space. We then performed dimensionality reduction via PCA. The 100 principal components accounted for all the variance in the data and, thus, these components were used as the library for the PCA model. We refer to these components as PCA sequences.

To learn to perform a novel motor task from a test set, the PCA model searched for good linear combinations of the PCA sequences. This search was conducted using a policy gradient procedure (Sutton, McAllester, Singh, and Mansour, 1999; Williams 1992). The linear coefficients were initialized to random values. At each iteration of the procedure, the gradient of the cost function with respect to the coefficients was numerically computed, and a line search in the direction of the gradient was performed (a golden section search method was implemented; see Press, Teukolsky, Vetterling, and Flannery, 1992, for details). This process was repeated until the cost function converged.

The GAR model was implemented as follows. Its library of torque sequences was created by running the model on 3000 motor tasks. The model's library size was set to 100. The sequences in this library at the end of training are referred to as GAR sequences. To learn to perform a novel motor task from a test set, the GAR model learned to linearly combine the GAR sequences using the additive regression algorithm described above.

The PCA and GAR models are two possible combinations of ways of creating libraries—one can create libraries of either PCA or GAR sequences—and ways of linearly combining sequences from

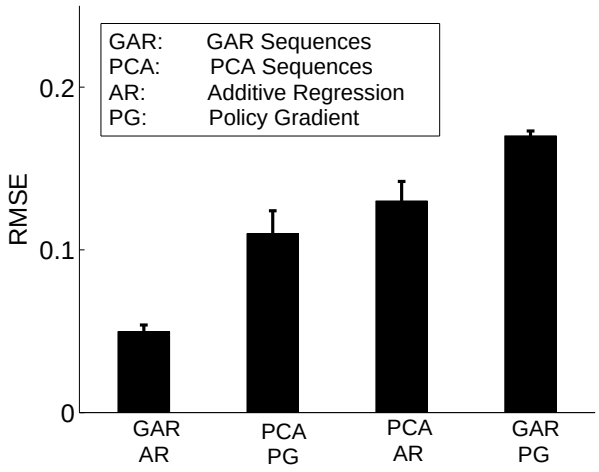

Figure 3: Average root mean squared errors of four systems on a test set of 100 novel motor tasks (the error bars show the standard errors of the means). The four systems use: (i) GAR sequences with additive regression (GAR model); (ii) PCA sequences with policy gradient (PCA model); (iii) PCA sequences with additive regression; and (iv) GAR sequences with policy gradient.

a library—one can learn linear coefficients through policy gradient or additive regression. The PCA model combines PCA sequences with policy gradient, whereas the GAR model combines GAR sequences with additive regression. For the sake of completeness, we also studied the remaining two combinations, namely, the combination of PCA sequences with additive regression and the combination of GAR sequences with policy gradient.

The results are shown in Figure 3. The horizontal axis gives a system's combination of library sequences and optimization technique. The vertical axis gives a system's average root mean squared error (RMSE where the error is between the desired and actual joint angles) on a test set of 100 novel motor tasks. Clearly, the GAR model (leftmost bar in figure) performed better than the PCA model (second bar from left). To further illustrate this point, the solid line in Figure 4 shows the Cartesian-space desired trajectory for a sample test task. The dashed line shows the trajectory achieved by the GAR model, and the dotted line shows the trajectory achieved by the PCA model. Whereas the GAR model found a curved trajectory that closely approximated the desired trajectory, the PCA model converged to a relatively straight-line movement which coarsely approximated the desired trajectory. Our simulation results suggest that this is a common outcome for the PCA model. It appears that the PCA model (and perhaps any system that uses policy gradient; see Figure 3) is prone to finding poor local minima of the error surface.

In addition to showing that the GAR model outperformed the PCA model, Figure 3 also shows that the GAR model outperformed the other systems considered here. Overall, the results are interesting because they suggest that it is not enough to choose a good library—consider that the system using GAR sequences with policy gradient performed poorly—and that it is also not enough to use a good optimization procedure—the system using PCA sequences with additive regression

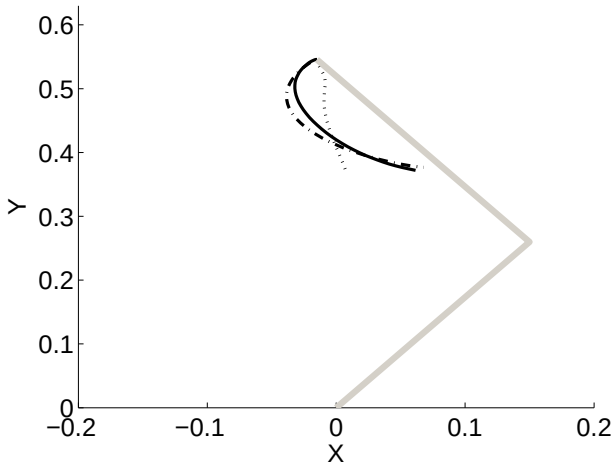

Figure 4: The solid line shows the Cartesian-space desired trajectory for a sample test task. The dashed line shows the trajectory achieved by the GAR model, and the dotted line shows the trajectory achieved by the PCA model.

performed poorly too. Instead, to achieve good performance it is necessary to consider the representational primitives and the optimization procedure as a pair. Representational primitives and optimization procedures are effective if a given procedure is able to find good solutions when the search space is based on these primitives.

Why does the GAR model work so well? Our results suggest that its due to its combination of "local" representational primitives (the GAR sequences) and a "local" optimization procedure (additive regression). To appreciate the coupling between representational primitives and optimization procedures, its important to keep in mind the differences between GAR and PCA sequences, and the differences between additive regression and gradient descent optimization procedures. Each individual GAR sequence is a solution to some task in the training set, whereas an individual PCA sequence is not necessarily a solution to a task but, rather, reflects properties of many tasks. In this sense, a GAR sequence can be regarded as a "local feature," and a PCA sequence can be regarded as a "global feature." Similarly, additive regression can be considered as a local optimization procedure because it adds at most one new feature to its linear combination at each iteration and because, at convergence, its linear combination tends to contain relatively few features. In contrast, gradient descent is a global optimization procedure because it finds linear combinations of all possible features. Because some features can have opposite effects, global optimization procedures lead to interference. Interference can be avoided by using a local optimization method. Local optimization methods have been shown to be effective in motor control in previous research. For example, Atkeson, Moore, and Schaal (1997) stored all previous experiences on control tasks in memory, and used a relatively local regression scheme (where locality was specified in terms of both space and time) to compute control signals for new tasks. They showed that their local learning method performed well, and also ameliorated the problem of global interference from features with opposing effects.

For linear systems and quadratic cost functions, we predict that the use of GAR versus PCA sequences, or additive regression versus gradient descent optimization procedures, should not matter much. Indeed, simulations on a linear system in Section 6 show that all four library/algorithm combinations work equally well. This is because a linear combination of either GAR or PCA sequences is, by linearity, a solution to some task. When searching for a good linear combination, a learner is searching among a set of task solutions for the particular solution which yields good performance on the current target task. This remains true regardless of whether a learner uses GAR or PCA sequences, or additive regression or gradient descent optimization procedures.

For nonlinear systems, however, this is not necessarily the case. With nonlinear systems, our results show that a learner using local primitives (which are task solutions) and local optimization procedures is preferable. This is because, when searching for a good linear combination, the local optimization procedure searches a set of combinations which are relatively close to solutions for some task. In the context of the GAR model, for example, we conjecture that each iteration of the additive regression procedure finds a linear combination of GAR sequences (again, each sequence is a solution to a task in the training set) which is itself close to a solution for some task due to the local nature of its search. In contrast, a global optimization procedure, such as gradient descent, would search among linear combinations which are far from any task solution. Finally, our results are consistent with empirical findings in the machine learning literature showing that additive schemes outperform gradient descent when searching for good linear combinations of features for novel classification tasks (Friedman, 2001; Perkins, Lacker, and Theiler, 2003; Viola and Jones, 2004).[4]

5.2 Visualizing Torque Sequences

The library of a GAR model is created on the basis of a wide variety of motor tasks. The torque sequences in the library should, therefore, be "representative" of the tasks they encode. Our goal here is to examine these sequences.

We trained a GAR model with 3000 training tasks using a library of size 100. We then ordered the sequences in the library by the percent of the model's total activation that a sequence accounted for. Figure 5 shows the Cartesian-space trajectories generated by the top three sequences. To generate these trajectories, the shoulder and elbow joint angles of the arm were initialized to $\pi/4$ and $\pi/2$ respectively. Each torque sequence was then applied to the arm, first with a coefficient of 1 and then with a coefficient of -1. Note that the trajectories span a wide range of directions. Several of the trajectories are highly curved, whereas others are closer to straight lines. This range is a result of the diverse set of tasks used to create the sequences. This graph illustrates that, even though the sequences are added to the library in an arbitrary order, the important sequences that remain in the library are representative of the motor tasks.

5.3 The GAR Model with Libraries of Different Sizes

Above we set the size of the library used by the GAR model to 100. Here we compare the model's performances with libraries of different sizes. If the size, denoted K, is too small, then torque sequences that are often useful for learning novel motor tasks might be removed. In contrast, if K is too big, then the library will contain many sequences which are nearly never used. Consequently, there ought to be an optimal value for K. We implemented the GAR model as described above

4. We thank an anonymous reviewer whose suggestions inspired these comments.

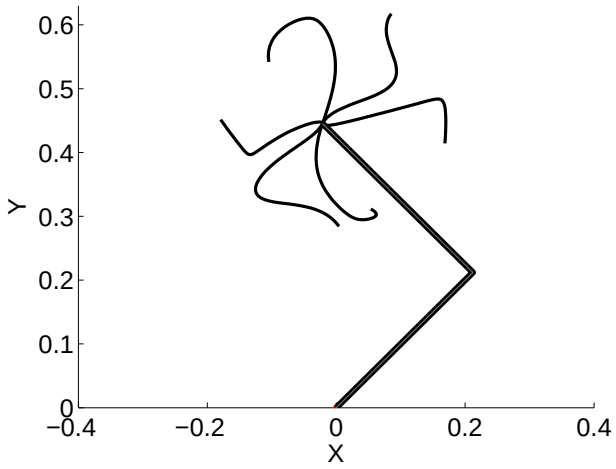

Figure 5: Cartesian-space trajectories generated by the three torque sequences that accounted for the largest percent of a GAR model's total activation. These trajectories were generated by initializing the shoulder and elbow joint angles of the arm to $\pi/4$ and $\pi/2$ respectively, and then applying the sequences to the arm with coefficients of 1 and -1.

using 3000 motor tasks. Three versions of the GAR model were used where the versions differed in the sizes of their libraries.

The results are shown in Figures 6 and 7. In Figure 6, the horizontal axis shows the number of motor tasks, and the vertical axis shows the percent of tasks in which a version of the GAR model needed to learn a torque sequence via feedback error learning. The latter value was obtained as follows. The motor tasks were divided into 60 blocks of 50 trials each. The percents for the blocks were then smoothed using a moving window of width 5. Results are reported for versions of the GAR model with library sizes of 50, 100, and 200. Early in training, a library has relatively few sequences, and feedback error learning must often be used. As training progresses, the library has many more useful sequences, and most novel motor tasks can be performed by linearly combining sequences from the library. In this case, feedback error learning is infrequently used. A comparison of the versions with different library sizes shows that the version with a library size of 50 used feedback error learning more often than versions with library sizes of 100 or 200. This suggests that a library size of 50 is too small.

From top to bottom, the three graphs in Figure 7 correspond to versions of the GAR model with library sizes of 50, 100, and 200. The sequences in a library are ordered according to the percent of a model's total activation that a sequence accounted for. The horizontal axis of each graph in Figure 7 plots the sequence number, and the vertical axis plots the percent of total activation that a sequence accounted for. The versions with libraries of size 100 and 200 show similar patterns of activation. In both cases, approximately the top 50 sequences accounted for nearly all the activation. The remaining sequences were rarely used. In contrast, the version with a library of size 50 had a different pattern of activation. Roughly all of the sequences in this library contributed to the model's total activation. We measured the average task error for each model (based on Equation 1) using

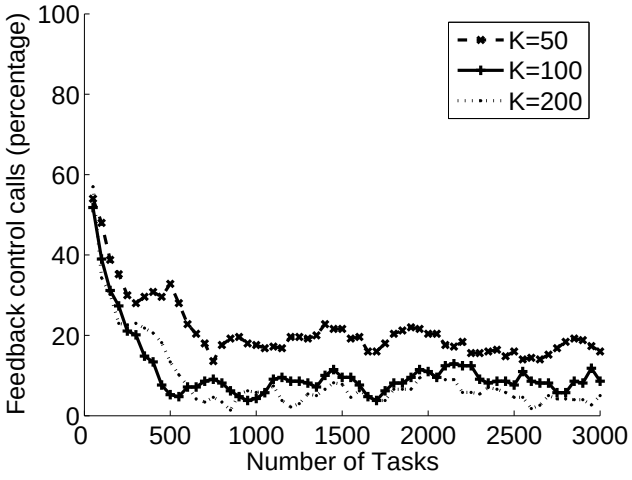

Figure 6: The horizontal axis shows the number of motor tasks, and the vertical axis shows the percent of tasks in which a version of the GAR model needed to learn a torque sequence via feedback error learning. The three curves in the figure correspond to versions of the GAR model with library sizes of 50, 100, and 200.

the last 1000 motor tasks. When $K = 50$, the average task error was 0.0925; when $K = 100$, the error was 0.0723; and when $K = 200$, the error was 0.0744. The corresponding standard error of the means were 0.010, 0.012, and 0.009. It seems that $K = 100$ is most efficient in the sense that it yielded good performance with a memory of moderate size. Furthermore, the version with $K = 100$ has the property that its use of sequences was relatively sparse. The top 10, 20, and 30 sequences accounted for 60, 78, and 88 percent of the version's total activation, respectively. Clearly, only a small fraction of the stored sequences tended to be used when learning a novel task.

5.4 GAR versus PCA in the Presence of Altered Dynamics

People are robust to changes in their arms' dynamics. For example, people can make accurate and smooth arm movements regardless of whether they carry no payload, a light payload, or a heavy payload. In this subsection, we compare the performances of the GAR and PCA models when they were trained without a payload, but a payload was added to the simulated arm during test trials.

The libraries for the GAR model (with a library of size 100) and the PCA model were created as described above with an arm that did not carry a payload. These models were then tested when the arm did carry a payload. Test trials were conducted as described above; that is, for each test task, a linear combination of torque sequences in a library was found via the additive regression algorithm for the GAR model, and via policy gradient for the PCA model. A set of 100 novel test tasks was generated. Models were evaluated on this set four times, once for each possible payload (payloads of 0, 1, 3, and 5 kg were used). Payloads were added to an arm by increasing the mass of the arm's elbow-wrist link (m_2 in Table 1). For the sake of completeness, we also tried the other two

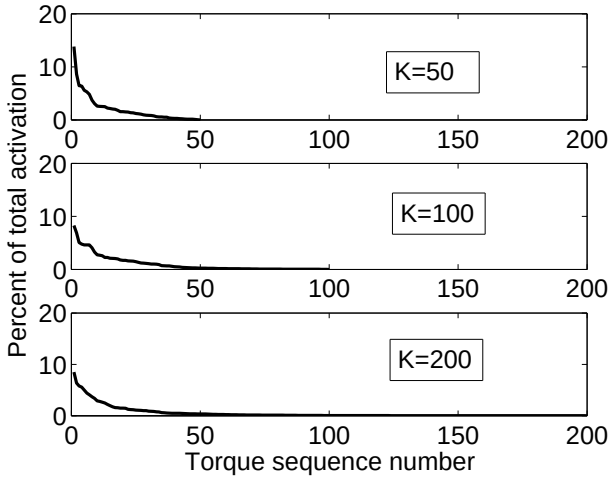

Figure 7: From top to bottom, the three graphs correspond to versions of the GAR model with library sizes of 50, 100, and 200. The sequences in a library are ordered according to the percent of a model's total activation that a sequence accounted for. The horizontal axis of each graph plots the sequence number, and the vertical axis plots the percent of total activation.

combinations of libraries with optimization algorithms, namely, the GAR sequences with policy gradient, and the PCA sequences with additive regression.

The results are shown in Figure 8. The vertical axis plots the average RMSE for each model for each payload. (The results for a payload of 0 are identical to those in Figure 3). For each payload, there are four bars corresponding to the four library/algorithm combinations. The performances of the PCA model degraded rapidly as the payload increased (2^{nd} bar in each set of bars). In contrast, the performances of the GAR model were robust (1^{st} bar in each set). We regard this successful generalization as a highly surprising result. It clearly demonstrates that the GAR model develops a useful library of torque sequences, and that the additive regression algorithm is a powerful optimization procedure for finding good linear combinations, even under test conditions that are very different from training conditions.

Why did the GAR model generalize so successfully? To address this question, we performed an additional analysis. The idea behind this analysis is to evaluate whether the GAR model generates similar libraries for different payloads. If this is the case, then additive regression should work well for tasks with novel payloads, even when using a library of GAR sequences constructed from zero-payload trials. We first generated a library of 100 GAR sequences using a training set of 3000 tasks where the simulated arm did not contain a payload. We then generated libraries for each non-zero payload using the same set of tasks. We compared each non-zero payload library to the zero-payload library. For each GAR sequence in a non-zero payload library, we found the sequence in the zero-payload library that was maximally correlated with this GAR sequence. For each non-zero payload, the average value of this maximum correlation is reported in Table 2. The GAR model successfully generalized from zero payloads to non-zero payloads because these correlations are large. The

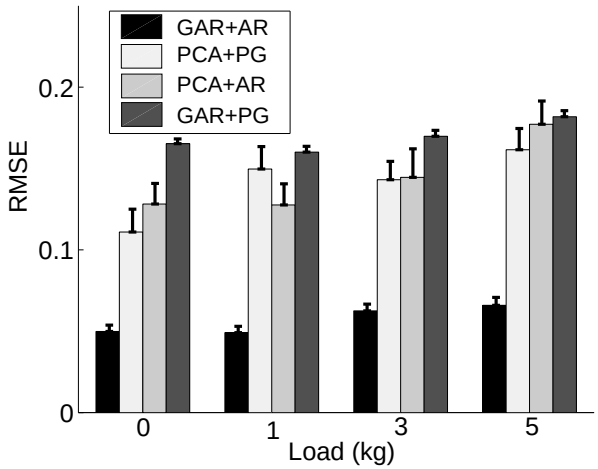

Figure 8: Average RMSEs of the GAR and PCA models on the test tasks when the arm carried different payloads.

Payload (kg)	Average maximum correlation	Standard error
1	0.84	0.08
3	0.81	0.04
5	0.73	0.06

Table 2: Average maximum correlation of the zero-payload library with the libraries built using non-zero payloads (see text for details).

other systems we evaluated were not able to take advantage of the similarities between solutions for zero-payload and non-zero payload tasks.

5.5 Motor Tasks Specified in Cartesian Space

In this subsection, we consider learning sequences for motor tasks when the desired trajectories are specified in Cartesian space instead of joint space. Using Cartesian trajectories adds an additional level of complexity. In addition to modeling the arm's inverse dynamics (a mapping from desired joint coordinates to torques), a system also needs to model the arm's inverse kinematics (a mapping from desired Cartesian coordinates to joint coordinates). An appealing feature of Cartesian trajectories is that they can be easily planned based on visuospatial information.

The cost function for this simulation is the sum of squared error between desired and actual positions of the arm's end-effector in Cartesian space:

$$J = \sum_{t=1}^{50} (r_x^*(t) - r_x(t))^2 + (r_y^*(t) - r_y(t))^2$$

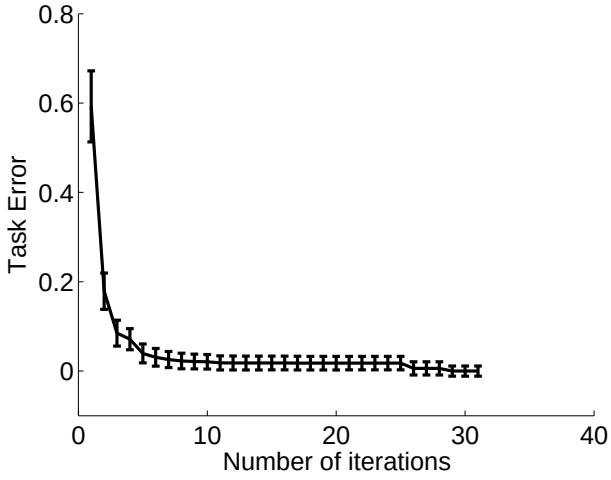

Figure 9: Results when motor tasks were specified in Cartesian space. The horizontal axis plots the number of iterations used by the GAR model, and the vertical axis show the average task error at each iteration.

where $(r_x^*(t), r_y^*(t))$ is the desired (x,y)-coordinates of the arm's end-effector in Cartesian space at time t, and $(r_x(t), r_y(t))$ is the actual coordinates. The GAR model was trained using 3000 motor tasks with a library of size 100. The library was constructed in the same way as before. An error threshold of $\varepsilon = 0.02$ was used to determine if a linear combination of torque sequences from the library provided a "good" aggregate sequence for a task (note that this is different from a threshold of $\varepsilon = 0.05$ used in previous simulations because the cost function is in different units now). We then created a test set of 100 tasks, and used the additive regression algorithm to learn a set of linear coefficients for each test task.

The results are shown in Figure 9. The horizontal axis plots the number of iterations used by the additive regression algorithm, and the vertical axis shows the average task error (Equation 8) at each iteration. Note that this error declined rapidly to a near-zero value. This outcome indicates that the GAR model has wide applicability in the sense that it is effective regardless of whether motor tasks are specified in joint space or Cartesian space.

6. GAR Model Applied to a Redundant and Unstable System

Until now, our simulations used a robotic arm. This section reports simulation results with a spring-mass system. In contrast to the robotic arm, this system allows us to evaluate different learners when the system to be controlled has linear dynamics, redundancy (three control signals move the system in a two-dimensional space), and is inherently unstable (zero or random control signals lead to divergent behavior). The system is schematically illustrated in Figure 10.

The spring-mass system has three elastic spring-like sticks that produce an opposing force when stretched or compressed. Stick 1 has resting length of l and connects the ground to point mass m_1. Stick 2 also has a resting length of l and connects m_1 to point mass m_2. Stick 3 has a resting length

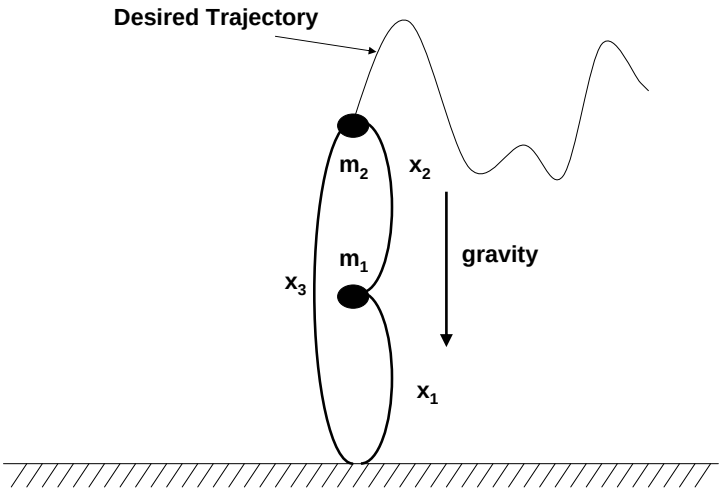

Figure 10: Schematic description of the spring-mass system used for the simulations.

of $2l$ and connects the ground to point mass m_2. All sticks have second-order linear dynamics. The dynamics of this system are governed by the following set of equations:

$$\begin{aligned} f_1 &= (l-x_1)k_1 + b_1\dot{x_1} + h_1u_1, \\ f_2 &= (l-x_2)k_2 + b_2\dot{x_2} + h_2u_2, \\ f_3 &= (2l-x_1)k_3 + b_3\dot{x_3} + h_3u_3, \\ x_3 &= x_1 + x_2, \\ \ddot{x_1} &= (f_1 - f_2 - mg)/m, \\ \ddot{x_2} &= (2f_2 + f_3 - f_1)/m \end{aligned}$$

where x_1, x_2, and x_3 are the current lengths of the sticks, f_1, f_2, and f_3 are the forces applied by the sticks due to elasticity, and the point masses m_1 and m_2 have the same weight m. Because the damping coefficients b_1, b_2, and b_3 were set to zero, the system exhibits a positive feedback effect which causes its behavior to diverge (this can be seen by setting the control signals u_1, u_2, and u_3 to zero). The constraint $x_3 = x_1 + x_2$ makes the system redundant as there are three inputs, u_1, u_2 and u_3, and only two free variables, x_1 and x_2. The parameter values are given in Table 3.

For this system, a task was defined by the desired trajectory for mass m_2 over a course of $T = 2.5$ seconds. A desired trajectory was generated as follows. First, four sine waves were generated with random frequencies ω_1, ω_2, ω_3, and ω_4, where ω_i was picked uniformly at random from the interval $[0.03, 0.3]$. The desired trajectory x_t^* was generated by $x_t^* = sin(\omega_1 t)sin(\omega_2 t) + sin(\omega_3 t)sin(\omega_4 t)$.

Constant	Value	Constant	Value	Constant	Value
k_1	20 kgm^2s^{-1}	k_2	20 kgm^2s^{-1}	k_3	40 kgm^2s^{-1}
b_1	0.0 kgm^2s^{-1}	b_2	0.0 kgm^2s^{-1}	b_3	0.0 kgm^2s^{-1}
h_1	10 $kgms^{-2}$	h_2	10 $kgms^{-2}$	h_3	20 $kgms^{-2}$
l	0.50 m	m	0.5 kg	g	10 ms^{-2}

Table 3: Values of constants used in the simulations of the spring-mass system.

Performance errors were quantified using a quadratic cost function:

$$c(x_3;x^*) = \int_0^T (x_t^* - x_{3,t})^2 dt$$

where $x_{3,t}$ is the position of mass m_2 at time t. Because this is a linear system with a quadratic cost function, the sequence of optimal (feedforward) control signals for a task can be computed using standard optimal control techniques. In our simulations, we discretized the system in time steps of size 0.025 seconds and integrated the system using a first-order Runge-Kutta method.

We created libraries of sequences as follows. We first generated a training set of 1000 tasks. For each task, we also computed the optimal sequence of control signals. Using these optimal sequences as data items, we created a library of PCA sequences by extracting the top thirty principal components based on these data items. We created a library of thirty GAR sequences using the additive regression procedure described above, with the exception that an optimal sequence (as opposed to a sequence found via feedback error learning) was added to the library when a good linear combination of library sequences could not be found.

As above, we compare the performances of four learning systems comprising all four combinations of representational primitives (PCA and GAR sequences) and optimization procedures (policy gradient and additive regression). The results on 100 test tasks are shown in Figure 11 (the leftmost bar in this figure gives the average RMSE using optimal control sequences). Note that all four learners performed nearly optimally. This is unsurprising as the quadratic error surface contains a single (global) minimum, and any reasonable optimization procedure will find this minimum. Also note that all four learners showed similar levels of performance (the differences in their performances are not statistically significant). This result is consistent with our predictions for linear systems with quadratic cost functions (see Section 5.1).

Although the learners showed similar levels of performance, a main point of this section is that they are not equivalent in terms of processing time. To quantify processing time, we examined the number of calls each learner made to the simulator of the spring-mass system. This simulator must be called each time a gradient is computed. On average, the learner using GAR sequences and additive regression made 49 calls, the learner using PCA sequences and policy gradient made 3879 calls, the learner using PCA sequences and additive regression made 62 calls, and the learner using GAR sequences and policy gradient made 3422 calls. Clearly, the additive regression algorithm is efficient in the sense that it made significantly fewer calls to the spring-mass simulator, irrespective of the library used.

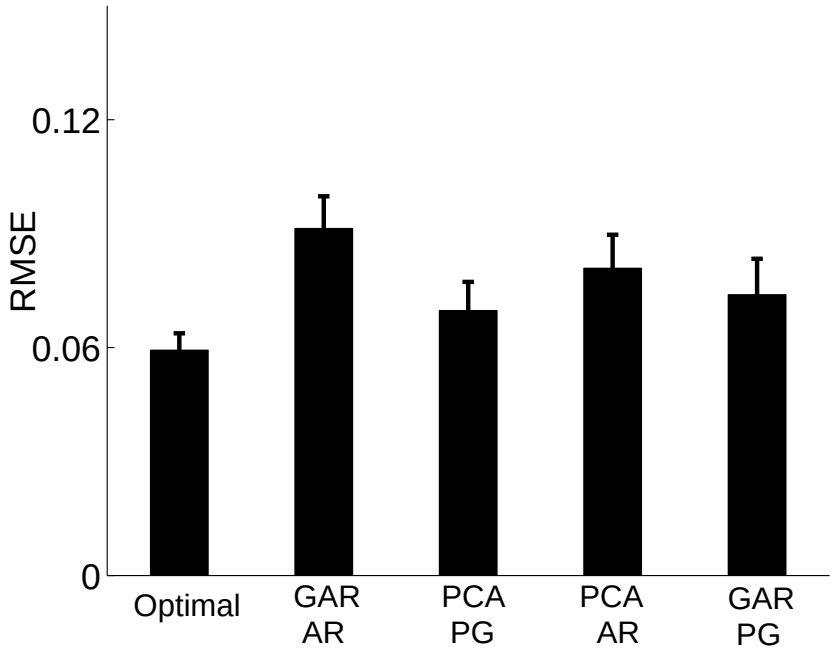

Figure 11: Results with the spring-mass system. The vertical axis shows the average RMSE on a test set with 100 tasks. The horizontal axis shows the learning system: (i) Optimal: optimal control signals; (ii) GAR+AR: GAR sequences with additive regression; (iii) PCA+PG: PCA sequences with policy gradient; (iv) PCA+AR: PCA sequences with additive regression; and (v) GAR+PG: GAR sequences with policy gradient.

7. Conclusions

In summary, the computational complexities arising in motor control can be ameliorated through the use of a library of motor synergies. We presented a new model, referred to as the Greedy Additive Regression (GAR) model, for learning a library of torque sequences, and for learning the coefficients of a linear combination of library sequences minimizing a cost function. Results using a simulated two-joint arm suggest that the GAR model consistently shows excellent performance in the sense that it rapidly learns to perform novel, complex motor tasks. Moreover, its library is overcomplete and sparse, meaning that only a small fraction of the stored torque sequences are used when learning a new movement. The library is also robust in the sense that, after an initial training period, nearly all novel movements can be learned as additive combinations of sequences in the library, and in the sense that it shows good generalization when an arm's dynamics are altered between training and test conditions, such as when a payload is added to the arm. Additionally, we showed that the GAR model works well regardless of whether motor tasks are specified in joint space or Cartesian space.

The GAR model appears to consistently outperform the PCA model, as described above. A comparison of these two models suggests why this is the case. The GAR model uses a library of local features—each sequence in its library is a solution to a single task from the training set—and a local optimization procedure, namely, additive regression. In contrast, the PCA model uses a library of global features—each item in its library reflects properties of all tasks in the training set—and policy gradient which is a global optimization procedure because it seeks good combinations of all items in its library. We conjecture that the local versus global nature of the GAR versus PCA models accounts for the performance advantages of the GAR model on nonlinear tasks. This account is consistent with other empirical findings in the machine learning literature (Friedman, 2001; Perkins, Lacker, and Theiler, 2003; Viola and Jones, 2004). Future work will need to provide a theoretical underpinning for this intuitive conjecture. The GAR and PCA models represent two ends of a local/global continuum. Future work should also study models that lie at intermediate points along this continuum, such as models that form linear combinations by adding a small number of features at each iteration, instead of the addition of a single feature as in the GAR model.

We have focused here on defining and evaluating the GAR model from a machine learning perspective. Future research will need to focus on the implications of the model for our understanding of motor control in biological organisms, the theoretical foundations of the model, and further empirical evaluations. In regard to our understanding of biological motor control, it would be interesting to know whether sets of motor synergies used by biological organisms are sparse and overcomplete as suggested by the GAR model, or are full-distributed and non-redundant as suggested by the PCA model. If they are sparse and overcomplete, then the computational advantages of the GAR model may help us understand why organisms have evolved or developed to use this type of representation. In regard to theoretical foundations, the engineering community is often reluctant to adopt new adaptive procedures for control unless these procedures have proven stability and performance guarantees. At the moment, no such guarantees exist for the GAR model. Future work will need to address these issues. In regard to empirical evaluations, future research will need to evaluate the GAR model with larger and more complex motor systems and motor tasks.

Acknowledgments

We thank two anonymous reviewers for their helpful comments on an earlier version of this manuscript. This work was supported by AFOSR research grant FA9550-06-1-0492.

References

C. G. Atkeson and D. J. Reinkensmeyer. Using associative content-addressable memories to control robots. In W. T. Miller III, R. S. Sutton, and P. J. Werbos, editors,, *Neural Networks for Control*. MIT Press, 1990.

C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11:75-113, 1997.

D. C. Bentivegna. *Learning from Observation Using Primitives*. Ph.D. dissertation, Georgia Institute of Technology, 2004.

N. Bernstein. *The Coordination and Regulation of Movements*. Pergamon Press, 1967.

P. Bühlmann. Boosting methods: Why they can be useful for high-dimensional data. In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC)*, 2003.

M. Chhabra and R. A. Jacobs. Properties of synergies arising from a theory of optimal motor behavior. *Neural Computation*, 18:2320-2342, 2006.

P. Corke. A robotics toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, 3:24-32, 1996.

A. d'Avella, P. Saltiel, and E. Bizzi. Combinations of muscle synergies in the construction of a natural motor behavior. *Nature Neuroscience*, 6:300-308, 2003.

A. Fod, M. J. Matarić, and O. C. Jenkins. Automated derivation of primitives for movement classification. *Autonomous Robots*, 12:39-54, 2002.

Y. Freund and R. E. Schapire. A decision- theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119-139, 1997.

J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189-1232, 2001.

J. M. Hollerbach and T. Flash. Dynamic interactions between limb segments during planar arm movement. *Biological Cybernetics*, 44:67-77, 1982.

A. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*. MIT Press, 2003.

O. C. Jenkins and M. J. Matarić. A spatio-temporal extension to Isomap nonlinear dimension reduction. In *Proceedings of the 21st International Conference on Machine Learning*, 2004.

M. Kawato, K. Furukawa, and R. Suzuki. Hierarchical neural-network model for control and learning of voluntary movement. *Biological Cybernetics*, 57:169-185, 1987.

M. Lau and J. J. Kuffner. Behavior planning for character animation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005.

J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics (SIGGRAPH)*, 21:491-500, 2002.

M. S. Lewicki and T. J. Sejnowski. Learning overcomplete representations. *Neural Computation*, 12:337-365, 2000.

W. Li and E. Todorov. Iterative linear-quadratic regulator design for nonlinear biological movement systems. In *Proceedings of the First International Conference on Informatics in Control, Automation, and Robotics*, 2004.

S. Mallat and Z. Zhang. Matching pursuit with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41:3397-3415, 1993.

T. W. Miller III, F. H. Glanz, and L. G. Kraft. Application of a general learning algorithm to the control of robotic manipulators. *International Journal of Robotic Research*, 6:84-98, 1987.

F. A. Mussa-Ivaldi, S. F. Giszter, and E. Bizzi. Linear combination of primitives in vertebrate motor control. *Proceedings of the National Academy of Sciences USA*, 91:7534-7538, 1994.

S. Perkins, K. Lacker, and J. Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. *Journal of Machine Learning Research*, 3:1333-1356, 2003.

J. Peters and S. Schaal. Reinforcement learning for parameterized motor primitives. In *Proceedings of the International Joint Conference on Neural Networks*, 2006.

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

A. Safanova, J. K. Hodgins, and N. S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics (SIGGRAPH)*, 23:514-521, 2004.

T. D. Sanger. Optimal unsupervised motor learning for dimensionality reduction of nonlinear control systems. *IEEE Transactions on Neural Networks*, 5:965-973, 1994.

T. D. Sanger. Optimal movement primitives. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*. MIT Press, 1995.

R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197-227, 1990.

E. C. Smith and M. S. Lewicki. Efficient auditory coding. *Nature*, 439:978-982, 2006.

M. Stolle and C. G. Atkeson. Policies based on trajectory libraries. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2006.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*. MIT Press, 1999.

Y. Tassa, T. Erez, and W. Smart. Receding horizon differential dynamic programming. In J. C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*. MIT Press, 2008.

K. A. Thoroughman and R. Shadmehr. Learning of action through adaptive combination of motor primitives. *Nature*, 407:742-747, 2000.

E. Todorov and Z. Ghahramani. Unsupervised learning of sensory-motor primitives. In *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2003.

E. Todorov and Z. Ghahramani. Analysis of the synergies underlying complex hand manipulation. In *Proceedings of the 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2004.

P. Viola and M. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57:137-154, 2004.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229-256, 1992.