*Cognitive Science*, *16*, 307-354, 1992.

# Forward models: Supervised learning with a distal teacher*

Michael I. Jordan
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology

David E. Rumelhart
Department of Psychology
Stanford University

**Abstract**

Internal models of the environment have an important role to play in adaptive systems in general and are of particular importance for the supervised learning paradigm. In this paper we demonstrate that certain classical problems associated with the notion of the "teacher" in supervised learning can be solved by judicious use of learned internal models as components of the adaptive system. In particular, we show how supervised learning algorithms can be utilized in cases in which an unknown dynamical system intervenes between actions and desired outcomes. Our approach applies to any supervised learning algorithm that is capable of learning in multi-layer networks.

*This paper is a revised version of MIT Center for Cognitive Science Occasional Paper #40. We wish to thank Michael Mozer, Andrew Barto, Robert Jacobs, Eric Loeb, and James McClelland for helpful comments on the manuscript. This project was supported in part by BRSG 2 S07 RR07047-23 awarded by the Biomedical Research Support Grant Program, Division of Research Resources, National Institutes of Health, by a grant from ATR Auditory and Visual Perception Research Laboratories, by a grant from Siemens Corporation, by a grant from the Human Frontier Science Program, and by grant N00014-90-J-1942 awarded by the Office of Naval Research.

Recent work on learning algorithms for connectionist networks has seen a progressive weakening of the assumptions made about the relationship between the learner and the environment. Classical supervised learning algorithms such as the perceptron (Rosenblatt, 1962) and the LMS algorithm (Widrow & Hoff, 1960) made two strong assumptions: (1) The output units are the only adaptive units in the network, and (2) there is a "teacher" that provides desired states for all of the output units. Early in the development of such algorithms it was recognized that more powerful supervised learning algorithms could be realized by weakening the first assumption and incorporating internal units that adaptively recode the input representation provided by the environment (Rosenblatt, 1962). The subsequent development of algorithms such as Boltzmann learning (Hinton & Sejnowski, 1986) and backpropagation (LeCun, 1985; Parker, 1985; Rumelhart, Hinton, & Williams, 1986; Werbos, 1974) have provided the means for training networks with adaptive nonlinear internal units. The second assumption has also been weakened—learning algorithms that require no explicit teacher have been developed (Becker & Hinton, 1989; Grossberg, 1987; Kohonen, 1982; Linsker, 1988; Rumelhart & Zipser, 1986). Such "unsupervised" learning algorithms generally perform some sort of clustering or feature extraction on the input data and are based on assumptions about the statistical or topological properties of the input ensemble.

In this paper we examine in some detail the notion of the "teacher" in the supervised learning paradigm. We argue that the teacher is less of a liability than has commonly been assumed and that the assumption that the environment provides desired states for the output of the network can be weakened significantly without abandoning the supervised learning paradigm altogether. Indeed, we feel that an appropriate interpretation of the role of the teacher is crucial in appreciating the range of problems to which the paradigm can be applied.

The issue that we wish to address is best illustrated by way of an example. Consider a skill-learning task such as that faced by a basketball player learning to shoot baskets. The problem for the learner is to find the appropriate muscle commands to propel the ball toward the goal. Different commands are appropriate for different locations of the goal in the visual scene; thus, a mapping from visual scenes to muscle commands is required. What learning algorithm might underly the acquisition of such a mapping? Clearly, clustering or feature extraction on the visual input is not sufficient. Moreover, it is difficult to see how to apply classical supervised algorithms to this problem, because there is no teacher to provide muscle commands as targets to the learner. The only target information provided to the learner is in terms of the outcome of the movement; that is, the sights and sounds of a ball passing through the goal.

The general scenario suggested by the example is shown in Figure 1. *Intentions* are provided as inputs to the learning system. The learner transforms intentions into *actions*, which are transformed by the environment into *outcomes*. Actions are *proximal* variables; that is, variables that the learner controls directly, while

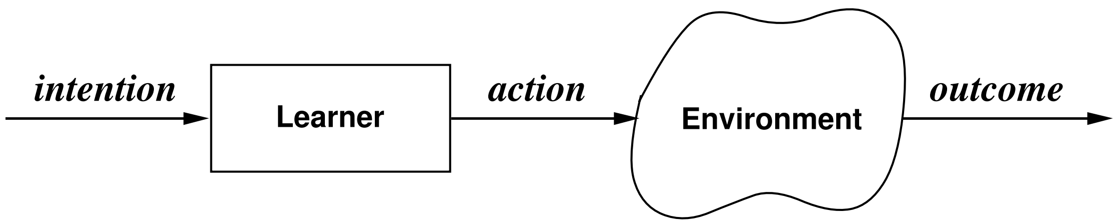


Figure 1: The distal supervised learning problem. Target values are available for the distal variables (the "outcomes") but not for the proximal variables (the "actions").

outcomes are *distal* variables, variables that the learner controls indirectly through the intermediary of the proximal variables. During the learning process, target values are assumed to be available for the distal variables but not for the proximal variables. Therefore, from a point of view outside the learning system, a "distal supervised learning task" is a mapping from intentions to desired outcomes. From the point of view of the learner, however, the problem is to find a mapping from intentions to actions that can be composed with the environment to yield desired distal outcomes. The learner must discover how to vary the components of the proximal action vector so as to minimize the components of the distal error.

The distal supervised learning problem also has a temporal component. In many environments the effects of actions are not punctate and instantaneous, but rather linger on and mix with the effects of other actions. Thus the outcome at any point in time is influenced by any of a number of previous actions. Even if there exists a set of variables that have a static relationship to desired outcomes, the learner often does not have direct control over those variables. Consider again the example of the basketball player. Although the flight of the ball depends only on the velocity of the arm at the moment of release–a static relationship–it is unlikely that the motor control system is able to control release velocity directly. Rather, the system outputs forces or torques, and these variables do not have a static relationship to the distal outcome.

In the remainder of the paper we describe a general approach to solving the distal supervised learning problem. The approach is based on the idea that supervised learning in its most general form is a two-phase procedure. In the first phase the learner forms a predictive internal model (a forward model) of the transformation from actions to distal outcomes. Because such transformations are often not known a priori, the internal model must generally be learned by exploring the outcomes associated with particular choices of actions. This auxiliary learning problem is itself a supervised learning problem, based on the error between internal, predicted outcomes and actual outcomes. Once the internal model has been at least partially learned, it can be used in an indirect manner to solve for the mapping from intentions to actions.

The idea of using an internal model to augment the capabilities of supervised learning algorithms has also been proposed by Werbos (1987), although his perspective differs in certain respects from our own. There have been a number of further developments of the idea (Kawato, 1990; Miyata, 1988; Munro, 1987; Nguyen & Widrow, 1989; Robinson & Fallside, 1989; Schmidhuber, 1990), based either on the work of Werbos or our own unpublished work (Jordan, 1983; Rumelhart, 1986). There are also close ties between our approach and techniques in optimal control theory (Kirk, 1970) and adaptive control theory (Goodwin & Sin, 1984; Narendra & Parthasarathy, 1990). We discuss several of these relationships in the remainder of the paper, although we do not attempt to be comprehensive.

## Distal supervised learning and forward models

This section and the following section present a general approach to solving distal supervised learning problems. We begin by describing our assumptions about the environment and the learner.

We assume that the environment can be characterized by a next-state function $f$ and an output function $g$. At time step $n-1$ the learner produces an *action* $\mathbf{u}[n-1]$. In conjunction with the state of the environment $\mathbf{x}[n-1]$ the action determines the next state $\mathbf{x}[n]$:

$$\mathbf{x}[n] = f(\mathbf{x}[n-1], \mathbf{u}[n-1]). \tag{1}$$

Corresponding to each state $\mathbf{x}[n]$ there is also a *sensation* $\mathbf{y}[n]$:

$$\mathbf{y}[n] = g(\mathbf{x}[n]). \tag{2}$$

(Note that sensations are output vectors in the current formalism—"outcomes" in the language of the introductory section). The next-state function and the output function together determine a state-dependent mapping from actions to sensations.

In the current paper we assume that the learner has access to the state of the environment; we do not address issues relating to state representation and state estimation. State representations might involve delayed values of previous actions and sensations (Ljung & Söderström, 1986), or they might involve internal state variables that are induced as part of the learning procedure (Mozer & Bachrach, 1990). Given the state $\mathbf{x}[n-1]$ and given the *input* $\mathbf{p}[n-1]$, the learner produces an action $\mathbf{u}[n-1]$:

$$\mathbf{u}[n-1] = h(\mathbf{x}[n-1], \mathbf{p}[n-1]).^1 \tag{3}$$

[1]The choice of time indices in Equations 1, 2, and 3 is based on our focus on the output at time $n$. In our framework a learning algorithm alters $\mathbf{y}[n]$ based on previous values of the states, inputs, and actions.

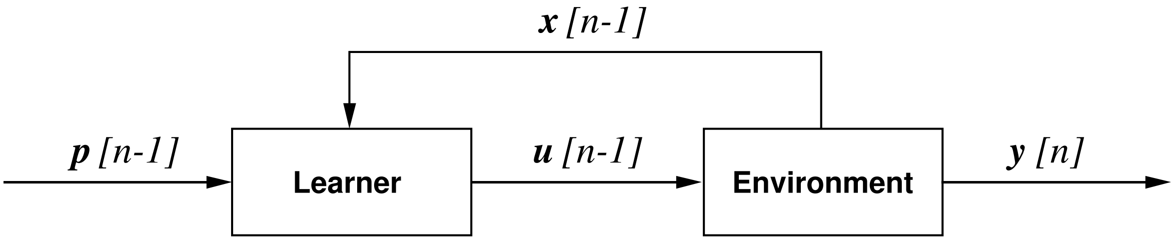


Figure 2: The composite performance system consisting of the learner and the environment. This system is a mapping from inputs $\mathbf{p}[n-1]$ to sensations $\mathbf{y}[n]$. The training data $\{\mathbf{p}_i[n-1], \mathbf{y}_i^*[n]\}$ specify desired input/output behavior across the composite system. Note that there is an implicit loop within the environment such that the output at time $n$ depends on the state at time $n-1$ (cf. Equation 1).

The goal of the learning procedure is to make appropriate adjustments to the input-to-action mapping $h$ based on data obtained from interacting with the environment.

A *distal supervised learning problem* is a set of training pairs $\{\mathbf{p}_i[n-1], \mathbf{y}_i^*[n]\}$, where $\mathbf{p}_i[n-1]$ are the input vectors and $\mathbf{y}_i^*[n]$ are the corresponding desired sensations. For example, in the basketball problem, the input might be a high-level intention of shooting a basket, and a desired sensation would be the corresponding visual representation of a successful outcome. Note that the distal supervised learning problem makes no mention of the actions that the learner must acquire; only inputs and desired sensations are specified. From a point of view outside the learning system the training data specify desired input/output behavior across the *composite performance system* consisting of the learner and the environment (see Figure 2). From the point of view of the learner, however, the problem is to find a mapping from inputs $\mathbf{p}[n-1]$ to actions $\mathbf{u}[n-1]$ such that the resulting distal sensations $\mathbf{y}[n]$ are the target values $\mathbf{y}^*[n]$. That is, the learner must find a mapping from inputs to actions that can be placed in series with the environment so as to yield the desired pairing of inputs and sensations. Note that there may be more than one action that yields a given desired sensation from any given state; that is, the distal supervised learning problem may be underdetermined. Thus, in the basketball example, there may be a variety of patterns of motor commands that yield the same desired sensation of seeing of the ball pass through the goal.

## Forward models

The learner is assumed to be able to observe states, actions, and sensations and can therefore model the mapping between actions and sensations. A *forward model* is an internal model that produces a predicted sensation $\hat{\mathbf{y}}[n]$ based on the state $\mathbf{x}[n-1]$ and the action $\mathbf{u}[n-1]$. That is, a forward model predicts the consequences of a

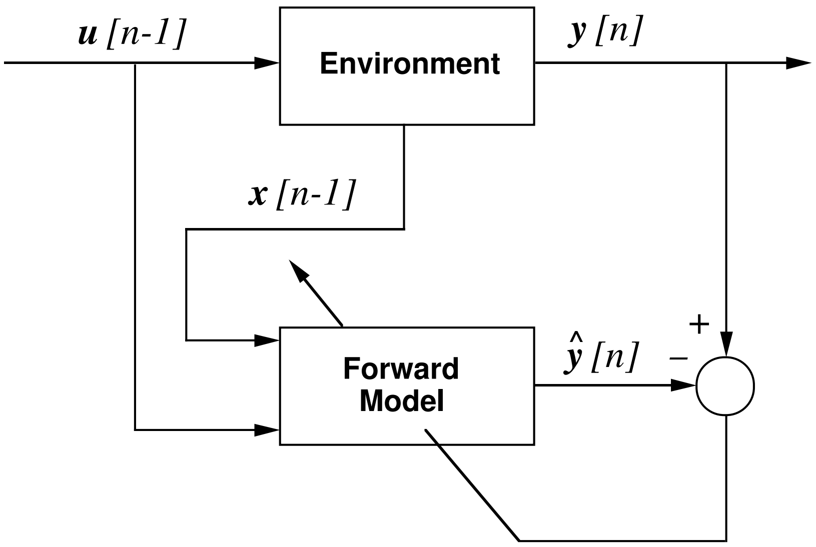


Figure 3: Learning the forward model using the prediction error $\mathbf{y}[n] - \hat{\mathbf{y}}[n]$.

given action in the context of a given state vector. As shown in Figure 3, the forward model can be learned by comparing predicted sensations to actual sensations and using the resulting *prediction error* to adjust the parameters of the model. Learning the forward model is a classical supervised learning problem in which the teacher provides target values directly in the output coordinate system of the learner.[2]

## Distal supervised learning

We now describe a general approach to solving the distal supervised learning problem. Consider the system shown in Figure 4, in which the learner is placed in series with a forward model of the environment. This *composite learning system* is a state-dependent mapping from inputs to predicted sensations. Suppose that the forward model has been trained previously and is a perfect model of the environment; that is, the predicted sensation equals the actual sensation for all actions and all states.

[2]In the engineering literature, this learning process is referred to as "system identification" (Ljung & Söderström, 1986).

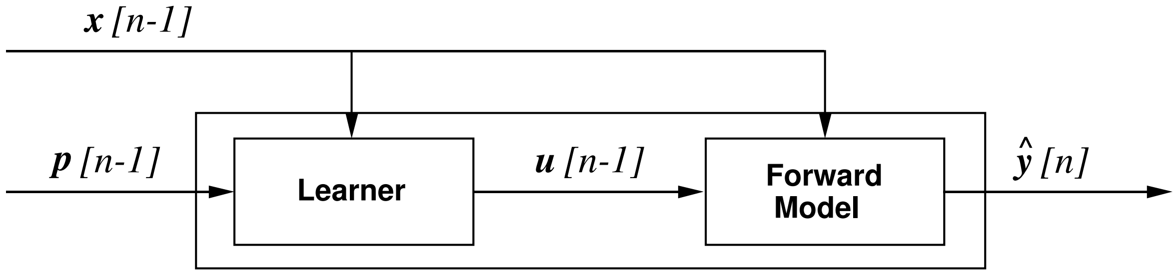


Figure 4: The composite learning system. This composite system maps from inputs $\mathbf{p}[n-1]$ to predicted sensations $\hat{\mathbf{y}}[n]$ in the context of a given state vector.

We now treat the composite learning system as a single supervised learning system and train it to map from inputs to desired sensations according to the data in the training set. That is, the desired sensations $\mathbf{y}_i^*$ are treated as targets for the composite system. Any supervised learning algorithm can be used for this training process; however, the algorithm must be constrained so that it does not alter the forward model while the composite system is being trained. By fixing the forward model, we require the system to find an optimal composite mapping by varying only the mapping from inputs to actions. If the forward model is perfect, and if the learning algorithm finds the globally optimal solution, then the resulting (state-dependent) input-to-action mapping must also be perfect in the sense that it yields the desired composite input/output behavior when placed in series with the environment.

Consider now the case of an imperfect forward model. Clearly an imperfect forward model will yield an imperfect input-to-action map if the composite system is trained in the obvious way, using the difference between the desired sensation and the predicted sensation as the error term. This difference, the *predicted performance error* $(\mathbf{y}^* - \hat{\mathbf{y}})$, is readily available at the output of the composite system, but it is an unreliable guide to the true performance of the learner. Suppose instead that we ignore the output of the composite system and substitute the *performance error* $(\mathbf{y}^* - \mathbf{y})$ as the error term for training the composite system (see Figure 5). If the performance error goes to zero the system has found a correct input-to-action map, regardless of the inaccuracy of the forward model. The inaccuracy in the forward model manifests itself as a bias during the learning process, but need not prevent the performance error from going to zero. Consider, for example, algorithms based on steepest descent. If the forward model is not too inaccurate the system can still move downhill and thereby reach the solution region, even though the movement is not in the direction of steepest descent.

To summarize, we propose to solve the distal supervised learning problem by training a composite learning system consisting of the learner and a forward model of the environment. This procedure solves implicitly for an input-to-action map by

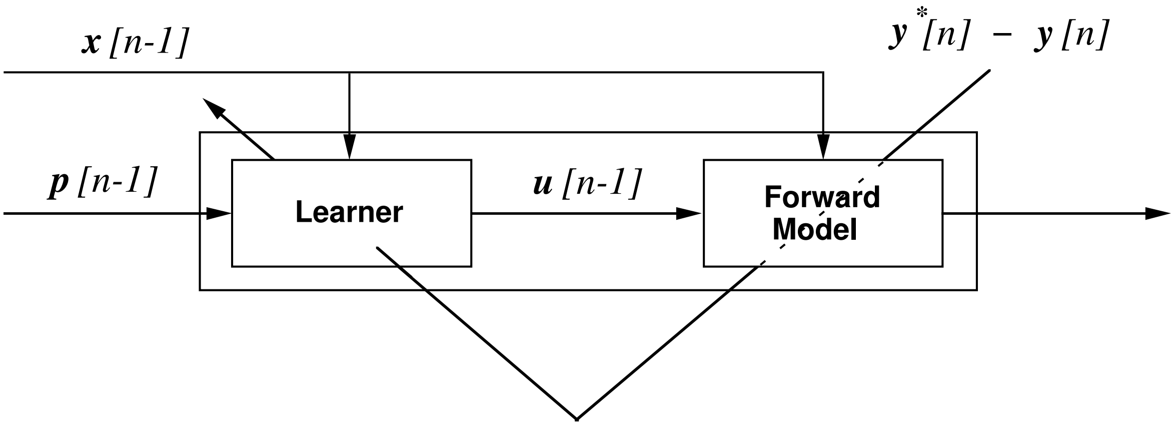


Figure 5: The composite system is trained using the performance error. The forward model is held fixed while the composite system is being trained.

training the composite system to map from inputs to distal targets. The training of the forward model must precede the training of the composite system, but the forward model need not be perfect, nor need it be pre-trained throughout all of state space. The ability of the system to utilize an inaccurate forward model is important; it implies that it may be possible to interleave the training of the forward model and the composite system.

In the remainder of the paper, we discuss the issues of interleaved training, inaccuracy in the forward model, and the choice of the error term in more detail. We first turn to an interesting special case of the general distal supervised learning problem—that of learning an inverse model of the environment.

## Inverse models

An *inverse model* is an internal model that produces an action $\mathbf{u}[n-1]$ as a function of the current state $\mathbf{x}[n-1]$ and the desired sensation $\mathbf{y}^*[n]$. Inverse models are defined by the condition that they yield the identity mapping when placed in series with the environment.

Inverse models are important in a variety of domains. For example, if the environment is viewed as a communications channel over which a message is to be transmitted, then it may be desirable to undo the distorting effects of the environment by placing it in series with an inverse model (Carlson, 1986). A second example, shown in Figure 6, arises in control system design. A controller receives the desired sensation $\mathbf{y}^*[n]$ as input and must find actions that cause actual sensations to be as close as possible to desired sensations; that is, the controller must invert

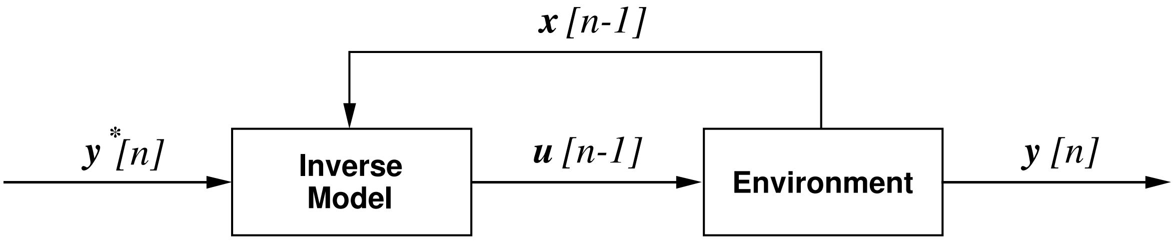


Figure 6: An inverse model as a controller.

the transformation from actions to sensations.[3] One approach to achieving this objective is to utilize an explicit inverse model of the environment as a controller.

Whereas forward models are uniquely determined by the environment, inverse models are generally not. If the environment is characterized by a many-to-one mapping from actions to sensations then there are generally an infinite number of possible inverse models. It is also worth noting that inverses do not always exist—it is not always possible to achieve a particular desired sensation from any given state. As we shall discuss, these issues of existence and uniqueness have important implications for the problem of learning an inverse model.

There are two general approaches to learning inverse models using supervised learning algorithms: the distal learning approach presented above and an alternative approach that we refer to as "direct inverse modeling" (cf. Jordan & Rosenbaum, 1989). We begin by describing the latter approach.

### Direct inverse modeling

Direct inverse modeling treats the problem of learning an inverse model as a classical supervised learning problem (Widrow & Stearns, 1985). As shown in Figure 7, the idea is to observe the input/output behavior of the environment and to train an inverse model directly by reversing the roles of the inputs and outputs. Data are provided to the algorithm by sampling in action space and observing the results in sensation space.

Although direct inverse modeling has been shown to be a viable technique in a number of domains (Atkeson & Reinkensmeyer, 1988; Kuperstein, 1988; Miller, 1987), it has two drawbacks that limit its usefulness. First, if the environment is characterized by a many-to-one mapping from actions to sensations, then the direct inverse modeling technique may be unable to find an inverse. The difficulty is that nonlinear many-to-one mappings can yield nonconvex inverse images, which are

[3]Control system design normally involves a number of additional constraints involving stability and robustness; thus, the goal is generally to invert the environment as nearly as possible subject to these additional constraints.

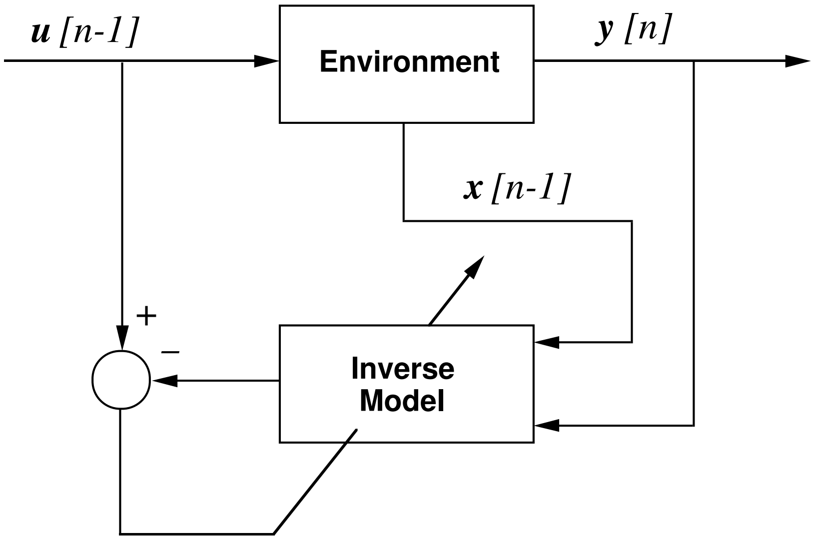


Figure 7: The direct inverse modeling approach to learning an inverse model.

problematic for direct inverse modeling.[4] Consider the situation shown in Figure 8. The nonconvex region on the left is the inverse image of a point in sensation space. Suppose that the points labelled by X's are sampled during the learning process. Three of these points correspond to the same sensation; thus, the training data as seen by the direct inverse modeling procedure are one-to-many—one input is paired with many targets. Supervised learning algorithms resolve one-to-many inconsistencies by averaging across the multiple targets (the form of the averaging depends on the particular cost function that is used). As is shown in the figure, however, the average of points lying in a nonconvex set does not necessarily lie in the set. Thus the globally optimal (minimum-cost) solution found by the direct inverse modeling approach is not necessarily a correct inverse model. (We present an example of such behavior in a following section).

The second drawback with direct inverse modeling is that it is not "goal-directed." The algorithm samples in action space without regard to particular targets or errors in sensation space. That is, there is no direct way to find an action that corresponds to a particular desired sensation. To obtain particular solutions the learner must sample over a sufficiently wide range of actions and rely on interpolation.

Finally, it is also important to emphasize that direct inverse modeling is restricted to the learning of inverse models—it is *not* applicable to the general distal

[4]A set is *convex* if for every pair of points in the set all points on the line between the points also lie in the set.

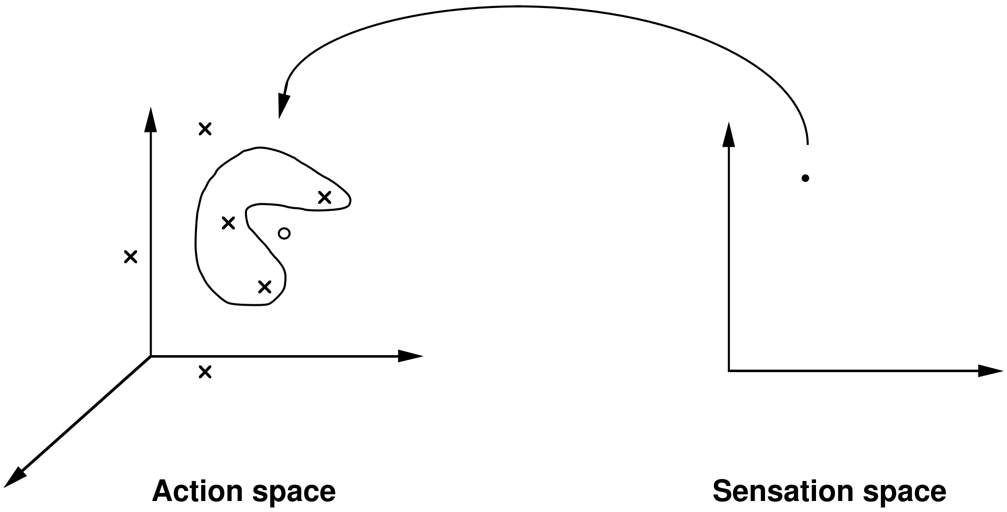


Figure 8: The convexity problem. The region on the left is the inverse image of the point on the right. The arrow represents the direction in which the mapping is learned by direct inverse modeling. The three points lying inside the inverse image are averaged by the learning procedure, yielding the vector represented by the small circle. This point is not a solution, because the inverse image is not convex.

supervised learning problem.

### The distal learning approach to learning an inverse model

The methods described earlier in this section are directly applicable to the problem of learning an inverse model. The problem of learning an inverse model can be treated as a special case of the distal supervised learning problem in which the input vector and the desired sensation are the same (that is, $\mathbf{p}[n-1]$ is equal to $\mathbf{y}^*[n]$ in Equation 3). Thus, an inverse model is learned by placing the learner and the forward model in series and learning an identity mapping across the composite system.[5]

A fundamental difference between the distal learning approach and direct inverse modeling approach is that rather than averaging over regions in action space, the distal learning approach finds particular solutions in action space. The globally optimal solution for distal learning is a set of vectors $\{\mathbf{u}_i\}$ such that the performance

[5]An interesting analogy can be drawn between the distal learning approach and indirect techniques for solving systems of linear equations. In numerical linear algebra, rather than solving explicitly for a generalized inverse of the coefficient matrix, solutions are generally found indirectly (e.g., by applying Gaussian elimination to both sides of the equation $GA = I$, where $I$ is the identity matrix).

errors $\{\mathbf{y}_i^* - \mathbf{y}_i\}$ are zero. This is true irrespective of the shapes of the inverse images of the targets $\mathbf{y}_i^*$. Vectors lying outside of an inverse image, such as the average vector shown in Figure 8, do not yield zero performance error and are therefore not globally optimal. Thus nonconvex inverse images do not present the same fundamental difficulties for the distal learning framework as they do for direct inverse modeling.

It is also true that the distal learning approach is fundamentally goal-directed. The system works to minimize the performance error; thus, it works directly to find solutions that correspond to the particular goals at hand.

In cases in which the forward mapping is many-to-one, the distal learning procedure finds a particular inverse model. Without additional information about the particular structure of the input-to-action mapping there is no way of predicting which of the possibly infinite set of inverse models the procedure will find. As is discussed below, however, the procedure can also be constrained to find particular inverse models with certain desired properties.

## Distal learning and backpropagation

In this section we describe an implementation of the distal learning approach that utilizes the machinery of the backpropagation algorithm. It is important to emphasize at the outset, however, that backpropagation is not the only algorithm that can be used to implement the distal learning approach. Any supervised learning algorithm can be used as long as it is capable of learning a mapping across a composite network that includes a previously trained subnetwork; in particular, Boltzmann learning is applicable (Jordan, 1983).

We begin by introducing a useful shorthand for describing backpropagation in layered networks. A layered network can be described as a parameterized mapping from an input vector $\mathbf{x}$ to an output vector $\mathbf{y}$:

$$\mathbf{y} = \phi(\mathbf{x}, \mathbf{w}), \tag{4}$$

where $\mathbf{w}$ is a vector of parameters (weights). In the classical paradigm, the procedure for changing the weights is based on the discrepancy between a target vector $\mathbf{y}^*$ and the actual output vector $\mathbf{y}$. The magnitude of this discrepancy is measured by a cost functional of the form:

$$J = \frac{1}{2}(\mathbf{y}^* - \mathbf{y})^T(\mathbf{y}^* - \mathbf{y}). \tag{5}$$

($J$ is the sum of squared error at the output units of the network). It is generally desired to minimize this cost.

Backpropagation is an algorithm for computing gradients of the cost functional. The details of the algorithm can be found elsewhere (e.g., Rumelhart, et al., 1986); our intention here is to develop a simple notation that hides the details. This is achieved formally by using the chain rule to differentiate $J$ with respect to the weight vector $\mathbf{w}$:

$$\nabla_{\mathbf{w}} J = -\frac{\partial \mathbf{y}}{\partial \mathbf{w}}^T (\mathbf{y}^* - \mathbf{y}). \tag{6}$$

This equation shows that any algorithm that computes the gradient of $J$ effectively multiplies the error vector $\mathbf{y}^* - \mathbf{y}$ by the transpose Jacobian matrix $(\partial \mathbf{y}/\partial \mathbf{w})^T$.[6] Although the backpropagation algorithm never forms this matrix explicitly (backpropagation is essentially a factorization of the matrix; Jordan, 1988), Equation 6 nonetheless describes the results of the computation performed by backpropagation.[7]

Backpropagation also computes the gradient of the cost functional with respect to the activations of the units in the network. In particular, the cost functional $J$ can be differentiated with respect to the activations of the input units to yield:

$$\nabla_{\mathbf{x}} J = -\frac{\partial \mathbf{y}}{\partial \mathbf{x}}^T (\mathbf{y}^* - \mathbf{y}). \tag{7}$$

We refer to Equation 6 as "backpropagation-to-weights" and Equation 7 as "backpropagation-to-activation." Both computations are carried out in one pass of the algorithm; indeed, backpropagation-to-activation is needed as an intermediate step in the backpropagation-to-weights computation.

In the remainder of this section we formulate two broad categories of learning problems that lie within the scope of the distal learning approach and derive expressions for the gradients that arise. For simplicity it is assumed in both of these derivations that the task is to learn an inverse model (that is, the inputs and the distal targets are assumed to be identical). The two formulations of the distal learning framework focus on different aspects of the distal learning problem and have different strengths and weaknesses. The first approach, the "local optimization" formulation, focuses on the local dynamical structure of the environment. Because it assumes that the learner is able to predict state transitions based on information that is available locally in time, it depends on prior knowledge of an adequate set of state variables for describing the environment. It is most naturally applied to problems in which target values are provided at each moment in time, although it can be extended to problems in which target values are provided intermittently (as we demonstrate in a following section). All of the computations needed for the local

[6] The Jacobian matrix of a vector function is simply its first derivative—it is a matrix of first partial derivatives. That is, the entries of the matrix $(\partial \mathbf{y}/\partial \mathbf{w})$ are the partial derivatives of the each of the output activations with respect to each of the weights in the network.

[7] To gain some insight into why a transpose matrix arises in backpropagation, consider a single-layer linear network described by $\mathbf{y} = W\mathbf{x}$, where $W$ is the weight matrix. The rows of $W$ are the incoming weight vectors for the output units of the network, and the columns of $W$ are the outgoing weight vectors for the input units of the network. Passing a vector forward in the network involves taking the inner product of the vector with each of the incoming weight vectors. This operation corresponds to multiplication by $W$. Passing a vector backward in the network corresponds to taking the inner product of the vector with each of the outgoing weight vectors. This operation corresponds to multiplication by $W^T$, because the rows of $W^T$ are the columns of $W$.

optimization formulation can be performed in feedforward networks, thus there is no problem with stability. The second approach, the "optimization-along-trajectories" formulation, focuses on global temporal dependencies along particular target trajectories. The computation needed to obtain these dependencies is more complex than the computation needed for the local optimization formulation, but it is more flexible. It can be extended to cases in which a set of state variables is not known a priori and it is naturally applied to problems in which target values are provided intermittently in time. There is potentially a problem with stability, however, because the computations for obtaining the gradient involve a dynamical process.

## Local optimization

The first problem formulation that we discuss is a local optimization problem. We assume that the process that generates target vectors is stationary and consider the following general cost functional:

$$J = \frac{1}{2}E\{(\mathbf{y}^* - \mathbf{y})^T(\mathbf{y}^* - \mathbf{y})\}, \tag{8}$$

where $\mathbf{y}$ is an unknown function of the state $\mathbf{x}$ and the action $\mathbf{u}$. The action $\mathbf{u}$ is the output of a parameterized inverse model of the form:

$$\mathbf{u} = h(\mathbf{x}, \mathbf{y}^*, \mathbf{w}),$$

where $\mathbf{w}$ is a weight vector.

Rather than optimizing $J$ directly, by collecting statistics over the ensemble of states and actions, we utilize an online learning rule (cf. Widrow & Stearns, 1985) that makes incremental changes to the weights based on the instantaneous value of the cost functional:

$$J_n = \frac{1}{2}(\mathbf{y}^*[n] - \mathbf{y}[n])^T(\mathbf{y}^*[n] - \mathbf{y}[n]). \tag{9}$$

An online learning algorithm changes the weights at each time step based on the stochastic gradient of $J$; that is, the gradient of $J_n$:

$$\mathbf{w}[n+1] = \mathbf{w}[n] - \eta \nabla_{\mathbf{w}} J_n,$$

where $\eta$ is a step size. To compute this gradient the chain rule is applied to Equation 9:

$$\nabla_{\mathbf{w}} J_n = -\frac{\partial \mathbf{u}}{\partial \mathbf{w}}^T \frac{\partial \mathbf{y}}{\partial \mathbf{u}}^T (\mathbf{y}^*[n] - \mathbf{y}[n]), \tag{10}$$

where the Jacobian matrices $(\partial \mathbf{y}/\partial \mathbf{u})$ and $(\partial \mathbf{u}/\partial \mathbf{w})$ are evaluated at time $n-1$. The first and the third factors in this expression are easily computed: The first factor describes the propagation of derivatives from the output units of the inverse model (the "action units") to the weights of the inverse model, and the third factor is the distal error. The origin of the second factor is problematic, however, because the dependence of $\mathbf{y}$ on $\mathbf{u}$ is assumed to be unknown a priori. Our approach to obtaining an estimate of this factor has two parts: First, the system acquires a parameterized forward model over an appropriate subdomain of the state space. This model is of the form:

$$\hat{\mathbf{y}} = \hat{f}(\mathbf{x}, \mathbf{u}, \mathbf{v}), \tag{11}$$

where $\mathbf{v}$ is a vector of weights and $\hat{\mathbf{y}}$ is the predicted sensation. Second, the distal error is propagated backward through the forward model; this effectively multiplies the distal error by an estimate of the transpose Jacobian matrix $(\partial \mathbf{y}/\partial \mathbf{u})$.

Putting these pieces together, the algorithm for learning the inverse model is based on the following estimated stochastic gradient:

$$\hat{\nabla}_{\mathbf{w}} J_n = -\frac{\partial \mathbf{u}}{\partial \mathbf{w}}^T \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{u}}^T (\mathbf{y}^*[n] - \mathbf{y}[n]). \tag{12}$$

This expression describes the propagation of the distal error $(\mathbf{y}^*[n] - \mathbf{y}[n])$ backward through the forward model and down into the inverse model where the weights are changed.[8] The network architecture in which these computations take place is shown in Figure 9. This network is a straightforward realization of the block diagram in Figure 5. It is composed of an inverse model, which links the state units and the input units to the action units, and a forward model, which links the state units and the action units to the predicted-sensation units.

### Learning the forward model

The learning of the forward model can itself be formulated as an optimization problem, based on the following cost functional:

$$L = \frac{1}{2} E\{(\mathbf{y} - \hat{\mathbf{y}})^T (\mathbf{y} - \hat{\mathbf{y}})\},$$

where $\hat{\mathbf{y}}$ is of the form given in Equation 11. Although the choice of procedure for finding a set of weights $\mathbf{v}$ to minimize this cost is entirely independent of the choice of procedure for optimizing $J$ in Equation 8, it is convenient to base the learning of the forward model on a stochastic gradient as before:

$$\nabla_{\mathbf{v}} L_n = -\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{v}}^T (\mathbf{y}[n] - \hat{\mathbf{y}}[n]), \tag{13}$$

where the Jacobian matrix $(\partial \hat{\mathbf{y}}/\partial \mathbf{v})$ is evaluated at time $n-1$. This gradient can be computed by the propagation of derivatives within the forward model and therefore requires no additional hardware beyond that already required for learning the inverse model.

[8] Note that the error term $(\mathbf{y}^*[n] - \mathbf{y}[n])$ is not a function of the output of the forward model; nonetheless, activation must flow forward in the model because the estimated Jacobian matrix $(\partial \hat{\mathbf{y}}/\partial \mathbf{u})$ varies as a function of the activations of the hidden units and the output units of the model.

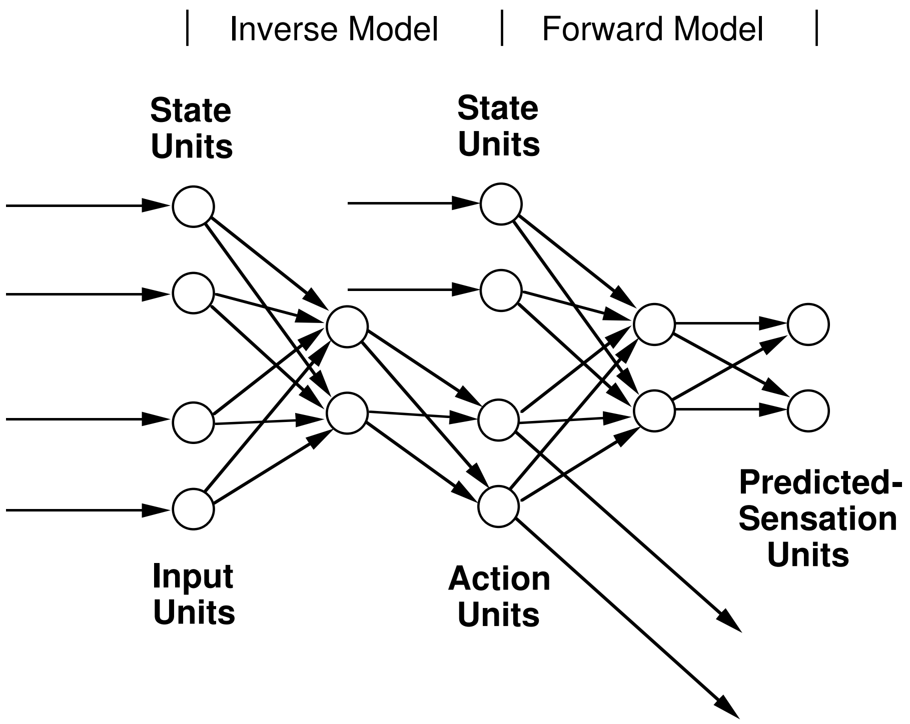


Figure 9: A feedforward network that includes a forward model. The action units are the output units of the system.

| | Name | Source |
|---|---|---|
| $\mathbf{y}^* - \mathbf{y}$ | performance error | environment, environment |
| $\mathbf{y} - \hat{\mathbf{y}}$ | prediction error | environment, model |
| $\mathbf{y}^* - \hat{\mathbf{y}}$ | predicted performance error | environment, model |

Table 1: The error signals and their sources

**The error signals**

It is important to clarify the meanings of the error signals used in Equations 12 and 13. As shown in Table 1, there are three error signals that can be formed from the variables $\mathbf{y}$, $\hat{\mathbf{y}}$, and $\mathbf{y}^*$—the *prediction error* $\mathbf{y} - \hat{\mathbf{y}}$, the *performance error* $\mathbf{y}^* - \mathbf{y}$, and the *predicted performance error* $\mathbf{y}^* - \hat{\mathbf{y}}$. All three of these error signals are available to the learner because each of the signals $\mathbf{y}^*$, $\mathbf{y}$ and $\hat{\mathbf{y}}$ are available individually—the target $\mathbf{y}^*$ and the actual sensation $\mathbf{y}$ are provided by the environment, whereas the predicted sensation $\hat{\mathbf{y}}$ is available internally.

For learning the forward model, the prediction error is clearly the appropriate error signal. The learning of the inverse model, however, can be based on either the performance error or the predicted performance error. Using the performance error (see Equation 12) has the advantage that the system can learn an exact inverse model even though the forward model is only approximate. There are two reasons for this: first, Equation 12 preserves the minima of the cost functional in Equation 9—they are zeros of the estimated gradient. That is, an inaccurate Jacobian matrix cannot remove zeros of the estimated gradient (points at which $\mathbf{y}^* - \mathbf{y}$ is zero), although it can introduce additional zeros (spurious local minima). Second, if the estimated gradients obtained with the approximate forward model have positive inner product with the stochastic gradient in Equation 10, then the expected step of the algorithm is downhill in the cost. Thus the algorithm can in principle find an exact inverse model even though the forward model is only approximate.

There may also be advantages to using the predicted performance error. In particular, it may be easier in some situations to obtain learning trials using the internal model rather than the external environment (Rumelhart, Smolensky, McClelland, & Hinton, 1986; Sutton, 1990). Such internal trials can be thought of as a form of "mental practice" (in the case of backpropagation-to-weights) or "planning" (in the case of backpropagation-to-activation). These procedures lead to improved performance if the forward model is sufficiently accurate. (Exact solutions cannot be found with such procedures, however, unless the forward model is exact).

### Modularity

In many cases the unknown mapping from actions to sensations can be decomposed into a series of simpler mappings, each of which can be modeled independently. For example, it may often be preferable to model the next-state function and the output function separately rather than modeling them as a single composite function. In such cases, the Jacobian matrix $(\partial\hat{\mathbf{y}}/\partial\mathbf{u})$ can be factored using the chain rule to yield the following estimated stochastic gradient:

$$\hat{\nabla}_{\mathbf{w}} J_n = -\frac{\partial \mathbf{u}}{\partial \mathbf{w}}^T \frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{u}}^T \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{x}}^T (\mathbf{y}^*[n] - \mathbf{y}[n]). \tag{14}$$

The estimated Jacobian matrices in this expression are obtained by propagating derivatives backward through the corresponding forward models, each of which are learned separately.

## Optimization along trajectories[9]

A complete inverse model allows the learner to synthesize the actions that are needed to follow any desired trajectory. In the local optimization formulation we effectively

[9]This section is included for completeness and is not needed for the remainder of the paper.

assume that the learning of an inverse model is of primary concern and the learning of particular target trajectories is secondary. The learning rule given by Equation 12 finds actions that invert the dynamics of the environment at the current point in state space, regardless of whether that point is on a desired trajectory or not. In terms of network architectures, this approach leads to using feedforward networks to model the local forward and inverse state transition structure (see Figure 9).

In the current section we consider a more specialized problem formulation in which the focus is on particular classes of target trajectories. This formulation is based on variational calculus and is closely allied with methods in optimal control theory (Kirk, 1970; LeCun, 1987). The algorithm that results is a form of "backpropagation-in-time" (Rumelhart, Hinton, & Williams, 1986) in a recurrent network that incorporates a learned forward model. The algorithm differs from the algorithm presented above in that it not only inverts the relationship between actions and sensations at the current point in state space but also moves the current state toward the desired trajectory.

We consider an ensemble of target trajectories $\{\mathbf{y}^*_\alpha[n]\}$ and define the following cost functional:

$$J = \frac{1}{2}E\{\sum_{n=1}^{N_\alpha}(\mathbf{y}^*_\alpha[n] - \mathbf{y}_\alpha[n])^T(\mathbf{y}^*_\alpha[n] - \mathbf{y}_\alpha[n])\}, \tag{15}$$

where $\alpha$ is an index across target trajectories and $\mathbf{y}_\alpha$ is an unknown function of the state $\mathbf{x}_\alpha$ and the action $\mathbf{u}_\alpha$. The action $\mathbf{u}_\alpha$ is a parameterized function of the state $\mathbf{x}_\alpha$ and the target $\mathbf{y}^*_\alpha$:

$$\mathbf{u}_\alpha = h(\mathbf{x}_\alpha, \mathbf{y}^*_\alpha, \mathbf{w}).$$

As in the previous formulation, we base the learning rule on the stochastic gradient of $J$, that is, the gradient evaluated along a particular sample trajectory $\mathbf{y}_\alpha$:

$$J_\alpha = \frac{1}{2}\sum_{n=1}^{N_\alpha}(\mathbf{y}^*_\alpha[n] - \mathbf{y}_\alpha[n])^T(\mathbf{y}^*_\alpha[n] - \mathbf{y}_\alpha[n]). \tag{16}$$

The gradient of this cost functional can be obtained using the calculus of variations (see also LeCun, 1987, Narendra & Parthasarathy, 1990). Letting $\mathbf{\Phi}[n]$ represent the vector of partial derivatives of $J_\alpha$ with respect to $\mathbf{x}_\alpha[n]$, and letting $\mathbf{\Psi}[n]$ represent the vector of partial derivatives of $J_\alpha$ with respect to $\mathbf{u}_\alpha[n]$, Appendix A shows that the gradient of $J_\alpha$ is given by the following recurrence relations:

$$\mathbf{\Phi}[n-1] = \frac{\partial \mathbf{z}_\alpha}{\partial \mathbf{x}_\alpha}^T \mathbf{\Phi}[n] + \frac{\partial \mathbf{u}_\alpha}{\partial \mathbf{x}_\alpha}^T \mathbf{\Psi}[n] - \frac{\partial \mathbf{y}_\alpha}{\partial \mathbf{x}_\alpha}^T (\mathbf{y}^*_\alpha[n] - \mathbf{y}_\alpha[n]) \tag{17}$$

$$\mathbf{\Psi}[n] = \frac{\partial \mathbf{z}_\alpha}{\partial \mathbf{u}_\alpha}^T \mathbf{\Phi}[n] \tag{18}$$

and

$$\nabla_{\mathbf{w}} J_\alpha[n] = \frac{\partial \mathbf{u}_\alpha}{\partial \mathbf{w}}^T \mathbf{\Psi}[n], \tag{19}$$

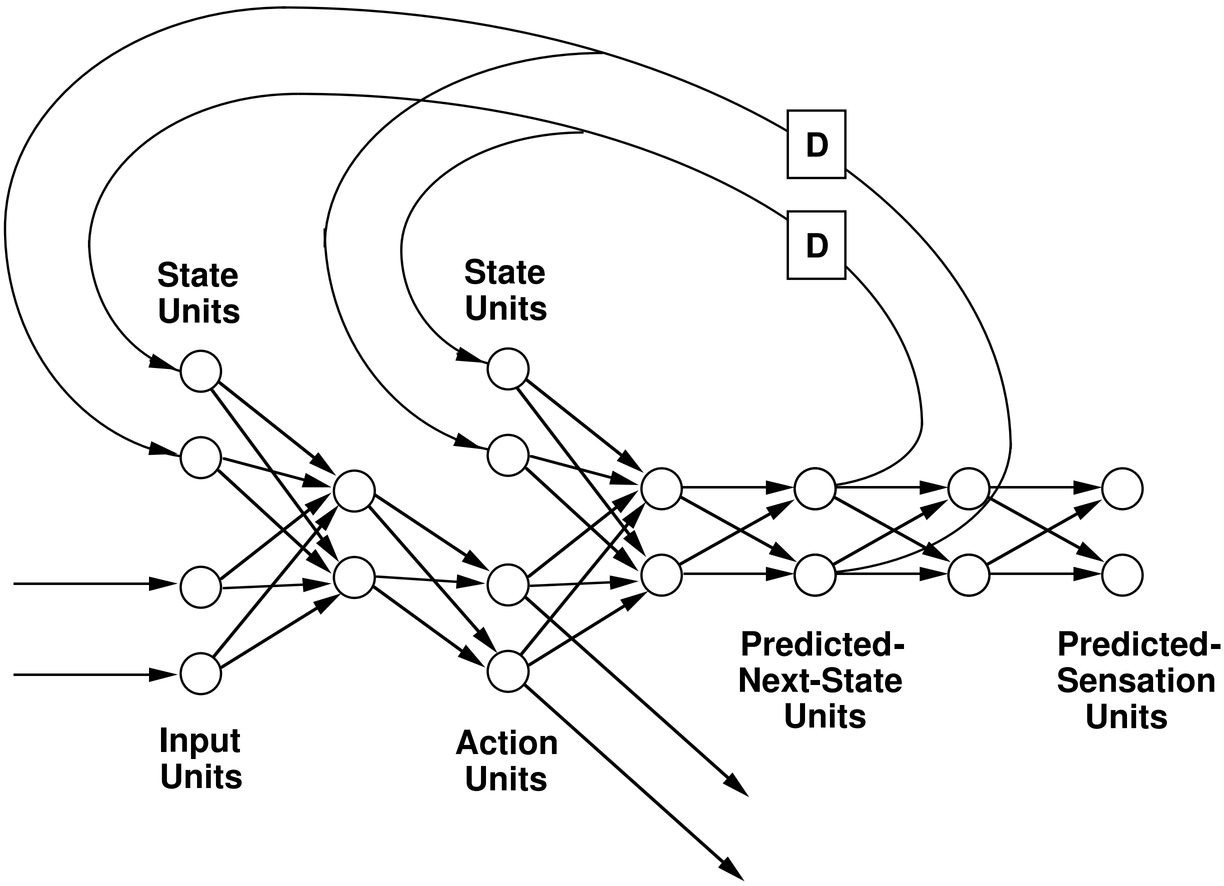


Figure 10: A recurrent network with a forward model. The boxes labeled by D's are unit delay elements.

where the Jacobian matrices are all evaluated at time step $n$ and $\mathbf{z}_\alpha$ stands for $\mathbf{x}_\alpha[n+1]$ (thus, the Jacobian matrices $(\partial \mathbf{z}_\alpha / \partial \mathbf{x}_\alpha)$ and $(\partial \mathbf{z}_\alpha / \partial \mathbf{u}_\alpha)$ are the derivatives of the next-state function). This expression describes backpropagation-in-time in a recurrent network that incorporates a forward model of the next-state function and the output function. As shown in Figure 10, the recurrent network is essentially the same as the network in Figure 9, except that there are explicit connections with unit delay elements between the next-state and the current state.[10] Backpropagation-in-time propagates derivatives backward through these recurrent connections as described by the recurrence relations in Equations 17 and 18.

As in the local optimization case, the equations for computing the gradient

[10] Alternatively, Figure 9 can be thought of as a special case of Figure 10 in which the backpropagated error signals stop at the state units (cf. Jordan, 1986).

involve the multiplication of the performance error $\mathbf{y}^* - \mathbf{y}$ by a series of transpose Jacobian matrices, several of which are unknown a priori. Our approach to estimating the unknown factors is once again to learn forward models of the underlying mappings and to propagate signals backward through the models. Thus the Jacobian matrices $(\partial \mathbf{z}_\alpha / \partial \mathbf{u}_\alpha)$, $(\partial \mathbf{z}_\alpha / \partial \mathbf{x}_\alpha)$, and $(\partial \mathbf{y}_\alpha / \partial \mathbf{x}_\alpha)$ in Equations 17, 18, and 19 are all replaced by estimated quantities in computing the estimated stochastic gradient of $J$.

In the following two sections, we pursue the presentation of the distal learning approach in the context of two problem domains. The first section describes learning in a static environment, whereas the second section describes learning in a dynamic environment. In both sections, we utilize the local optimization formulation of distal learning.

## Static environments

An environment is said to be *static* if the effect of any given action is independent of the history of previous actions. In static environments the mapping from actions to sensations can be characterized without reference to a set of state variables. Such environments provide a simplified domain in which to study the learning of inverse mappings. In this section, we present an illustrative static environment and focus on two issues: (1) the effects of nonconvex inverse images in the transformation from sensations to actions and (2) the problem of goal-directed learning.

The problem that we consider is that of learning the forward and inverse kinematics of a three-joint planar arm. As shown in Figure 11 and Figure 12 the configuration of the arm is characterized by the three joint angles $q_1, q_2$, and $q_3$, and the corresponding pair of Cartesian variables $x_1$ and $x_2$. The function that relates these variables is the *forward kinematic* function $\mathbf{x} = g(\mathbf{q})$. It is obtained in closed form using elementary trigonometry:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} l_1 cos(q_1) + l_2 cos(q_1 + q_2) + l_3 cos(q_1 + q_2 + q_3) \\ l_1 sin(q_1) + l_2 sin(q_1 + q_2) + l_3 sin(q_1 + q_2 + q_3) \end{bmatrix}, \tag{20}$$

where $l_1, l_2$, and $l_3$ are the link lengths.

The forward kinematic function $g(\mathbf{q})$ is a many-to-one mapping—for every Cartesian position that is inside the boundary of the workspace, there are an infinite number of joint angle configurations to achieve that position. This implies that the *inverse kinematic* relation $g^{-1}(\mathbf{x})$ is not a function; rather, there are an infinite number of inverse kinematic functions corresponding to particular choices of points $\mathbf{q}$ in the inverse images of each of the Cartesian positions. The problem of learning an inverse kinematic controller for the arm is that of finding a particular inverse among the many possible inverse mappings.

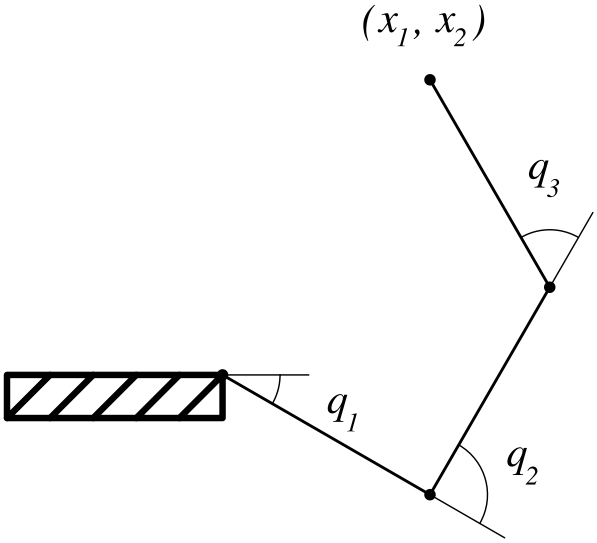


Figure 11: A three-joint planar arm.

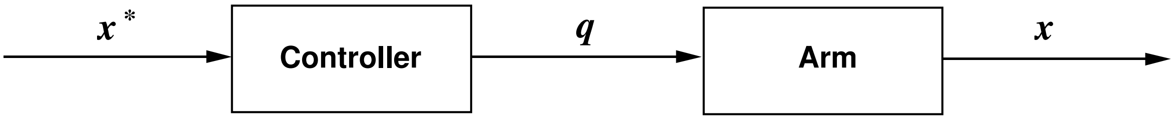


Figure 12: The forward and inverse mappings associated with arm kinematics.

## Simulations

In the simulations reported below, the joint-angle configurations of the arm were represented using the vector $[\cos(q_1 - \frac{\pi}{2}), \cos(q_2), \cos(q_3)]^T$, rather than the vector of joint angles. This effectively restricts the motion of the joints to the intervals $[-\frac{\pi}{2}, \frac{\pi}{2}]$, $[0, \pi]$, and $[0, \pi]$, respectively, assuming that each component of the joint-angle configuration vector is allowed to range over the interval $[-1, 1]$. The Cartesian variables $x_1$ and $x_2$ were represented as real numbers ranging over $[-1, 1]$. In all of the simulations, these variables were represented directly as real-valued activations of units in the network. Thus, three units were used to represent joint-angle configurations and two units were used to represent Cartesian positions. Further details on the simulations are provided in Appendix B.

### The nonconvexity problem

One approach to learning an inverse mapping is to provide training pairs to the learner by observing the input/output behavior of the environment and reversing the role of the inputs and outputs. This approach, which we referred to earlier as "direct inverse modeling," has been proposed in the domain of inverse kinematics by Kuperstein (1988). Kuperstein's idea is to randomly sample points $\mathbf{q}'$ in joint space and to use the real arm to evaluate the forward kinematic function $\mathbf{x} = g(\mathbf{q}')$, thereby obtaining training pairs $(\mathbf{x}, \mathbf{q}')$ for learning the controller. The controller is learned by optimization of the following cost functional:

$$J = \frac{1}{2} E\{(\mathbf{q}' - \mathbf{q})^T (\mathbf{q}' - \mathbf{q})\} \tag{21}$$

where $\mathbf{q} = h(\mathbf{x}^*)$ is the output of the controller.

As we discussed earlier, a difficulty with the direct inverse modeling approach is that the optimization of the cost functional in Equation 21 does not necessarily yield an inverse kinematic function. The problem arises because of the many-to-one nature of the forward kinematic function (cf. Figure 8). In particular, if two or more of the randomly sampled points $\mathbf{q}'$ happen to map to the same endpoint, then the training data that is provided to the controller is one-to-many. The particular manner in which the inconsistency is resolved depends on the form of the cost functional—use of the sum-of-squared error given in Equation 21 yields an arithmetic average over points that map to the same endpoint. An average in joint space, however, does not necessarily yield a correct result in Cartesian space, because the inverse images of nonlinear transformations are not necessarily convex. This implies that the output of the controller may be in error even though the system has converged to the minimum of the cost functional.

In Figure 13 we demonstrate that the inverse kinematics of the three-joint arm is not convex. To see if this nonconvexity has the expected effect on the direct inverse modeling procedure we conducted a simulation in which a feedforward network with one hidden layer was used to learn the inverse kinematics of the three-joint arm. The simulation provided target vectors to the network by sampling randomly from a uniform distribution in joint space. Input vectors were obtained by mapping the target vectors into Cartesian space according to Equation 20. The initial value of the root-mean-square (RMS) joint-space error was 1.41, filtered over the first 500 trials. After $50,000$ learning trials the filtered error reached asymptote at a value of 0.43. A vector field was then plotted by providing desired Cartesian vectors as inputs to the network, obtaining the joint-angle outputs, and mapping these outputs into Cartesian space using Equation 20. The resulting vector field is shown in Figure 14. As can be seen, there is substantial error at many positions of the workspace, even though the learning algorithm has converged. If training is continued, the loci of the errors continue to shift, but the RMS error remains approximately constant. Although this error is partially due to the finite learning rate and the random sampling

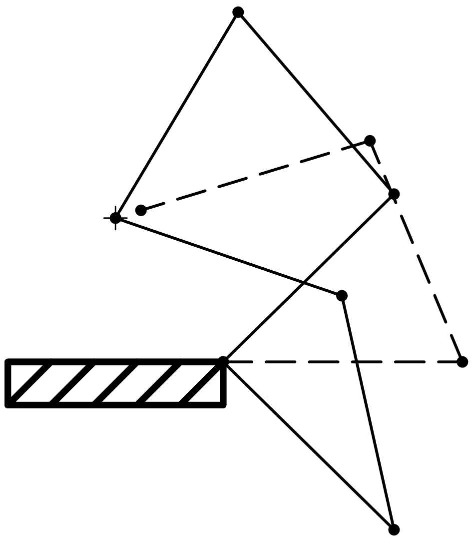

Figure 13: The nonconvexity of inverse kinematics. The dotted configuration is an average in joint space of the two solid configurations.

procedure ("misadjustment," see Widrow & Stearns, 1985), the error remains above 0.4 even when the learning rate is taken to zero. Thus, misadjustment cannot account for the error, which must be due to the nonconvexity of the inverse kinematic relation. Note, for example, that the error observed in Figure 13 is reproduced in the lower left portion of Figure 14.

In Figure 15, we demonstrate that the distal learning approach can find a particular inverse kinematic mapping. We performed a simulation that was initialized with the incorrect controller obtained from direct inverse modeling. The simulation utilized a forward model that had been trained previously (the forward model was trained during the direct inverse modeling trials). A grid of 285 evenly spaced positions in Cartesian space was used to provide targets during the second phase of the distal learning procedure.[11] On each trial the error in Cartesian space was passed backward through the forward model and used to change the weights of the controller. After $28,500$ such learning trials (100 passes through the grid of targets), the resulting vector field was plotted. As shown in the figure, the vector error decreases toward zero throughout the workspace; thus, the controller is converging toward a particular inverse kinematic function.

[11]The use of a grid is not necessary; the procedure also works if Cartesian positions are sampled randomly on each trial.

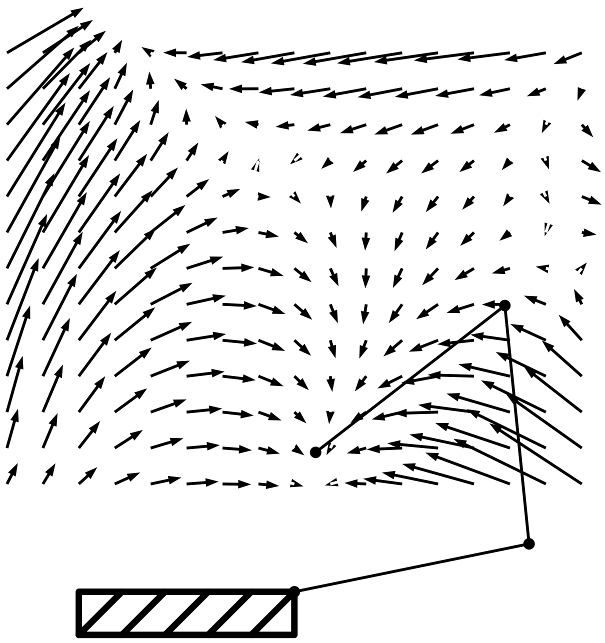

Figure 14: Near-asymptotic performance of direct inverse modeling. Each vector represents the error at a particular position in the workspace.

### Additional constraints

A further virtue of the distal learning approach is the ease with which it is possible to incorporate additional constraints in the learning procedure and thereby bias the choice of a particular inverse function. For example, a minimum-norm constraint can be realized by adding a penalty term of the form $-\lambda \mathbf{x}$ to the propagated errors at the output of the controller. Temporal smoothness constraints can be realized by incorporating additional error terms of the form $\lambda(\mathbf{x}[n] - \mathbf{x}[n-1])$. Such constraints can be defined at other sites in the network as well, including the output units or hidden units of the forward model. It is also possible to provide additional contextual inputs to the controller and thereby learn multiple, contextually-appropriate inverse functions. These aspects of the distal learning approach are discussed in more detail in Jordan (1988, 1990).

### Goal-directed learning

Direct inverse modeling does not learn in a goal-directed manner. To learn a specific Cartesian target, the procedure must sample over a sufficiently large region of joint space and rely on interpolation. Heuristics may be available to restrict the search to certain regions of joint space, but such heuristics are essentially prior knowledge

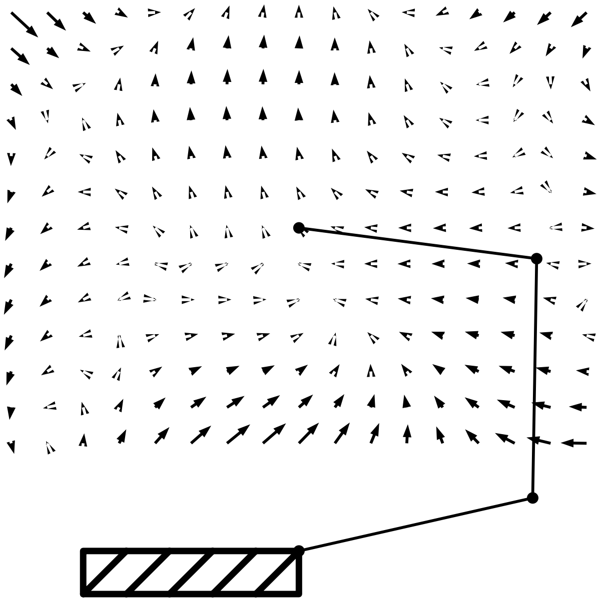

Figure 15: Near-asymptotic performance of distal learning.

about the nature of the inverse mapping and can equally well be incorporated into the distal learning procedure.

Distal learning is fundamentally goal-directed. It is based on the performance error for a specific Cartesian target and is capable of finding an exact solution for a particular target in a small number of trials. This is demonstrated by the simulation shown in Figure 16. Starting from the controller shown in Figure 14, a particular Cartesian target was presented for ten successive trials. As shown in Figure 16, the network reorganizes itself so that the error is small in the vicinity of the target. After ten additional trials, the error at the target is zero within the floating-point resolution of the simulation.

### Approximate forward models

We conducted an additional simulation to study the effects of inaccuracy in the forward model. The simulation varied the number of trials allocated to the learning of the forward model from 50 to 5000. The controller was trained to an RMS criterion of 0.001 at the three target positions $(-0.25, 0.25)$, $(0.25, 0.25)$, and $(0.0, 0.65)$. As shown in Figure 17, the results demonstrate that an accurate controller can be found with an inaccurate forward model. Fewer trials are needed to learn the target positions to criterion with the most accurate forward model; however, the dropoff in learning rate with less accurate forward models is relatively slight. Reasonably

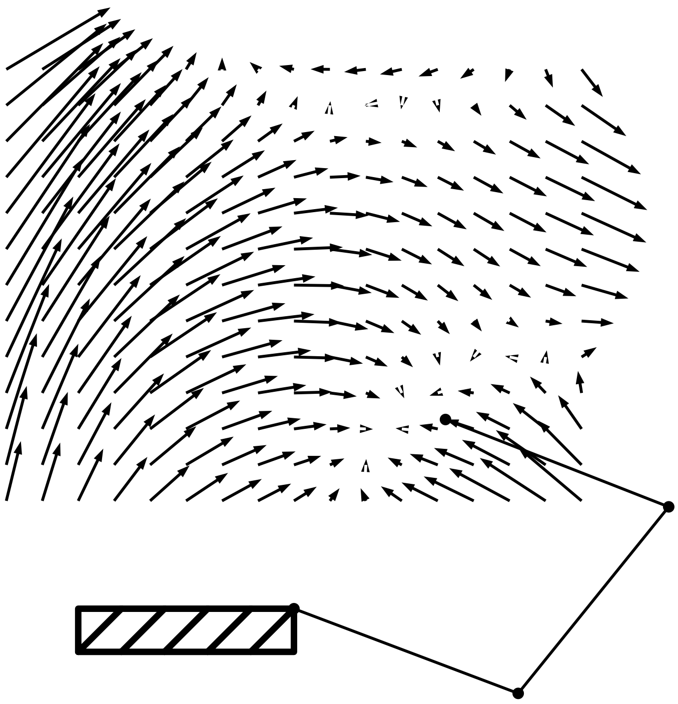

Figure 16: Goal-directed learning. A Cartesian target in the lower right portion of the figure was presented for ten successive trials. The error vectors are close to zero in the vicinity of the target.

rapid learning is obtained even when the forward model is trained for only 50 trials, even though the average RMS error in the forward model is 0.34 m after 50 trials, compared to 0.11 m after 5000 trials.

### Further comparisons with direct inverse modeling

In problems with many output variables it is often unrealistic to acquire an inverse model over the entire workspace. In such cases the goal-directed nature of distal learning is particularly important because it allows the system to obtain inverse images for a restricted set of locations. However, the forward model must also be learned over a restricted region of action space, and there is no general a priori method for determining the appropriate region of the space in which to sample. That is, although distal learning is goal-directed in its acquisition of the inverse model, it is not inherently goal-directed in its acquisition of the forward model.

Because neither direct inverse modeling nor distal learning is entirely goal-directed, in any given problem it is important to consider whether it is more reasonable to acquire the inverse model or the forward model in a non-goal-directed

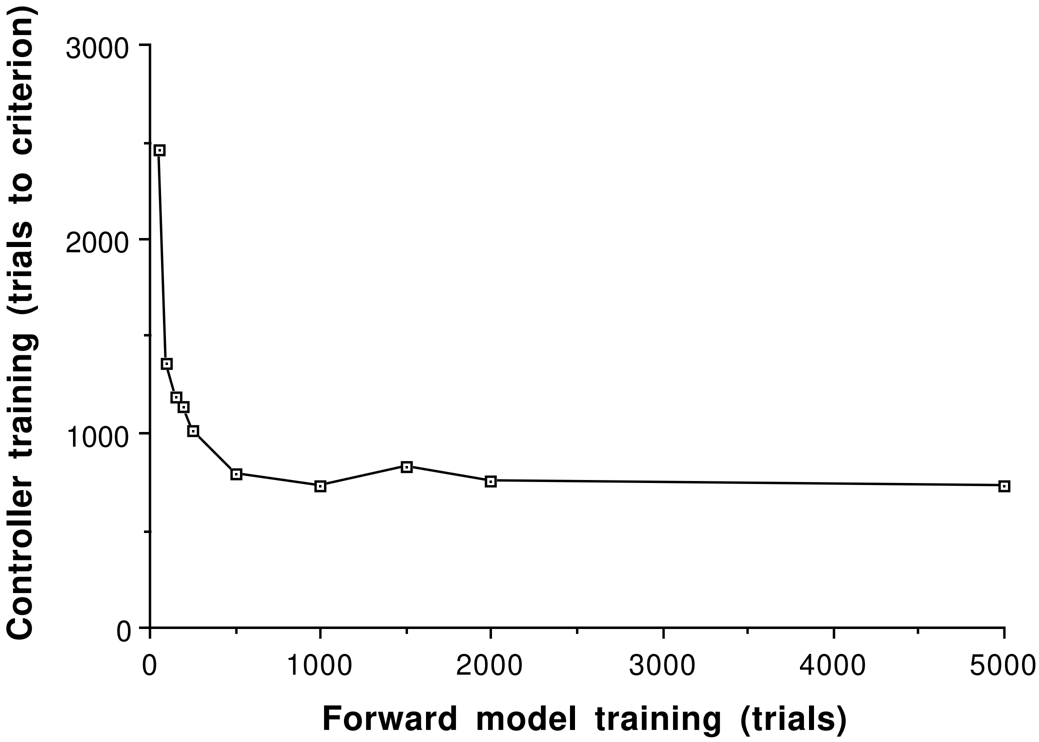


Figure 17: Number of trials required to train the controller to an RMS criterion of 0.001 as a function of the number of trials allocated to training the forward model. Each point is an average over three runs.

manner. This issue is problem-dependent, depending on the nature of the function being learned, the nature of the class of functions that can be represented by the learner, and the nature of the learning algorithm. It is worth noting, however, that there is an inherent tradeoff in complexity between the inverse model and the forward model, due to the fact that their composition is the identity mapping. This tradeoff suggests a complementarity between the classes of problems for which direct inverse modeling and distal learning are appropriate. We believe that distal learning is more generally useful, however, because an inaccurate forward model is generally acceptable whereas an inaccurate inverse model is not. In many cases, it may be preferable to learn an inaccurate forward model that is specifically inverted at a desired set of locations rather than learning an inaccurate inverse model directly and relying on interpolation.

## Dynamic environments: One-step dynamic models

To illustrate the application of distal learning to problems in which the environment has state, we consider the problem of learning to control a two-joint robot arm. Controlling a dynamic robot arm involves finding the appropriate torques to cause the arm to follow desired trajectories. The problem is difficult because of the nonlinear couplings between the motions of the two links and because of the fictitious torques due to the rotating coordinate systems.

The arm that we consider is the two-link version of the arm shown previously in Figure 11. Its configuration at each point in time is described by the joint angles $q_1(t)$ and $q_2(t)$, and by the Cartesian variables $x_1(t)$ and $x_2(t)$. The kinematic function $\mathbf{x}(t) = g(\mathbf{q}(\mathbf{t}))$ that relates joint angles to Cartesian variables can be obtained by letting $l_3$ equal zero in Equation 20:

$$\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} l_1 cos(q_1(t)) + l_2 cos(q_1(t) + q_2(t)) \\ l_1 sin(q_1(t)) + l_2 sin(q_1(t) + q_2(t)) \end{bmatrix},$$

where $l_1$ and $l_2$ are the link lengths. The state space for the arm is the four-dimensional space of positions and velocities of the links.

The essence of robot arm dynamics is a mapping between the torques applied at the joints and the resulting angular accelerations of the links. This mapping is dependent on the state variables of angle and angular velocity. Let $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$ represent the vector of joint angles, angular velocities, and angular accelerations, respectively, and let $\boldsymbol{\tau}$ represent the torques. In the terminology of earlier sections, $\mathbf{q}$ and $\dot{\mathbf{q}}$ together constitute the "state" and $\boldsymbol{\tau}$ is the "action." For convenience, we take $\ddot{\mathbf{q}}$ to represent the "next-state" (see the discussion below). To obtain an analog of the next-state function in Equation 1, the following differential equation can be derived for the angular motion of the links, using standard Newtonian or Lagrangian dynamical formulations (Craig, 1986):

$$M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + G(\mathbf{q}) = \boldsymbol{\tau}, \tag{22}$$

where $M(\mathbf{q})$ is an inertia matrix, $C(\mathbf{q}, \dot{\mathbf{q}})$ is a matrix of Coriolis and centripetal terms, and $G(\mathbf{q})$ is the vector of torque due to gravity. Our interest is not in the physics behind these equations per se, but in the functional relationships that they define. In particular, to obtain a "next-state function," we rewrite Equation 22 by solving for the accelerations to yield:

$$\ddot{\mathbf{q}} = M^{-1}(\mathbf{q})[\boldsymbol{\tau} - C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - G(\mathbf{q})], \tag{23}$$

where the existence of $M^{-1}(\mathbf{q})$ is always assured (Craig, 1986). Equation 23 expresses the state-dependent relationship between torques and accelerations at each moment in time: Given the state variables $\mathbf{q}(t)$ and $\dot{\mathbf{q}}(t)$, and given the torque $\boldsymbol{\tau}(t)$, the acceleration $\ddot{\mathbf{q}}(t)$ can be computed by substitution in Equation 23. We refer to this computation as the *forward dynamics* of the arm.

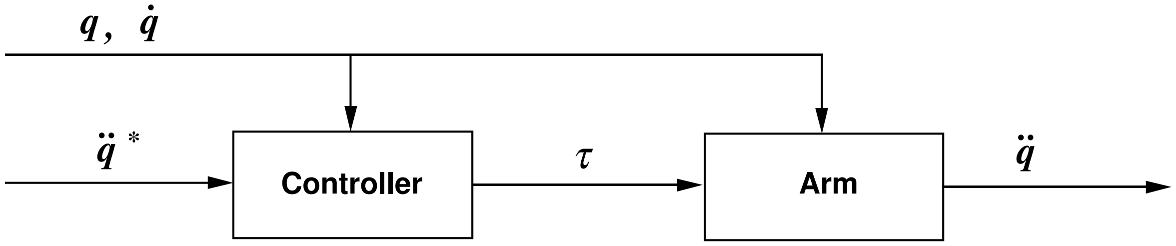


Figure 18: The forward and inverse mappings associated with arm dynamics.

An inverse mapping between torques and accelerations can be obtained by interpreting Equation 22 in the proper manner. Given the state variables $\mathbf{q}(t)$ and $\dot{\mathbf{q}}(t)$, and given the acceleration $\ddot{\mathbf{q}}(t)$, substitution in Equation 22 yields the corresponding torques. This (algebraic) computation is refered to as *inverse dynamics.* It should be clear that inverse dynamics and forward dynamics are complementary computations: Substitution of $\boldsymbol{\tau}$ from Equation 22 into Equation 23 yields the requisite identity mapping.

These relationships between torques, accelerations, and states are summarized in Figure 18. It is useful to compare this figure with the kinematic example shown in Figure 12. In both the kinematic case and the dynamic case, the forward and inverse mappings that must be learned are fixed functions of the instantaneous values of the relevant variables. In the dynamic case, this is due to the fact that the structural terms of the dynamical equations (the terms $M$, $C$, and $G$) are explicit functions of state rather than time. The dynamic case can be thought of as a generalization of the kinematic case in which additional contextual (state) variables are needed to index the mappings that must be learned.[12]

Figure 18 is an instantiation of Figure 6, with the acceleration playing the role of the "next-state." In general, for systems described by differential equations, it is convenient to define the notion of "next-state" in terms of the time derivative of one or more of the state variables (e.g., accelerations in the case of arm dynamics). This definition is entirely consistent with the development in preceding sections; indeed, if the differential equations in Equation 22 are simulated in discrete time on a computer, then the numerical algorithm must compute the accelerations defined by Equation 23 to convert the positions and velocities at the current time step into the positions and velocities at the next time step.[13]

[12]This perspective is essentially that underlying the local optimization formulation of distal learning.

[13]Because of the amplification of noise in differentiated signals, however, most realistic implementations of forward dynamical models would utilize positions and velocities rather than accelerations. In such cases the numerical integration of Equation 23 would be incorporated as part of the forward model.

## Learning a dynamic forward model

A forward model of arm dynamics is a network that learns a prediction $\hat{\ddot{\mathbf{q}}}$ of the acceleration $\ddot{\mathbf{q}}$, given the position $\mathbf{q}$, the velocity $\dot{\mathbf{q}}$, and the torque $\boldsymbol{\tau}$. The appropriate teaching signal for such a network is the actual acceleration $\ddot{\mathbf{q}}$, yielding the following cost functional:

$$L = \frac{1}{2}E\{(\ddot{\mathbf{q}} - \hat{\ddot{\mathbf{q}}})^T(\ddot{\mathbf{q}} - \hat{\ddot{\mathbf{q}}})\}. \tag{24}$$

The prediction $\hat{\ddot{\mathbf{q}}}$ is a function of the position, the velocity, the torque and the weights:

$$\hat{\ddot{\mathbf{q}}} = \hat{f}(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau}, \mathbf{w}).$$

For an appropriate ensemble of control trajectories, this cost functional is minimized when a set of weights is found such that $\hat{f}(\cdot, \mathbf{w})$ best approximates the forward dynamical function given by Equation 23.

An important difference between kinematic problems and dynamic problems is that it is generally infeasible to produce arbitrary random control signals in dynamical environments, because of considerations of stability. For example, if $\boldsymbol{\tau}(t)$ in Equation 22 is allowed to be a stationary white-noise stochastic process, then the variance of $\mathbf{q}(t)$ approaches infinity (much like a random walk). This yields data that is of little use for learning a model. We have used two closely related approaches to overcome this problem. The first approach is to produce random *equilibrium* positions for the arm rather than random torques. That is, we define a new control signal $\mathbf{u}(t)$ such that the augmented arm dynamics are given by:

$$M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + G(\mathbf{q}) = k_v(\dot{\mathbf{q}} - \dot{\mathbf{u}}) + k_p(\mathbf{q} - \mathbf{u}), \tag{25}$$

for fixed constants $k_p$ and $k_v$. The random control signal $\mathbf{u}$ in this equation acts as a "virtual" equilibrium position for the arm (Hogan, 1985) and the augmented dynamics can be used to generate training data for learning the forward model. The second approach also utilizes Equation 25 and differs from the first approach only in the choice of the control signal $\mathbf{u}(t)$. Rather than using random controls, the target trajectories themselves are used as controls (that is, the trajectories utilized in the second phase of learning are also used to train the forward model). This approach is equivalent to using a simple fixed-gain proportional-derivative (PD) feedback controller to stabilize the system along a set of reference trajectories and thereby generate training data.[14] Such use of an auxiliary feedback controller is similar to its use in the feedback-error learning (Kawato, et al., 1987) and direct inverse modeling (Atkeson & Reinkensmeyer, 1988; Miller, 1987) approaches. As is discussed below, the second approach has the advantage that it does not require the forward model to be learned in a separate phase.

[14] A PD controller is a device whose output is a weighted sum of position errors and velocity errors. The position errors and the velocity errors are multiplied by fixed numbers (gains) before being summed.

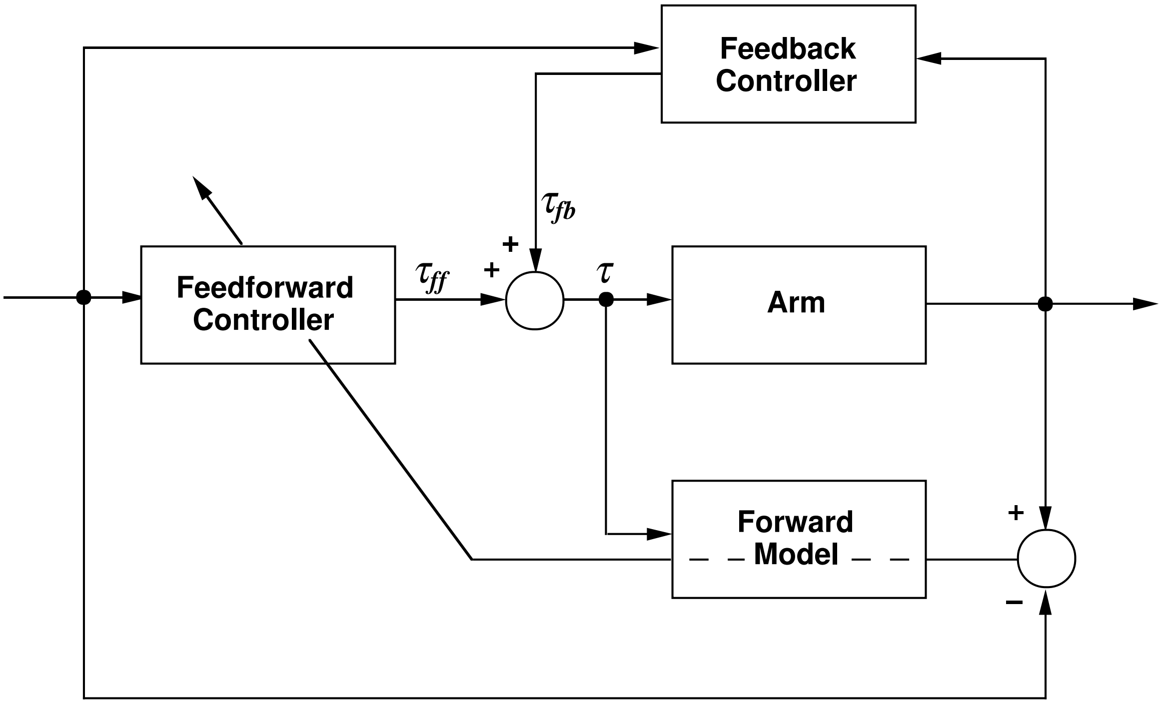


Figure 19: The composite control system.

### Composite control system

The composite system for controlling the arm is shown in Figure 19. The control signal in this diagram is the torque $\boldsymbol{\tau}$, which is the sum of two components:

$$\boldsymbol{\tau} = \boldsymbol{\tau}_{ff} + \boldsymbol{\tau}_{fb},$$

where $\boldsymbol{\tau}_{ff}$ is a feedforward torque and $\boldsymbol{\tau}_{fb}$ is the (optional) feedback torque produced by the auxiliary feedback controller. The feedforward controller is the learning controller that converges toward a model of the inverse dynamics of the arm. In the early phases of learning, the feedforward controller produces small random torques, thus the major source of control is provided by the error-correcting feedback controller.[15] When the feedforward controller begins to be learned it produces torques that allow the system to follow desired trajectories with smaller error, thus the role of the feedback controller is diminished. Indeed, in the limit where the feedforward controller converges to a perfect inverse model, the feedforward torque causes the system to follow a desired trajectory without error and the feedback controller is

[15]As is discussed below, this statement is not entirely accurate. The learning algorithm itself provides a form of error-correcting feedback control.

therefore silent (assuming no disturbances). Thus the system shifts automatically from feedback-dominated control to feedforward-dominated control over the course of learning (see also Atkeson & Reinkensmeyer, 1988; Kawato, et al., 1987; Miller, 1988).

There are two error signals utilized in learning inverse dynamics: The prediction error $\ddot{\mathbf{q}} - \hat{\ddot{\mathbf{q}}}$ and the performance error $\ddot{\mathbf{q}}^* - \ddot{\mathbf{q}}$.[16] The prediction error is used to train the forward model as discussed in the previous section. Once the forward model is at least partially learned, the performance error can be used in training the inverse model. The error is propagated backward through the forward model and down into the feedforward controller where the weights are changed. This process minimizes the distal cost functional:

$$J = \frac{1}{2} E\{(\ddot{\mathbf{q}}^* - \ddot{\mathbf{q}})^T (\ddot{\mathbf{q}}^* - \ddot{\mathbf{q}})\}. \tag{26}$$

## Simulations

The arm was modeled using rigid body dynamics assuming the mass to be uniformly distributed along the links. The links were modeled as thin cylinders. Details on the physical constants are provided in Appendix C. The simulation of the forward dynamics of the arm was carried out using a fourth-order Runge-Kutta algorithm with a sampling frequency of 200 Hz. The control signals provided by the networks were sampled at 100 Hz.

Standard feedforward connectionist networks were used in all of the simulations. There were two feedforward networks in each simulation—a controller and a forward model—with overall connectivity as shown in Figure 18 (with the box labelled "Arm" being replaced by a forward model). Both the controller and the forward model were feedforward networks with a single layer of logistic hidden units. In all of the simulations, the state variables, torques, and accelerations were represented directly as real-valued activations in the network. Details of the networks used in the simulations are provided in Appendix B.

In all but the final simulation reported below, the learning of the forward model and the learning of an inverse model were carried out in separate phases. The forward model was learned in an initial phase by using a random process to drive the augmented dynamics given in Equation 25. The random process was a white noise position signal chosen uniformly within the workspace shown in Figure 20. The learning of the forward model was terminated when the filtered RMS prediction error reached 0.75 rad/s$^2$.

[16] As noted above, it is also possible to include the numerical integration of $\hat{\ddot{\mathbf{q}}}$ as part of the forward model and learn a mapping whose output is the predicted next-state $(\hat{\mathbf{q}}[n], \hat{\dot{\mathbf{q}}}[n])$. This approach may be preferred for systems in which differentiation of noisy signals is a concern.

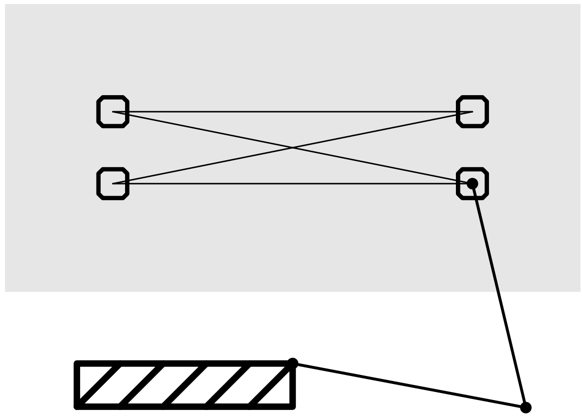

Figure 20: The workspace (the grey region) and four target paths. The trajectories move from left to right along the paths shown.

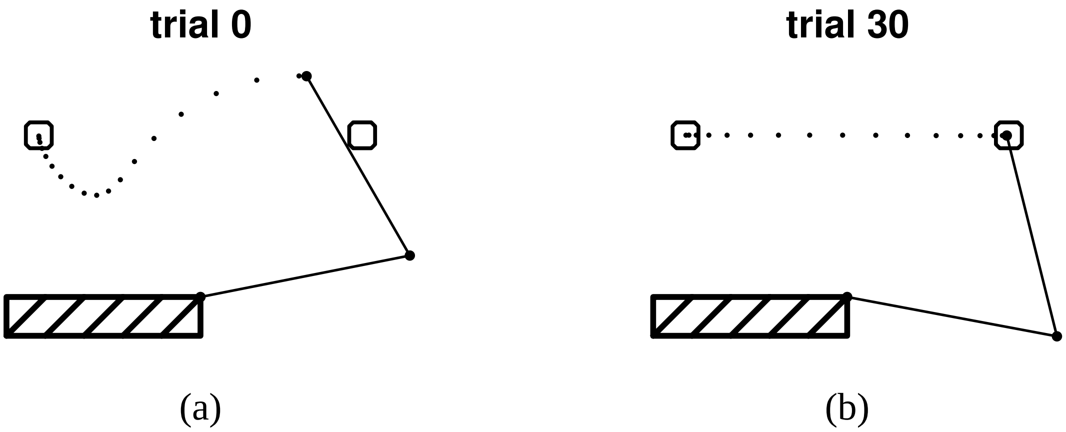


Figure 21: Performance on one of the four learned trajectories. (a) Before learning. (b) After 30 learning trials.

**Learning with an auxiliary feedback controller**

After learning the forward model, the system learned to control the arm along the four paths shown in Figure 20. The target trajectories were minimum jerk trajectories of one second duration each. An auxiliary proportional-derivative (PD) feedback controller was used, with position gains of 1.0 $\mathrm{N} \cdot \mathrm{m/rad}$ and velocity gains of 0.2 $\mathrm{N} \cdot \mathrm{m} \cdot \mathrm{s/rad}$. Figure 21 shows the performance on a particular trajectory before learning (with the PD controller alone) and during the 30th learning trial. The corresponding waveforms are shown in Figure 22 and Figure 23. The middle graphs in these figures show the feedback torques (dashed lines) and the feedforward torques (solid lines). As can be seen, in the early phases of learning the torques are

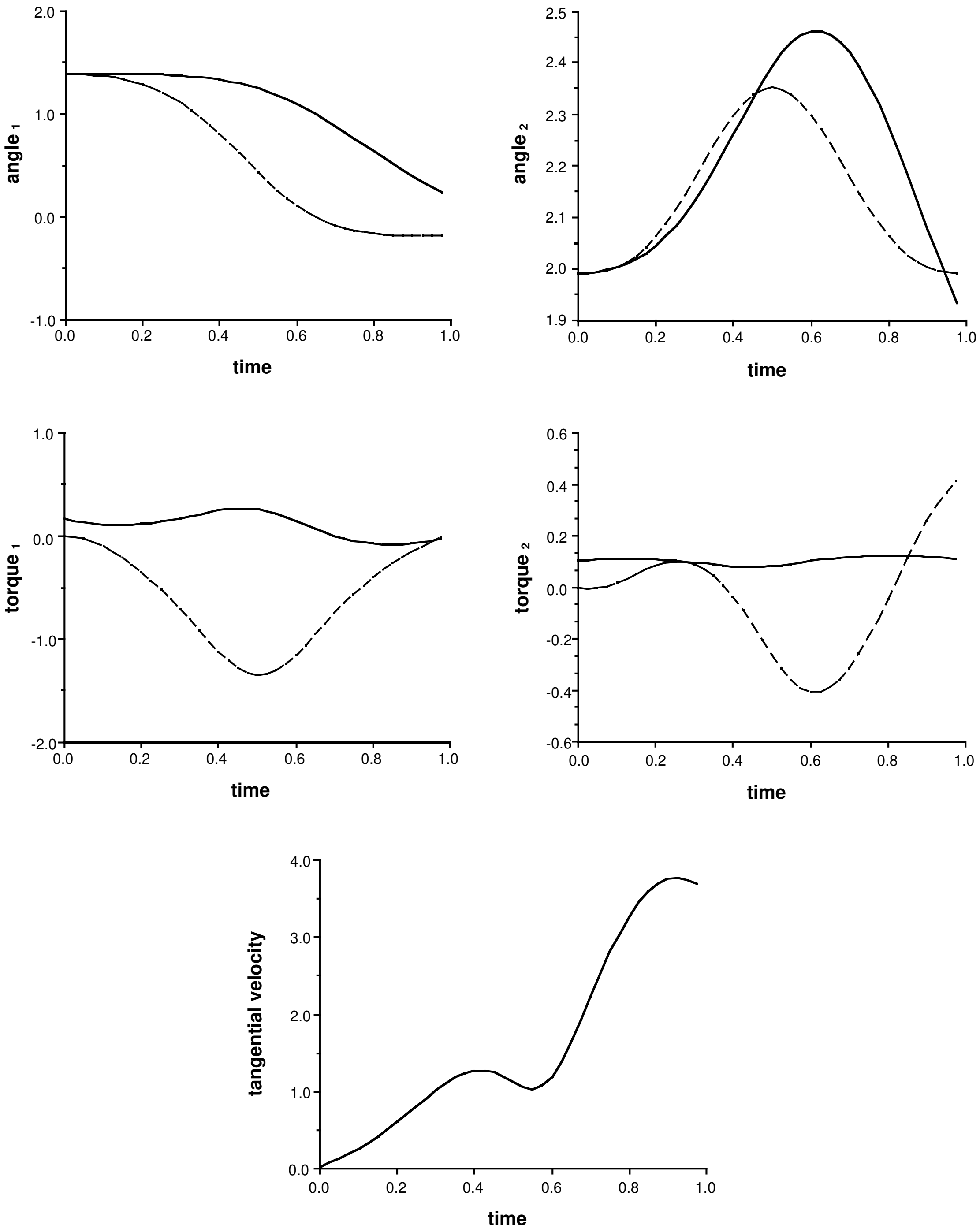


Figure 22: Before learning. In the top graphs, the dotted line is the reference angle and the solid line is the actual angle. In the middle graphs, the dotted line is the feedback torque and the solid line is the feedforward torque.

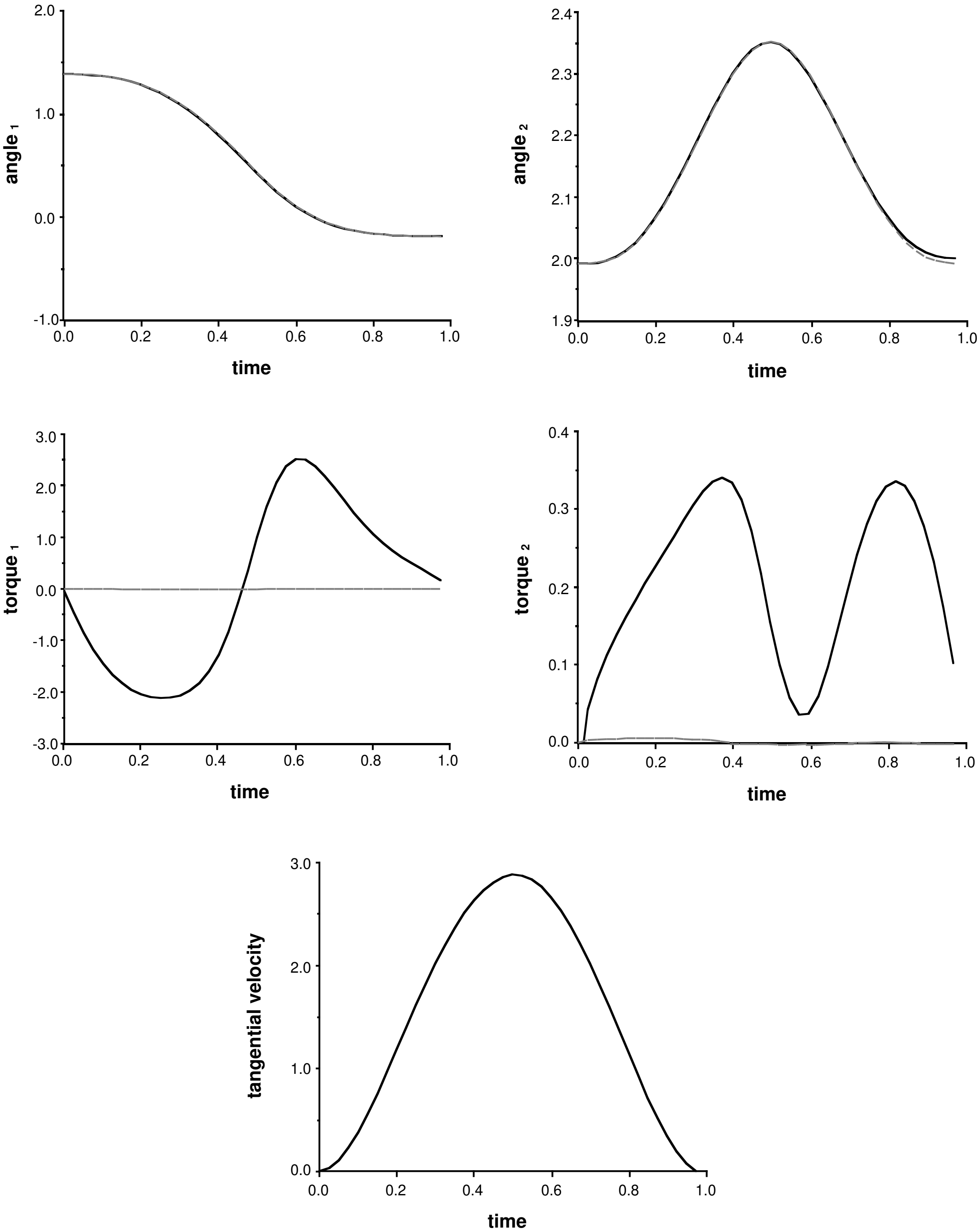


Figure 23: After learning. In the top graphs, the dotted line is the reference angle and the solid line is the actual angle. In the middle graphs, the dotted line is the feedback torque and the solid line is the feedforward torque.

generated principally by the feedback controller and in later phases the torques are generated principally by the feedforward controller.

### Learning without an auxiliary feedback controller

An interesting consequence of the goal-directed nature of the forward modeling approach is that it is possible to learn an inverse dynamic model without using an auxiliary feedback controller. To see why this is the case, first note that minimum jerk reference trajectories (and other "smooth" reference trajectories) change slowly in time. This implies that successive time steps are essentially repeated learning trials on the same input vector; thus, the controller converges rapidly to a "solution" for a local region of state space. As the trajectory evolves, the solution tracks the input; thus, the controller produces reasonably good torques prior to any "learning." Put another way, the distal learning approach is itself a form of error-correcting feedback control in the parameter space of the controller. Such error correction must eventually give way to convergence of the weights if the system is to learn an inverse model; nonetheless, it is a useful feature of the algorithm that it tends to stabilize the arm during learning.

This behavior is demonstrated by the simulations shown in Figure 24. The figure shows performance on the first learning trial as a function of the learning rate. The results demonstrate that changing the learning rate essentially changes the gain of the error-correcting behavior of the algorithm. When the learning rate is set to 0.5, the system produces nearly perfect performance on the first learning trial. This feature of the algorithm makes it important to clarify the meaning of the learning curves obtained with the distal learning approach. Figure 25 shows two such learning curves. The lower curve is the RMS error that is obtained with a learning rate of 0.1. The upper curve is the RMS error that is obtained when the learning rate is temporarily set to zero after each learning trial. Setting the learning rate to zero allows the effects of learning to be evaluated separately from the error-correcting behavior. The curves clearly reveal that on the early trials the main contributor to performance is error correction rather than learning.

### Combining forward dynamics and forward kinematics

Combining the forward dynamic models of this section with the forward kinematic models of the preceding section makes it possible to train the controller using Cartesian target trajectories. Given that the dynamic model and the kinematic model can be learned in parallel, there is essentially no performance decrement associated with using the combined system. In our simulations, we find that learning times increase by approximately eight percent when using Cartesian targets rather than joint angle targets.

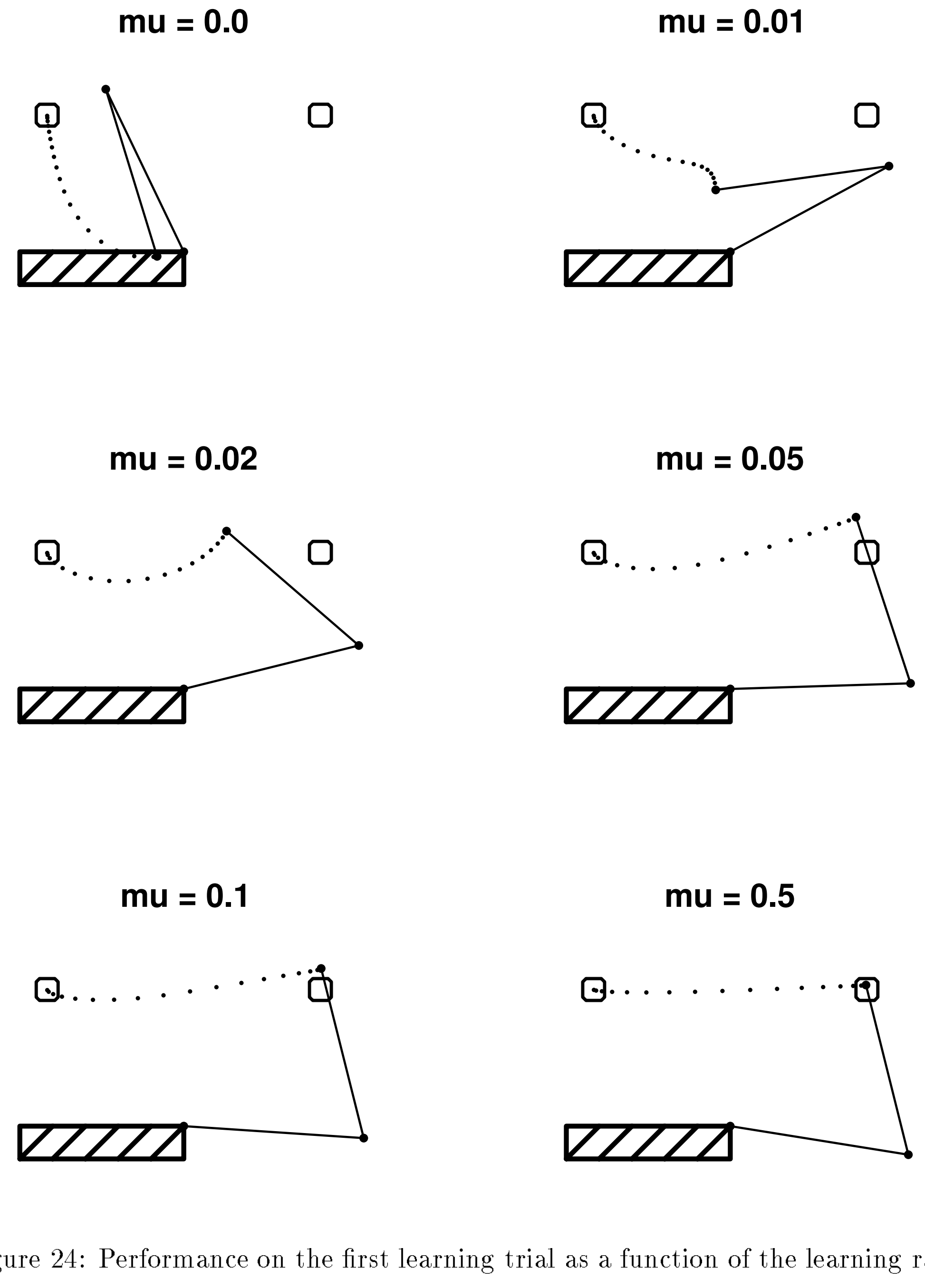


Figure 24: Performance on the first learning trial as a function of the learning rate.

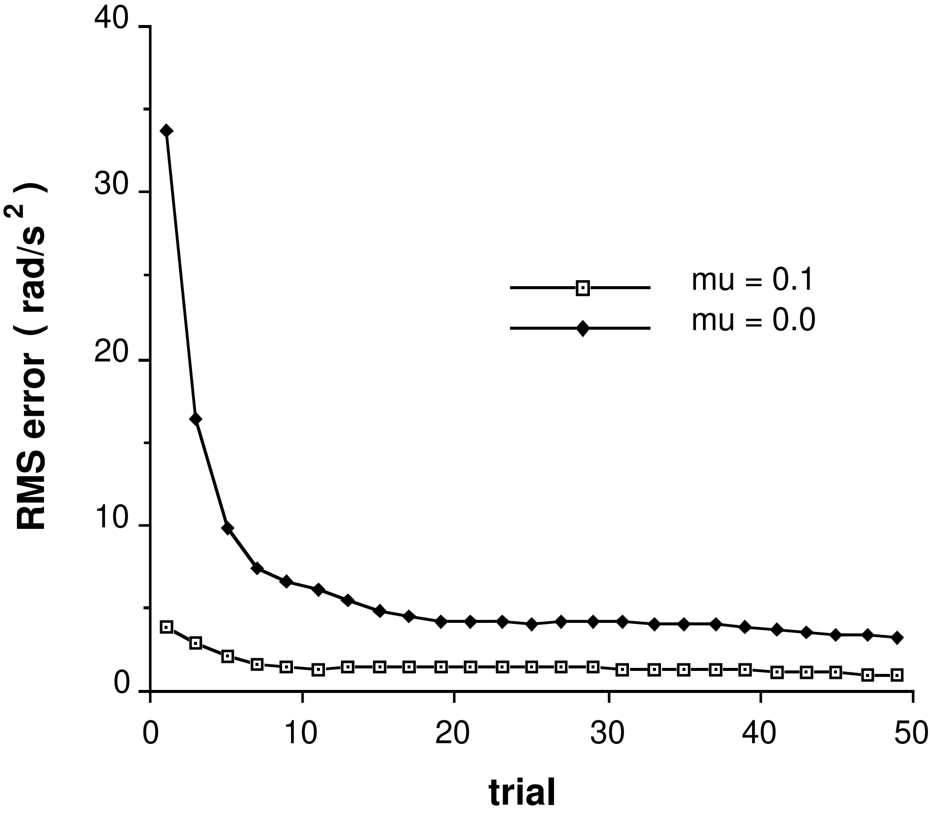


Figure 25: RMS error for zero and non-zero learning rates.

### Learning the forward model and the controller simultaneously

The distal learning approach involves using a forward model to train the controller; thus, learning of the forward model must precede the learning of the controller. It is not necessary, however, to learn the forward model over the entire state space before learning the controller—a *local* forward model is generally sufficient. Moreover, as we have discussed, the distal learning approach does not require an exact forward model—approximate forward models often suffice. These two facts, in conjunction with the use of smooth reference trajectories, imply that it should be possible to learn the forward model and the controller simultaneously. An auxiliary feedback controller is needed to stabilize the system initially; however, once the forward model begins to be learned, the learning algorithm itself tends to stabilize the system. Moreover, as the controller begins to be learned, the errors decrease and the effects of the feedback controller diminish automatically. Thus the system bootstraps itself toward an inverse model.

The simulation shown in Figure 26 demonstrates the feasibility of this approach.

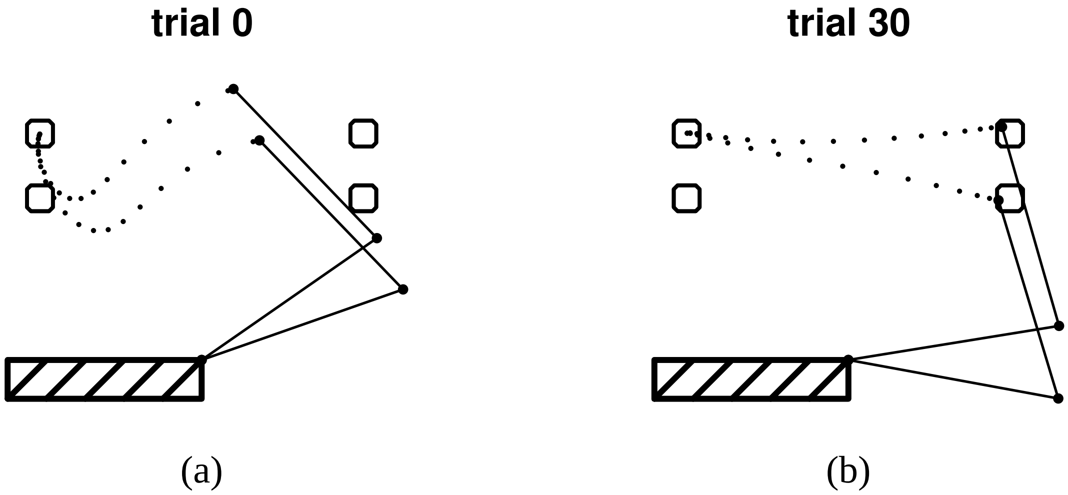


Figure 26: Learning the forward model and the controller simultaneously. (a) Performance before learning on two of the target trajectories. (b) Performance after 30 learning trials.

Using the same architecture as in previous experiments the system learned four target trajectories starting with small random weights in both the controller and the forward model. On each time step two passes of the backpropagation algorithm were required—one pass with the prediction error $\ddot{\mathbf{q}} - \hat{\ddot{\mathbf{q}}}$ to change the weights of the forward model, and a second pass with the performance error $\ddot{\mathbf{q}}^* - \ddot{\mathbf{q}}$ to change the weights of the controller. An auxiliary proportional-derivative (PD) feedback controller was used, with position gains of 1.0 $\mathrm{N \cdot m/rad}$ and velocity gains of 0.2 $\mathrm{N \cdot m \cdot s/rad}$. As shown in the figure, the system converges to an acceptable level of performance after 30 learning trials.

Although the simultaneous learning procedure requires more presentations of the target trajectories to achieve a level of performance comparable to that of the two-phase learning procedure, the simultaneous procedure is in fact more efficient than two-phase learning because it dispenses with the initial phase of learning the forward model. This advantage must be weighed against certain disadvantages; in particular, the possibility of instability is enhanced because of the error in the gradients obtained from the partially-learned forward model. In practice we find that it is often necessary to use smaller step sizes in the simultaneous learning approach than in the two-phase learning approach. Preliminary experiments have also shown that is worthwhile to choose specialized representations that enhance the speed with which the forward model converges. This can be done separately for the state variable input and the torque input.

## Dynamic environments: Simplified models

In the previous section we demonstrated how the temporal component of the distal supervised learning problem can be addressed by knowledge of a set of state variables for the environment. Assuming prior knowledge of a set of state variables is tantamount to assuming that the learner has prior knowledge of the maximum delay between the time at which an action is issued and the time at which an effect is observed in the sensation vector. In the current section we present preliminary results that aim to broaden the scope of the distal learning approach to address problems in which the maximum delay is not known (see also Werbos, 1987).

A simple example of such a problem is one in which a robot arm is required to be in a certain configuration at time $T$, where $T$ is unknown, and where the trajectory in the open interval from 0 to $T$ is unconstrained.[17] One approach to solving such problems is to learn a one-step forward model of the arm dynamics and then to use backpropagation-in-time in a recurrent network that includes the forward model and a controller (Jordan, 1990; Kawato, 1990).[18] In many problems involving delayed temporal consequences, however, it is neither feasible nor desirable to learn a dynamic forward model of the environment, either because the environment is too complex or because solving the task at hand does not require knowledge of the evolution of all of the state variables. Consider for example the problem of predicting the height of a splash of water when stones of varying size are dropped into a pond. It is unlikely that a useful one-step dynamic model could be learned for the fluid dynamics of the pond. Moreover, if the control problem is to produce splashes of particular desired heights, it may not be necessary to model fluid dynamics in detail. A simple forward model that predicts an integrated quantity—splash height as a function of the size of the stone—may suffice.

Jordan and Jacobs (1990) illustrated this approach by using distal learning to solve the problem of learning to balance an inverted pendulum on a moving cart. This problem is generally posed as an avoidance control problem in which the only corrective information provided by the environment is a signal to indicate that failure has occured (Barto, Sutton, & Anderson, 1983). The delay between actions (forces applied to the cart) and the failure signal is unknown and indeed can be arbitrarily large. In the spirit of the foregoing discussion, Jordan and Jacobs also assumed that it is undesirable to model the dynamics of the cart-pole system; thus, the controller cannot be learned by using backpropagation-in-time in a recurrent network that includes a one-step dynamic model of the plant.

[17] A unique trajectory may be specified by enforcing additional constraints on the temporal evolution of the actions; however, the only explicit target information is assumed to be that provided at the final time step.

[18] In Kawato's work, backpropagation-in-time is implemented in a spatially-unrolled network and the gradients are used to change activations rather than weights; however, the idea of using a one-step forward dynamic model is the same. See also Nguyen & Widrow (1989) for an application to a kinematic problem.

The approach adopted by Jordan and Jacobs involves learning a forward model whose output is an integrated quantity—an estimate of the inverse of the time until failure. This estimate is learned using temporal difference techniques (Sutton, 1988). At time steps on which failure occurs, the target value for the forward model is unity:

$$e(t) = 1 - \hat{z}(t),$$

where $\hat{z}(t)$ is the output of the forward model, and $e(t)$ is the error term used to change the weights. On all other time steps, the following temporal difference error term is used:

$$e(t) = \frac{1}{1 + \hat{z}^{-1}(t+1)} - \hat{z}(t),$$

which yields an increasing arithmetic series along any trajectory that leads to failure. Once learned, the output of the forward model is used to provide a gradient for learning the controller. In particular, because the desired outcome of balancing the pole can be described as the goal of maximizing the time until failure, the algorithm learns the controller by using zero minus the output of the forward model as the distal error signal.[19]

The forward model used by Jordan and Jacobs differs in an important way from the other forward models described in this paper. Because the time-until-failure depends on future actions of the controller, the mapping that the forward model must learn depends not only on fixed properties of the environment but also on the controller. When the controller is changed by the learning algorithm, the mapping that the forward model must learn also changes. Thus the forward model must be updated continuously during the learning of the controller. In general, for problems in which the forward model learns to estimate an integral of the closed-loop dynamics, the learning of the forward model and the controller *must* proceed in parallel.

Temporal difference techniques provide the distal learning approach with enhanced functionality. They make it possible to learn to make long-term predictions and thereby adjust controllers on the basis on quantities that are distal in time. They can also be used to learn multi-step forward models. In conjunction with backpropagation-in-time, they provide a flexible set of techniques for learning actions on the basis of temporally-extended consequences.

# Discussion

In this paper we have argued that the supervised learning paradigm is broader than is commonly assumed. The distal supervised learning framework extends supervised learning to problems in which desired values are available only for the distal

[19] This technique can be considered as an example of using supervised learning algorithms to solve a reinforcement learning problem (see below).

consequences of a learner's actions and not for the actions themselves. This is a significant weakening of the classical notion of the "teacher" in the supervised learning paradigm. In this section we provide further discussion of the class of problems that can be treated within the distal supervised learning framework. We discuss possible sources of training data and we contrast distal supervised learning with reinforcement learning.

### How is training data obtained?

To provide support for our argument that distal supervised learning is more realistic than classical supervised learning it is necessary to consider possible sources of training data for distal supervised learning. We discuss two such sources, which we refer to as *imitation* and *envisioning*.

One of the most common ways for humans to acquire skills is through imitation. Skills such as dance or athletics are often learned by observing another person performing the skill and attempting to replicate their behavior. Although in some cases a teacher may be available to suggest particular patterns of limb motion, such direct instruction does not appear to be a necessary component of skill acquisition. A case in point is speech acquisition—children acquire speech by hearing speech sounds, not by receiving instruction on how to move their articulators.

Our conception of a distal supervised learning problem involves a set of (intention, desired outcome) training pairs. Learning by imitation clearly makes desired outcomes available to the learner. With regard to intentions, there are three possibilities. First, the learner may know or be able to infer the intentions of the person serving as a model. Alternatively, an idiosyncratic internal encoding of intentions is viable as long as the encoding is consistent. For example, a child acquiring speech may have an intention to drink, may observe another person obtaining water by uttering the form "water," and may utilize the acoustic representation of "water" as a distal target for learning the articulatory movements for expressing a desire to drink, even though the other person uses the water to douse a fire. Finally, when the learner is acquiring an inverse model, as in the simulations reported in this paper, the intention is obviously available because it is the same as desired outcome.

Our conception of distal supervised learning problem as a set of training pairs is of course an abstraction that must be elaborated when dealing with complex tasks. In a complex task such as dance, it is presumably not easy to determine the choice of sensory data to be used as distal targets for the learning procedure. Indeed, the learner may alter the choice of targets once he or she has achieved a modicum of skill. The learner may also need to decompose the task into simpler tasks and to set intermediate goals. We suspect that the role of external "teachers" is to help with these representational issues rather than to provide proximal targets directly to the learner.

Another source of data for the distal supervised learning paradigm is a process that we refer to as "envisioning." Envisioning is a general process of converting abstract goals into their corresponding sensory realization, without regard to the actions needed to achieve the goals. Envisioning involves deciding what it would "look like" or "feel like" to perform some task. This process presumably involves general deductive and inductive reasoning abilities as well as experience with similar tasks. The point that we want to emphasize is that envisioning need not refer to the actions that are needed to actually carry out a task; that is the problem solved by the distal learning procedure.

### Comparisons with reinforcement learning

An alternative approach to solving the class of problems that we have discussed in this paper is to use reinforcement learning algorithms (Barto, 1989; Sutton, 1984). Reinforcement learning algorithms are based on the assumption that the environment provides an *evaluation* of the actions produced by the learner. Because the evaluation can be an arbitrary function, the approach is in principle applicable to the general problem of learning on the basis of distal signals.

Reinforcement learning algorithms work by updating the probabilities of emitting particular actions. The updating procedure is based on the evaluations received from the environment. If the evaluation of an action is favorable then the probability associated with that action is increased and the probabilities associated with all other actions are decreased. Conversely, if the evaluation is unfavorable, then the probability of the given action is decreased and the probabilities associated with all other actions are increased. These characteristic features of reinforcement learning algorithms differ in important ways from the corresponding features of supervised learning algorithms. Supervised learning algorithms are based on the existence of a signed error vector rather than an evaluation. The signed error vector is generally, although not always, obtained by comparing the actual output vector to a target vector. If the signed error vector is small, corresponding to a favorable evaluation, the algorithm initiates no changes. If the signed error vector is large, corresponding to an unfavorable evaluation, the algorithm corrects the current action in favor of a particular alternative action. Supervised learning algorithms do not simply increase the probabilities of all alternative actions; rather, they choose particular alternatives based on the directionality of the signed error vector.[20]

It is important to distinguish between learning *paradigms* and learning *algorithms*. Because the same learning algorithm can often be utilized in a variety of learning paradigms, a failure to distinguish between paradigms and algorithms can lead to misunderstanding. This is particularly true of reinforcement learning tasks and supervised learning tasks because of the close relationships between evaluative signals and signed error vectors. A signed error vector can always be converted into an evaluative signal (any bounded monotonic function of the norm of the signed

[20]As pointed out by Barto, Sutton & Anderson (1983), this distinction between reinforcement learning and supervised learning is significant only if the learner has a repertoire of more than two actions.

error vector suffices); thus, reinforcement learning algorithms can always be used for supervised learning problems. Conversely, an evaluative signal can always be converted into a signed error vector (using the machinery that we have discussed; see also Munro, 1987); thus, supervised learning algorithms can always be used for reinforcement learning problems. The definition of a learning paradigm, however, has more to do with the manner in which a problem is naturally posed than with the algorithm used to solve the problem. In the case of the basketball player, for example, assuming that the environment provides directional information such as "too far to the left," "too long," or "too short," is very different from assuming that the environment provides evaluative information of the form "good," "better," or "best". Furthermore, learning algorithms differ in algorithmic complexity when applied across paradigms: Using a reinforcement learning algorithm to solve a supervised learning problem is likely to be inefficient because such algorithms do not take advantage of directional information. Conversely, using supervised learning algorithms to solve reinforcement learning problems is likely to be inefficient because of the extra machinery that is required to induce a signed error vector.

In summary, although it has been suggested that the difference between reinforcement learning and supervised learning is the latter's reliance on a "teacher," we feel that this argument is mistaken. The distinction between the supervised learning paradigm and the reinforcement learning paradigm lies in the interpretation of environmental feedback as an error signal or as an evaluative signal, not the coordinate system in which such signals are provided. Many problems involving distal credit assignment may be better conceived of as supervised learning problems rather than reinforcement learning problems if the distal feedback signal can be interpreted as a performance error.

## Conclusions

There are a number of difficulties with the classical distinctions between "unsupervised," "reinforcement," and "supervised" learning. Supervised learning is generally said to be dependent on a "teacher" to provide target values for the output units of a network. This is viewed as a limitation because in many domains there is no such teacher. Nevertheless, the environment often does provide sensory information about the consequences of an action which can be employed in making internal modifications just as if a teacher had provided the information to the learner directly. The idea is that the learner first acquires an internal model that allows prediction of the consequences of actions. The internal model can be used as a mechanism for transforming distal sensory information about the consequences of actions into proximal information for making internal modifications. This two-phase procedure extends the scope of the supervised learning paradigm to include a broad range of problems in which actions are transformed by an unknown dynamical process before being compared to desired outcomes.

We first illustrated this approach in the case of learning an inverse model of a simple "static" environment. We showed that our method of utilizing a forward model of the environment has a number of important advantages over the alternative method of building the inverse model directly. These advantages are especially apparent in cases where there is no unique inverse model. We also showed that this idea can be extended usefully to the case of a dynamic environment. In this case, we simply elaborate both the forward model and the learner (i.e., controller) so they take into account the current state of the environment. Finally, we showed how this approach can be combined with temporal difference techniques to build a system capable of learning from sensory feedback that is subject to an unknown delay.

We also suggested that comparative work in the study of learning can be facilitated by making a distinction between learning algorithms and learning paradigms. A variety of learning algorithms can often be applied to a particular instance of a learning paradigm; thus, it is important to characterize not only the paradigmatic aspects of any given learning problem, such as the nature of the interaction between the learner and the environment and the nature of the quantities to be optimized, but also the tradeoffs in algorithmic complexity that arise when different classes of learning algorithms are applied to the problem. Further research is needed to delineate the natural classes at the levels of paradigms and algorithms and to clarify the relationships between levels. We believe that such research will begin to provide a theoretical basis for making distinctions between candidate hypotheses in the empirical study of human learning.

# References

Atkeson, C. G., & Reinkensmeyer, D. J. (1988). Using associative content-addressable memories to control robots. *IEEE Conference on Decision and Control.* San Francisco, CA.

Barto, A. G. (1989). From chemotaxis to cooperativity: Abstract exercises in neuronal learning strategies. In R. M. Durbin, R. C. Maill, & G. J. Mitchison (Eds.), *The computing neurone.* Reading, MA: Addison-Wesley Publishers.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC-13*, 834-846.

Becker, S. & Hinton, G. E. (1989). *Spatial coherence as an internal teacher for a neural network.* (Tech. Rep. CRG-TR-89-7). Toronto: University of Toronto.

Carlson, A. B. (1986). *Communication Systems.* New York: McGraw-Hill.

Craig, J. J. (1986). *Introduction to Robotics.* Reading, MA: Addison-Wesley Publishers.

Gelfand, I. M. & Fomin, S. V. (1963). *Calculus of variations.* Englewood Cliffs, N. J.: Prentice-Hall.

Goodwin, G. C. & Sin, K. S. (1984). *Adaptive filtering prediction and control.* Englewood Cliffs, NJ: Prentice-Hall.

Grossberg, S. (1987). Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11, 23-63.

Hinton, G. E. & Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: Volume 1*, 282-317. Cambridge, MA: MIT Press.

Hogan, N. (1984). An organising principle for a class of voluntary movements. *Journal of Neuroscience*, *4*, 2745-2754.

Jordan, M. I. (1983). *Mental practice.* Unpublished dissertation proposal, Center for Human Information Processing, University of California, San Diego.

Jordan, M. I. (1986). Serial order: A parallel, distributed processing approach. (Technical Report 8604). La Jolla, CA: University of California, San Diego.

Jordan, M. I. (1988). *Supervised learning and systems with excess degress of freedom.* (COINS Tech. Rep. 88-27). Amherst, MA: University of Massachusetts, Computer and Information Sciences.

Jordan, M. I., & Rosenbaum, D. A. (1989). Action. In M. I. Posner (Ed.), *Foundations of Cognitive Science.* Cambridge, MA: MIT Press.

Jordan, M.I. (1990). Motor learning and the degrees of freedom problem. In M. Jeannerod (Ed.), *Attention and Performance, XIII.* Hillsdale, NJ: Erlbaum.

Jordan, M. I., & Jacobs, R. A. (1990). Learning to control an unstable system with forward modeling. In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems 2.* San Mateo, CA: Morgan Kaufmann.

Kawato, M. (1990). Computational schemes and neural network models for formation and control of multijoint arm trajectory. In W. T. Miller, III, R. S. Sutton, & P. J. Werbos (Eds.), *Neural Networks for Control.* Cambridge: MIT Press.

Kawato, M., Furukawa, K., & Suzuki, R. (1987). A hierarchical neural-network model for control and learning of voluntary movement. *Biological Cybernetics*, *57*, 169-185.

Kirk, D. E. (1970). *Optimal control theory.* Englewood Cliffs, NJ: Prentice-Hall.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, *43*, 56-69.

Kuperstein, M. (1988). Neural model of adaptive hand-eye coordination for single postures. *Science*, *239*, 1308-1311.

LeCun, Y. (1985). A learning scheme for asymmetric threshold networks. *Proceedings of Cognitiva 85.* Paris, France.

LeCun, Y. (1987). *Modèles connexionnistes de l'apprentissage.* Unpublished doctoral dissertation, Université de Paris, VI.

Linsker, R. (1988). Self-organization in a perceptual network. *Computer*, *21*, 105-117.

Ljung, L. & Söderström, T. (1986). *Theory and practice of recursive identification.* Cambridge: MIT Press.

Miller, W. T. (1987). Sensor-based control of robotic manipulators using a general learning algorithm. *IEEE Journal of Robotics and Automation*, *3*, 157-165.

Mozer, M. C. & Bachrach, J. (1990). *Discovering the structure of a reactive environment by exploration.* In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems 2.* San Mateo, CA: Morgan Kaufmann.

Munro, P. (1987). A dual back-propagation scheme for scalar reward learning. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum.

Narendra, K. S. & Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, *1*, 4-27.

Nguyen, D. & Widrow, B. (1989). The truck backer-upper: An example of self-learning in neural networks. In: *Proceedings of the International Joint Conference on Neural Networks*, *2*, 357-363. Piscataway, NJ: IEEE Press.

Parker, D. (1985). *Learning-logic.* (Tech. Rep. TR-47). Cambridge, MA: MIT Sloan School of Management.

Robinson, A. J. & Fallside, F. (1989). Dynamic reinforcement driven error propagation networks with application to game playing. *Proceedings of Neural Information Systems.* American Institute of Physics.

Rosenblatt, F. (1962). *Principles of neurodynamics.* New York: Spartan.

Rumelhart, D. E. (1986). Learning sensorimotor programs in parallel distributed processing systems. *US-Japan Joint Seminar on Competition and Cooperation in Neural Nets, II.* Unpublished presentation.

Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing*: *Volume 1*, 318-363. Cambridge, MA: MIT Press.

Rumelhart, D. E., Smolensky, P., McClelland, J. L. & Hinton, G. E. (1986). Schemata and sequential thought processes in PDP models. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing*: *Volume 2*, 7-57. Cambridge, MA: MIT Press.

Rumelhart, D. E. & Zipser, D. (1986). Feature discovery by competitive learning. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing*: *Volume 1*, 151-193. Cambridge, MA: MIT Press.

Schmidhuber, J. H. (1990). An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In: *Proceedings of the International Joint Conference on Neural Networks*, *2*, 253-258. Piscataway, NJ: IEEE Press.

Sutton, R. S. (1984). *Temporal credit assignment in reinforcement learning.* (COINS Tech. Rep. 84-02). Amherst, MA: University of Massachusetts, Computer and Information Sciences.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9-44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning.*

Werbos, P. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* Unpublished doctoral dissertation, Harvard University.

Werbos, P. (1987). Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics*, *17*, 7-20.

Widrow, B. & Hoff, M. E. (1960). Adaptive switching circuits. *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record*, *Part 4*, 96-104.

Widrow, B., & Stearns, S. D. (1985). *Adaptive signal processing.* Englewood Cliffs, NJ: Prentice-Hall.

# Appendix A

To obtain an expression for the gradient of Equation 16, we utilize a continuous-time analog, derive a necessary condition, and then convert the result into discrete time. To simplify the exposition we compute partial derivatives with respect to the actions $\mathbf{u}$ instead of the weights $\mathbf{w}$. The resulting equations are converted into gradients for the weights by premultiplying by the transpose of the Jacobian matrix $(\partial \mathbf{u}/\partial \mathbf{w})$.

Let $\mathbf{u}(t)$ represent an action trajectory and let $\mathbf{y}(t)$ represent a sensation trajectory. These trajectories are linked in the forward direction by the dynamical equations:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

and

$$\mathbf{y} = g(\mathbf{x}).$$

The action vector $\mathbf{u}$ is assumed to depend on the current state and the target vector:

$$\mathbf{u} = h(\mathbf{x}, \mathbf{y}^*).$$

The functional to be minimized is given by the following integral:

$$J = \frac{1}{2}\int_0^T (\mathbf{y}^* - \mathbf{y})^T(\mathbf{y}^* - \mathbf{y})dt,$$

which is the continuous-time analog of Equation 16 (we have suppressed the subscript $\alpha$ to simplify the notation).

Let $\mathbf{\Phi}(t)$ and $\mathbf{\Psi}(t)$ represent vectors of time-varying Lagrange multipliers and define the Lagrangian:

$$L(t) = \frac{1}{2}(\mathbf{y}^* - \mathbf{y})^T(\mathbf{y}^* - \mathbf{y}) + [f(\mathbf{x}, \mathbf{u}) - \dot{\mathbf{x}}]^T\mathbf{\Phi} + [h(\mathbf{x}, \mathbf{y}^*) - \mathbf{u}]^T\mathbf{\Psi}.$$

The Lagrange multipliers have an interpretation as sensitivities of the cost with respect to variations in $\dot{\mathbf{x}}$ and $\mathbf{y}$, respectively. Because these sensitivities become partial derivatives when the problem is converted to discrete time, we are interested in solving for $\mathbf{\Psi}(t)$.

A necessary condition for an optimizing solution is that it satisfy the Euler-Lagrange equations (Gelfand & Fomin, 1963):

$$\frac{\partial L}{\partial \mathbf{x}} - \frac{d}{dt}\frac{\partial L}{\partial \dot{\mathbf{x}}} = 0$$

and

$$\frac{\partial L}{\partial \mathbf{u}} - \frac{d}{dt}\frac{\partial L}{\partial \dot{\mathbf{u}}} = 0$$

at each moment in time. These equations are the equivalent in function space of the familiar procedure of setting the partial derivatives equal to zero.

Substituting for $L(t)$ and simplifying we obtain:

$$\dot{\mathbf{\Phi}} = -\frac{\partial f}{\partial \mathbf{x}}^T \mathbf{\Phi} - \frac{\partial k}{\partial \mathbf{x}}^T \mathbf{\Psi} + \frac{\partial g}{\partial \mathbf{x}}^T (\mathbf{y}^* - \mathbf{y})$$

and

$$\mathbf{\Psi} = \frac{\partial f}{\partial \mathbf{u}}^T \mathbf{\Phi}.$$

Using an Euler approximation, these equations can be written in discrete time as recurrence relations:

$$\mathbf{\Phi}[n-1] = \mathbf{\Phi}[n] + h\frac{\partial f}{\partial \mathbf{x}}^T \mathbf{\Phi}[n] + h\frac{\partial k}{\partial \mathbf{x}}^T \mathbf{\Psi}[n] - h\frac{\partial g}{\partial \mathbf{x}}^T (\mathbf{y}^*[n] - \mathbf{y}[n]) \tag{27}$$

and

$$\mathbf{\Psi}[n] = \frac{\partial f}{\partial \mathbf{u}}^T \mathbf{\Phi}[n], \tag{28}$$

where $h$ is the sampling period of the discrete approximation. To utilize these recurrence relations in a discrete-time network, the sampling period $h$ is absorbed in the network approximations of the continuous-time mappings. The network approximation of $f$ must also include an identity feedforward component to account for the initial autoregressive term in Equation 27. Premultiplication of Equation 28 by the transpose of the Jacobian matrix $(\partial \mathbf{u}/\partial \mathbf{w})$ then yields Equations 17, 18, and 19 in the main text.

# Appendix B

The networks used in all of the simulations were standard feedforward connectionist networks (see Rumelhart, Hinton, & Williams, 1986).

*Activation functions*—The input units and the output units of all networks were linear and the hidden units were logistic with asymptotes of $-1$ and 1.

*Input and target values*—In the kinematic arm simulations, the joint angles were represented using the vector $[\cos(q_1 - \frac{\pi}{2}), \cos(q_2), \cos(q_3)]^T$. The Cartesian targets were scaled to lie between $-1$ and 1 and fed directly into the network.

In the dynamic arm simulations, all variables—joint angles, angular velocities, angular accelerations, and torques—were scaled and fed directly into the network. The scaling factors were chosen such that the scaled variables ranged approximately from $-1$ to 1.

*Initial weights*—Initial weights were chosen randomly from a uniform distribution on the interval $[-0.5, 0.5]$.

*Hidden units*—A single layer of 50 hidden units was used in all networks. No attempt was made to optimize the number of the hidden units or their connectivity.

*Parameter values*—A learning rate of 0.1 was used in all of the kinematic arm simulations. The momentum was set to 0.5.

In the dynamic arm simulations, a learning rate of 0.1 was used in all cases, except for the simulation shown in Figure 24 in which the learning rate was manipulated explicitly. No momentum was used in the dynamic arm simulations.

## Appendix C

The dynamic arm was modeled using rigid-body mechanics. The link lengths were 0.33 m for the proximal link and 0.32 m for the distal link. The masses of the links were 2.52 kg and 1.3 kg.

The mass was assumed to be distributed uniformly along the links. The moments of inertia of the links about their centers of mass were therefore given by $I_i = m_i l_i^2/12$, yielding 0.023 $\mathrm{kg} \cdot \mathrm{m}^2$ and 0.012 $\mathrm{kg} \cdot \mathrm{m}^2$ for the proximal and distal links, respectively.