Neural Network Generation of Temporal Sequences from Single Static Vector Inputs using Varying Length Distal Target Sequences

PhD Dissertation

Shaun Gittens

University of Maryland at College Park

December 14, 2006

Dissertation Committee :

Dr. Carol Espy-Wilson

Dr. Bill Gasarch

Dr. Jack Minker

Dr. Don Perlis

Dr. James Reggia (Chair)

Abstract

Training an agent to operate in an environment whose mappings are largely unknown is generally recognized to be exceptionally difficult. Further, granting such a learning agent the ability to produce an appropriate sequence of actions entirely from a single input stimulus remains a key problem. Various reinforcement learning techniques have been utilized to handle such learning tasks, but convergence to optimal policies is not guaranteed for many of these methods. Traditional supervised learning methods hold more assurances of convergence, but these methods are not well suited for tasks where desired actions in the output space of the learner, termed *proximal* actions, are not available for training. Rather, target outputs from the environment are *distal* from where the learning takes place. For example, a child acquiring language who makes speech errors must learn to correct them based on heard information that reaches his/her auditory cortex which is distant from the motor cortical regions that control speech output. While distal supervised learning techniques for neural networks have been devised, it remains to be established how they can be trained to produce sequences of proximal actions from only a single static input. In this research, I develop an architecture which incorporates recurrent multi-layered neural networks that possess some form of history in the form of a context vector into the distal supervised learning framework, enabling it to learn to generate correct proximal sequences from single static input stimuli. This is in contrast to existing distal learning methods designed for non-recurrent neural network learners that utilize no concept of memory of their prior behavior. Also, I adapt a technique in this research known as teacher forcing for use in distal sequential learning settings which is shown to result in more efficient usage of the recurrent neural network's context layer. The effectiveness of my approach is demonstrated by applying it to acquire varying length phoneme sequence generation behavior using only previously heard and stored auditory phoneme sequences. The results indicate that simple recurrent backpropagation networks can be integrated with distal learning methods to

create effective sequence generators even when they do not constantly update current state information.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

What series of robot hand and arm movements is required to draw a square using a paintbrush on a canvas? What sequence of motor commands should be issued to the brain's primary motor cortex which could eventually yield the verbal utterance "mother" from a subject's mouth? These are types of problems that are addressed in an active area of research within machine learning which is concerned with how one trains an agent to learn to exhibit some desired time varying behavior while acting in an external environment. Existing supervised learning strategies for training neural nets are well studied and effective in many domains, but a teacher must provide the correct series of desired proximal actions to the agent in order to be successful. Here, the term *proximal* describes the immediate actions taken by the learning agent while operating in the environment. In contrast, the term *distal* describes the consequences which result in the environment as a direct result of the proximal actions taken by the learning agent. In the canvas painting example, for instance, the distal target behavior sought by the trainer would be the painted square, i.e. a visual result that is far removed from the motor control commands used to generate it, hence the term "distal". The series of arm joint angles required by the robot to attempt such a goal would constitute proximal actions. In the current scenario, correct proximal targets are not available for training the learner (i.e., there is no teacher contribution that explicitly moves the arm through the desired movement sequence which can be used for training.) The desired outputs sought by the teacher (e.g., the intended square in this case, perceived visually) actually exist in the output parameter space of the environment function rather than in the learner's action space (e.g., robot arm movements.)

Reinforcement learning strategies are often used to handle adaptive learning problems as the environment function is generally undefined or very difficult to characterize. Very effective methods have been developed which demonstrate learning optimal to near-optimal policies exclusively through interaction with an external environment ([2],[31],[52],[53],[57],[58],[63]). Even so, reinforcement learning has its drawbacks and is far from being a perfected science. It can be very difficult for an agent to learn even a good policy, much less the optimal policy, in complex and unfamiliar environments. This is even more so the case when the reward function, which drives learning, is designed with little or no a priori teacher bias. Many of the most popular reinforcement learning techniques studied today are not guaranteed to converge to optimal policies.

Traditional supervised learning methods have stronger convergence assurances than reinforcement learning but are ill-suited for use in a distal environment. Jordan, et al. [23] demonstrates that supervised learning can be used to train a learner situated in a complex environment where only desired distal targets are available for training. In this framework, another neural network (the "forward model") can be set in serial with the learner and be trained to emulate the environment. The additional neural net can then, in turn, be used to assist in training the learning agent using the target distal outputs provided by the teacher. Variations of this methodology of learning have been shown to be particularly effective in a variety of domains. One such domain includes studies in constructing computational simulation of brain function as it has been shown that human brains utilize similar "forward models" in many aspects of motor task learning and development ([4],[15],[29],[70],[71],[72]) (e.g., motor control, etc.) Some training of distal learning agents to produce sequences or strings of actions is also demonstrated for non-sequential neural network learners [24]. However, these methods have not been effectively studied in training distal learners with recurrent links. Moreover, such recurrent networks should potentially be capable of generating varying length series of discrete time actions even when provided with a single input stimulus.

Unlike existing distal learning methods designed for non-recurrent neural network learners, the methods presented here are developed in order to train recurrent neural networks which utilize

some type of history in the form of a context vector. Using the latter, a neural network will be better equipped to learn the appropriate sequential proximal behavior given only a static input vector and without being provided with information about the current state of the world. Such a distal learner requires only a similarly designed recurrent network for its forward model and the desired distal sequences for training. Such an architecture can be useful in that, for one, should the current state generator (e.g., camera in a vision system, audio sensor) fail or be removed, good sequences can presumably still be learned and completed as the learning agent can be guided by its own memory. Also, the use of an exponential decay memory layer (described in detail in Section 2.2) in many recurrent neural network implementations may effectively supplement or even replace the current state information used to drive existing distal supervised learning implementations.

1.1 Goals

The goal of this research is to develop a system that can train recurrent neural networks situated in a complex environment when provided with desired distal target sequences to drive learning under the assurances afforded to a supervised learning framework. Not only could this work expand the use of recurrent neural nets in more complex domains, but it may even improve on existing domains of distal sequential learning tasks previously handled by reinforcement learning and non-recurrent distal learning implementations.

Recurrent neural networks have been found to possess tremendous value in many fields ([35]). They have been used successfully to solve or address many problems such as robot control in producing time-series behavior. These recurrent neural networks have been shown to exhibit useful qualities and properties including the robustness commonly found in many instances of neural network applications. Also, they exhibit forms of fault tolerance and can be shown to generalize very well using only training data.

However, many problems that exist in the real world are not framed in the same manner as that presently set up for recurrent neural networks. As in any supervised learning method, the teacher or

”supervisor” must have available a priori all sequences the recurrent neural network should know by the time training has concluded.

In many real world complex problem domains, the time-varying sequential behavior worth learning takes place in some external environment. For example, Jordan ([24]) describes a case where a person is required to learn how to propel a basketball into a basket (Figure 1.1). All that is known to the person (learning agent from here on) beforehand are the necessary inputs and desired distal outcomes of the environment. In this example, the input to this learning agent would comprise the intent to shoot the ball into the basket, and the position of the ball in his/her visual field could comprise the current state of the learning task. Ultimately, the desired distal outcome in the environment sought by the agent should comprise the sights and sounds of the basketball going through the hoop. What the learner in this task must somehow acquire is the necessary series of arm motions required in order to successfully accomplish this task.

In order to handle the training of neural networks to operate in environments like the one described above, Jordan suggests the creation of a separate neural network (termed a forward model) which can be trained through its own interactions in the environment to mimic the latter’s mapping of the learner’s proximal actions to distal consequences. When completed, this forward model neural network can then be employed to assist in training the actual learning neural network of interest. This use of a second neural network to assist in training the original untrained feed-forward neural network acting in the environment is referred to in general as *distal* supervised learning.

Jordan uses some good applications to demonstrate the actual learning of time-varying proximal behavior in the output space of the learning neural network in order to accomplish the learning of the task. At this point, many researchers have followed this paradigm to develop similar systems capable of addressing some very interesting distal problems ([27], [38], [42], [60]). This method is a very effective way of solving the inverse modeling problem, where, once trained, the learning neural network in question can be characterized as the *inverse* function of the unknown environment.

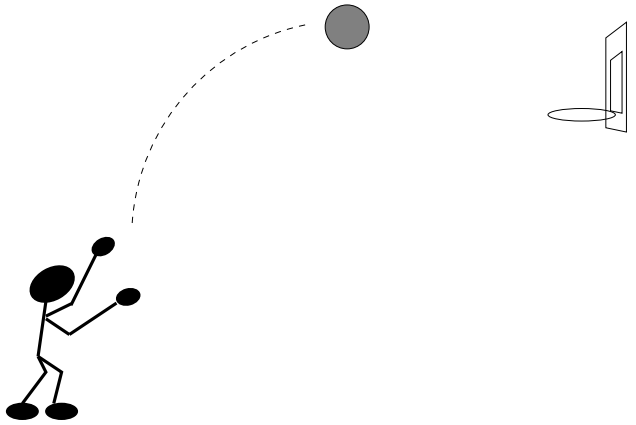


Figure 1.1: A basketball shooting example for using the distal supervised learning paradigm.

Recurrent neural networks contain recurrent links between neural elements in order to encourage time-varying behavior based on action history. This information can be taken from the previous step or even a history of previous actions in the form of an exponential trace memory. As already mentioned, such recurrent neural networks have been shown to be very useful in real world applications. To my knowledge, the distal learning paradigm has not been extended to training recurrent neural networks.

Also, of particular interest to this study is not merely the production of time-varying sequential behavior through interaction in the environment, but sequential behavior that can result from just a single static input stimulus (e.g., a picture or a single goal position.) In typical studies in which the acquisition of correct sequence generating behavior is the goal, the input stimulus will change with every new time step or subsequent action of the learner. It has been shown that some trajectory learning behavior can be demonstrated without the use of recurrency, but that is while using current state updates from the environment at every step of the action-generating process. The typical distal learner relies heavily on such updates to drive its neural network to generate its next action or output. Here, a paradigm is sought that can use just a single input vector (which can be thought of as a single plan, a thought, or intention of the system) in order to generate some time-varying sequence of proximal actions which can yield a very specific trajectory output in environment space.

Past literature has not fully addressed the problem domain of training a neural network to produce the appropriate sequential behavior necessary to yield a very specific trajectory in the environment space from a single static input stimulus. This dissertation addresses this particular problem and maintains that adding recurrency to neural networks trained in the external environment of interest can be the best course of action in learning to produce the correct proximal sequential behavior from learning agents given only a single input or intention from which to work.

Jordan [24] briefly suggests how one might reconfigure his distal supervised learning framework to potentially learn specific trajectories in an external environment. His modification, however, still relied heavily on using a steady stream of current state updates from the environment to determine subsequent actions local to the agent. In addition, this modification still did not address the handling of distal sequence generation tasks which require only single input stimuli to generate multiple actions and, hence, multiple consequences in the environment. Here, recurrency is added to the original distal supervised learning framework at the level of the distal learner of interest as well as its forward model in order to further facilitate learning and to add capabilities and functionality that could not be easily addressed under Jordan's initial suggestion.

1.2 Specific Aims

The specific aims of this study are as follows:

1. Expand the capabilities of the existing distal supervised learning paradigm to manage training of often used recurrent neural architectures.

2. Create a model of the information processing done by cerebral cortex in learning to produce the correct motor phoneme sequence response for a desired stored representation of the intended word in associative memory. The capacity of this system to readily and efficiently learn sequences in an external environment as well as the presence of short term memory inherent in the recurrency of this system will be an important factor in creating such a model.

The key generalization here is to generate a sequence of correct outputs for a single given fixed input stimulus.

3. Create a SOM that can process and store phoneme or vector sequences such that unique activation patterns for each sequence will be obtained. In designing a more efficient sequential SOM model for this study, I incorporate modifications in the SARDNET SOM architecture that consider which particular input vectors are most expected (candidate vectors) in calculating the correct SOM output. These modifications in unique mapping capability will lend themselves greatly towards enhancing the capability of my model to demonstrate a simple form of the phoneme sequence acquisition task previously described. Here, the map organization and uniqueness of the modified SARDNET output will be analyzed and compared to that of the original architecture.

4. Incorporate varying recurrent network architecture types and training methods into a recurrent distal supervised learning system. The recurrent network used primarily in this study, often termed the Jordan network [23], is only one of many different types of recurrent network architectures ([13],[8]). Numerous recurrent network training methods exist as well ([6],[35],[37],[40],[42],[43],[67],[68]) and are used successfully in varying learning tasks and problem domains. By implementing other recurrent network types and contrasting their performances, pros, cons, etc., I hope to ascertain which blend of recurrent architectures, used in learner and forward model alike, could be utilized in maximizing performance on various types of training tasks and problems driven by desired sequences obtainable through the environment.

1.3 Contributions

The primary contribution of my work is the modification of the existing distal supervised learning architecture to allow training of recurrent neural networks which operate in external environments

(Sections 3.1-3.3). The current distal supervised learning architecture, developed by Jordan [24], was originally designed to train single input/single output standard feed-forward neural networks from desired outcomes that should result from interactions with an environment. Without consistently being informed of its current state in the world after each action it took, a traditional distal learner would be incapable of performing sequence generating tasks from a single unchanging input stimulus, whereas my approach can handle such situations. I demonstrate the utility of the modified distal learning framework by training a recurrent network in a sequential environment called the concatenation environment whose behavior is well understood.

Second, just as in typical non-distal sequential learning tasks, recurrent networks can be useful in their utilization of previous output memory in generating time-varying behavior while operating in a distal setting. They become especially useful when only a single static input vector is supplied to the learner as it is in distal sequence generation tasks. Section 3.4 describes a method which I adapt from a strategy referred to as teacher forcing, often used to improve training in standard recurrent networks, for use in recurrent distal learning systems. Through this method, recurrent distal learner actions are made approximately more ”correct” before being stored in memory in order to hasten the training process. Though the actual correct action sequences are not available for training, these approximated entries for memory updates tend to demonstrate noticeably improved training results.

Third, once trained, I developed a self-organizing map to represent associative memory and uniquely characterize a sequence of auditory feature vectors based primarily on the SARDNET SOM architecture [21]. Though shown in previous studies to be useful in providing unambiguous activation patterns from differing input vector sequences, some measure of ambiguity still existed with the original SARDNET which could potentially be detrimental in the phoneme sequence generation process previously described. In this work, I develop a modified method of producing activation patterns in the SARDNET SOM, called the candidate-driven method (Section 4.3), which considers the closeness of the most likely candidate vector to the responsible input vector, as well as the proximity of the current node to the winning node in the SOM’s output lattice, in

determining a meaningful real-valued output between 0 and 1 rather than just a strict binary 0 or 1 value as in SARDNET.

Fourth, I implemented a prototype non-recurrent distal learning system capable of training neural networks to generate single motor phonemes responsible for yielding desired auditory phoneme vectors from single input vectors (Section 5.2.) A key problem encountered in this implementation was how to map outputs to the environment into their corresponding distal feedback. In order to construct the motor-to-auditory mapping required for this single phoneme acquisition system, I devised a method for creating a smooth and continuous mapping from a finite number of paired vectors (Appendix B.) As a result, my implementation is able to take any vector in the space of motor phonemes, including any of the motor phoneme vectors listed, and generate a reasonable facsimile of an auditory vector feature for use in this study.

Fifth, to test this modified system on a substantial distal sequence learning problem, I designed a simplified simulation that takes as inspiration the manner in which humans produce phoneme sequences in speech function acquisition, and looks to see if a recurrent neural network can be trained in similar fashion (Section 5.3.) In order to create such an ambitious simulation, a sequential environment is constructed that accepts a sequence of motor feature vectors and responds with a sequence of corresponding neural activity patterns emanating from associative memory. This complex sequential environment is a composite of two non-linear component mappings: 1) a mapping which transforms a sequence of motor phoneme feature vectors into corresponding heard auditory vector sequences, and 2) a self-organizing map (SOM) representing associative memory of auditory sequences.

1.4 Dissertation Organization

The rest of this dissertation is organized as follows: In Chapter 2, previous works which were pertinent in the creation of the architecture addressed here are reviewed. In Chapter 3, self-organizing maps (SOMs) which are designed to accept and uniquely characterize sequential, and not single,

inputs are discussed. In creating a computational model of sequential cognitive function, a viable model of cortical map activation is absolutely necessary. In the case of simulating phoneme sequence, or spoken word, acquisition, some approximation of associative memory responsible for storing of previously heard words should be addressed. Self-organizing maps (SOM), introduced by Kohonen in ([26]), were created in part to attempt to model the map formation found in the human brain and have been studied extensively for years. Few projects have addressed the need for SOMs to adequately store sequential inputs in a manner in which each unique sequence will result in a unique set of activations in the SOM. The one-shot, multi-winner SOM (Schultz [54]) and the SARDNET self-organizing map (James [21]) are two very promising methods, but fall short of guaranteeing 100% uniqueness in mapping sequences to unique SOM activations that are required for this particular study. Also in this chapter, I address the modification I devised in making one such construct more appropriate for this study.

In Chapter 4, I detail my own work in developing a type of distal recurrent supervised learning architecture which makes use of time-delay links between layers of computational processing units in both the distal learner and the forward neural model. Specifically, this architecture is capable of enabling distal learners to handle distal sequence generation tasks using only single input stimuli and no current state updates in order to drive themselves in determining subsequent actions. In Chapter 5, I discuss the results of the newly created architecture presented in Chapter 4 primarily as an application to the study of the acquisition of the cognitive ability of phoneme sequence generation. One of the more common uses of traditional distal supervised learning at present lies in the creation of computational models of human cognitive task acquisition ([15],[29]). Modeling acquisition of speech and motor control functionalities, in particular, are domains which are active topics of study ([15],[17],[29],[70],[71]). One intention of this study is to increase the capabilities of such distal supervised learning models of cognition to encompass more cognitive phenomena said to occur based on the most current neuroscientific studies.

Lastly, Chapter 6 discusses the ramifications of the new distal sequential architecture introduced in this dissertation and addresses potential future directions to improve it, its use in mod-

eling cognitive sequential tasks such as phoneme sequence generation, as well as in various other problem domains.

Chapter 2

Background

2.1 Feedforward Neural Networks

2.1.1 Description

The creation of neural networks is motivated by theories of how the interactions among neuronal cells in the brain are thought to generate cognitive functions. From what we gather from past neurobiological studies, neurons act to either fire or not fire if they receive enough overall excitation from other neurons that synapse to them. Put another way, intelligent function emanating from the brain is considered to be a result of the total cooperative interactions of neurons in the brain based on inputs it receives from input stimuli. Map formation in the cortex is another consequence of group neural interactions in the brain.

Some of the earliest neural networks came in the form of *perceptrons* which essentially consist of one layer of computational "neurons", each of which receives real-valued input from all input elements to the system via weighted connections, w_{ij}, where i and j reference neural elements and input elements, respectively (Figure 2.1).

In essence, the set of weights, represented by weight vector, $\vec{\mathbf{w}}$, determined the output of the perceptron. In order to ascertain the best weight vector, $\vec{\mathbf{w}}$, a very simple, iterative procedure was developed ([51].) The single-layered architecture of the perceptron, however, hindered its computational power as it was shown to be able to handle only linearly separable relations between

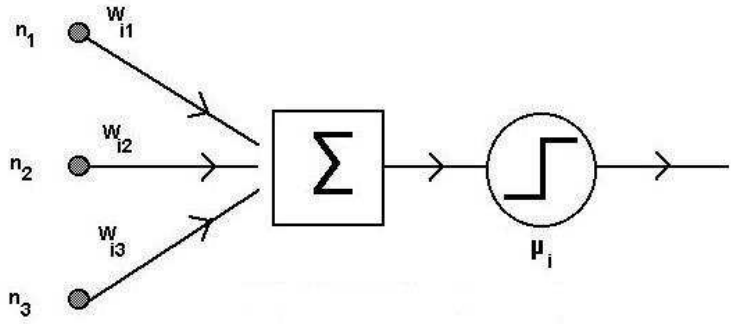


Figure 2.1: An example of a typical perceptron set up.

inputs and target outputs ([34]). This insight seriously limited the effectiveness of neural network research for some time. By equipping neural networks with another hidden layer of neural elements between the layers of input and output nodes (see Figure 2.2), it was later determined that perceptrons can be made to classify linearly and non-linearly separable tasks alike. Furthermore, by changing the output functions of the neural elements from a step function to smooth and differentiable step-like functions, finding the best set of weights becomes an exercise in determining the weight vector which minimizes the following error function, J($\vec{\mathbf{w}}$):

$$J = \frac{1}{2}(\sum(t_i - o_i)^2)$$

It was shown that such a *multi-layered* perceptron could approximate any differentiable function when given enough input/ output examples whether linearly separable or not.

2.1.2 Supervised Learning (Back-propagation)

In a supervised learning framework, there exists a learning agent that can be characterized as some function $y = h(p, \vec{\mathbf{w}})$, where $\vec{\mathbf{w}}$ represents the internal state of the learner (in this case, the weight vector in a neural network), p is some input vector and y would be the resulting output vector. Given some set of target input/output pairs $\{(\vec{\mathbf{p}}_i, \vec{\mathbf{y}}_i^*)|1 \leq i \leq n\}$, the task of the learner is to adjust the parameter vector, $\vec{\mathbf{w}}$, in such a way as to minimize the performance error between target output

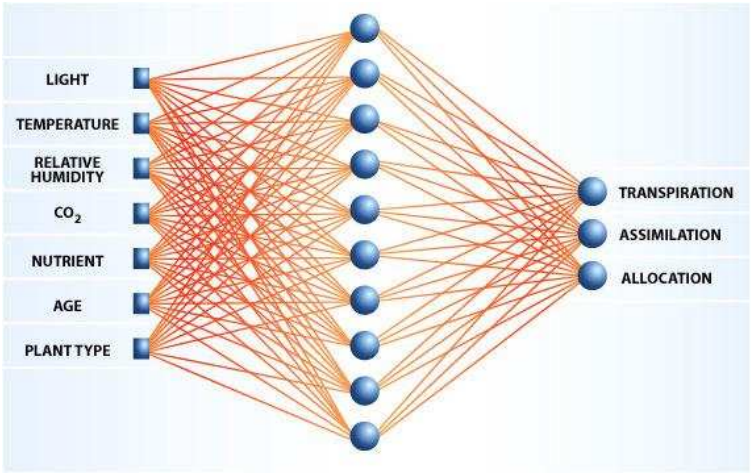


Figure 2.2: Example of a standard multi-layered neural network architecture at work (taken from http://aemc.jpl.nasa.gov/activities/bio_regen.cfm)

vector $\vec{\mathbf{y}}_i^*$ and the neural network learner's own actual output vector, $\vec{\mathbf{y}}_i$, given input vector $\vec{\mathbf{p}}_i$ (see Figure 2.3). The expected performance error, J, used to judge the effectiveness of the learner's training can be formulated as follows :

$$J = \frac{1}{2}E\{(\vec{\mathbf{y}}_i^* - \vec{\mathbf{y}}_i)^T(\vec{\mathbf{y}}_i^* - \vec{\mathbf{y}}_i)\}, 1 \leq i \leq n, \tag{2.1}$$

However, rather than take into account all desired input/output pairs in determining the cost, a more instantaneous online evaluation for the n-th input/output pair can be done as follows:

$$J_n = \frac{1}{2}(\vec{\mathbf{y}}^*[n] - \vec{\mathbf{y}}[n])^T(\vec{\mathbf{y}}^*[n] - \vec{\mathbf{y}}[n]), \tag{2.2}$$

In order to change the weight vector, $\vec{\mathbf{w}}$, of the learner to minimize this cost function, the gradient of J with respect to $\vec{\mathbf{w}}$ can be approximated as follows :

$$\nabla_{\vec{\mathbf{w}}} J_n = -\frac{\partial \vec{\mathbf{y}}}{\partial \vec{\mathbf{w}}}^T (\vec{\mathbf{y}}^*[n] - \vec{\mathbf{y}}[n]), \tag{2.3}$$

Knowing this, the weight vector at time n, denoted as $\vec{\mathbf{w}}_n$, can then be adjusted using this equation:

$$\vec{\mathbf{w}}[n] = \vec{\mathbf{w}}[n-1] - \eta \nabla_{\vec{\mathbf{w}}} J_n, \tag{2.4}$$

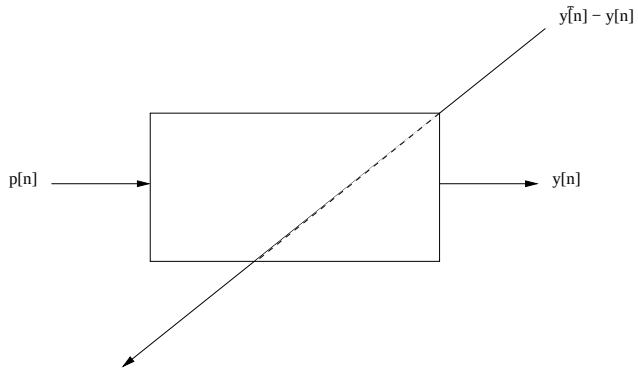


Figure 2.3: Visual demonstration of standard back-propagation procedure. The error-back propagation procedure can move a multi-layered feedforward neural network (denoted by the box above) incrementally towards producing some desired behavior given an input **p**[n] and its corresponding target output **y***[n]. Here $0 < n < k$, where k signifies the number of input/output pairs used to train the neural network. Over many training steps (epochs), the weight parameter vector **w** (not shown) of the neural network is adjusted using the difference vector between the target output **y***[n] and the actual neural network output **y**[n], where y[n] = h(**p**[n],**w**).

where η is a parameter which controls the rate of incremental weight vector updates. This is the basis of most gradient descent methods of supervised neural network learning.

The *back-propagation* method (Rumelhart [51]) is merely a form of gradient descent designed to find the local minimum of the error function, J($\vec{\mathbf{w}}$), over weight vector space. Figure 2.3 demonstrates a key component of the back-propagation procedure, where the difference between target and actual outputs is propagated back through a neural network module to change the weight vector incrementally into one which more closely approximates the desired output. As such, solving for the best set of weights for the neural network or multi-layer perceptron becomes a matter of finding the weight vector, $\vec{\mathbf{w}}$, which minimizes J.

The error function at this point may be minimized by approximating the gradient of this function and running some form of hill descent procedure which can provide a weight vector which provides a gradient as close to zero as possible. The exercise for determining such weights now becomes the task of finding the set of weights which minimize this function. Since the landscape

of the error function is unknown, the gradient is approximated roughly given the current weight vector and an iterative procedure of gradient descent is employed in an effort to find the weight vector which yields the minimum of the error function (see Table 2.1). This method, however, poses problems where it often may converge to some local minima of the function instead of the global minimum which would give the best answer. Gradient descent neural network training methods require approximating the gradient of the error function at the point in the weight space where the neural network is currently, and in changing that weight vector in the negative direction of the gradient. This, thereby, has the effect of moving it, in theory, closer to the local minima of the error function. In many complex domains, the local minima require a great deal of computational effort to be found and are often not sufficient in learning the task presented to the neural network when found.

Apart from standard hill descent techniques, other types of weight space selectors have been sought to find the global minima. Some such methods include genetic algorithms, evolutionary programming, support vector machines, etc. However, a sizeable amount of the energy spent in trying to solve this problem has been used to develop more efficient types of gradient descent methods. Many early devices sought to improve gradient descent back propagation by manipulating or adjusting the learning rate in order to more quickly find the local minimum. Other methods being developed sought ways to avoid getting trapped in local minima en route to better solutions or even, ideally, a global minimum ([49], [6], [41]).

Some very powerful methods utilize the gradient information to use a more informed, pertinent search for the global minimum given a weight-by-weight adjusting scheme or even a learning rate per each individual weight term rather than adhering to one single learning rate for the entire gradient computed term. These methods require use of the gradient just as an indicator for direction. The actual descent is regulated by assigning an individual learning rate to each weight vector and raising or lowering them according to the information received about the error function landscape. Two of the most popular methods which operate in this fashion include Quickprop (Fahlman [14]) and RPROP (Riedmiller et al.[49], Igel et. al [19].) Presently, many such gradient descent methods

Error Back-propagation Procedure
repeat for each training pair, *n*:
1) obtain input **p**[n] and target output $\mathbf{y}^*$[n].
2) compute neural net output, **y** = h(**p**[n],**w**).
3) compute error vector at output layer : $\Delta_i = \mathbf{y}^*[n] - \mathbf{y}$.
4) update all weights leading to each unit in the output layer: $w_{ji} = w_{ji} + \alpha a_j f'(in_i)\Delta_i$
5) **for each** subsequent layer,
- compute new Delta values for new layer: $\Delta_j = f'(in_j)\sum w_{ji}\Delta_i$
- then use it to update weights to the next layer: $w_{kj} = w_{kj} + \alpha a_k \Delta_j$
end
6) repeat from step 1) until:
- performance criteria is met or
- number of training loops (epochs) is reached.

Table 2.1: Error back-propagation procedure for training neural networks

continue to be developed in seeking to enhance the way in which optimal weight vectors can be found in the effective training of neural networks.

Effective adaptive learning schemes have been also developed which, once given the performance of the neural network immediately following a weight change, will automatically increment or decrement the learning rate of the neural network training algorithm and repeat the evaluation until only improvements result. Also, there are methods which seek to substantially change the back propagation method as it was originally designed. In one previous study Joost [22] argued that the standard error function typically used in back-propagation is flawed in that it is polynomial (namely binomial) in structure and, hence, encounters the pitfalls inherent in executing the

gradient descent of such functions. For one primary pitfall, he notes that in following the opposite direction of the gradient for a binomial function, successive gradients themselves approach 0 as the minimum draws close, thereby substantially slowing and inhibiting the search for the global minimum. Joost advocates the use of a different type of error function which is non-polynomial in structure and will not slow or diminish to zero the closer it gets to the local minimum. The new error function suggested is based on the conjugate gradient function in order to circumvent those pitfalls (Joost [22]). He argues that it works better and bypasses the shortcomings of the polynomial error function discussed previously.

2.1.3 Feedforward Neural Network Strengths and Limitations

There are, however, limitations to the training of these neural nets. For one, there is always the possibility of overfitting the weights of the neural network. In this situation, the neural network may be trained to learn the relation between input/output pairs provided by the supervisor but not be capable of generalizing from unseen inputs to new outputs. If it is the case that too many neural elements are placed in an intermediary or hidden layer, the neural network may become *over-trained*. By this, it means such over-partitioning of the input space may result in training the neural network to learn only the specific relationships between the training inputs and their corresponding target outputs and little else. When this occurs, the neural network can be so specific that it would be incapable of correctly categorizing other inputs not explicitly provided in the training data. This would not be beneficial to one who is looking to train the neural network to be able to classify some general relationship between inputs and outputs.

When the neural network back-propagation method is run, the method is iterated many times with each pass through the training data being called an *epoch*. When the training is complete (say over tens of thousands of epochs) the multi-layered neural network should know the inputs and outputs that the teacher provides. Furthermore, to ensure that neural network has not only memorized the training data, but has also learned to generalize effectively, one can provide validation data on which to test the neural network throughout training. Here, validation data are input/output pairs

which also share the same relationship as those pairs in the training data but are withheld for later verification purposes. If the performance of the neural network should be measured (where root mean squared error is one measure of performance success) then the validation data should score relatively well with the neural network while training if the relation to be learned is to be ensured or guaranteed to be found. At this point, if it is not the case that the RMSE is low compared to that of the training data, overfitting has occurred. To avoid such a circumstance, there are many things a trainer may need to be wary of when training a neural network:

1. not to make the number of hidden elements too high. If this is made too high, the input space will be partitioned far too much and the task or relation can become very specific toward the input/output training data. By keeping the number of hidden elements low, one can ensure that very general partitions can be found to approximate well the relation sought.

2. to provide very good representative training data for the function to be approximated. If there are major holes in the input space which cannot be accounted for in the training data, learning the appropriate function would be very difficult.

Another limitation seen in standard multi-layered feedforward neural networks lies in the inexplicable manner in which it encodes its approximation of the unknown function. It is quite possible for the neural network to be trained to correctly approximate the relation suggested by the training data provided to it by the trainer. However, there the ability for researchers to actually go in and extract what knowledge the neural network has actually acquired is severely limited indeed. As such, though neural networks can be very powerful tools as function approximators or classifiers, they are not very effective tools for data mining or knowledge discovery.

As for strengths, multi-layered feedforward neural network architectures have been shown to be extremely effective in approximating unknown functions. As will be seen later, a neural network can approximate the workings of some unknown system and ideally, if trained efficiently, can be used to forecast reasonably good guesses to outputs of some previously unseen arbitrary input. This ability to generalize given only desired input/output pairs makes applying neural networks

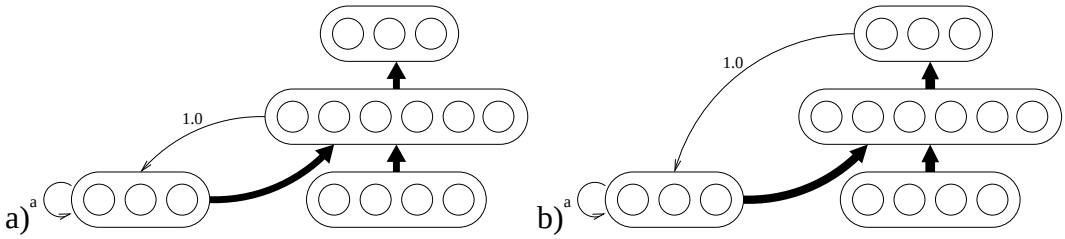


Figure 2.4: Two popular implementations of recurrent neural networks : the Elman network (left) and the Jordan network (right). Ellipsoids in both cases denote layers consisting of neuronal processing units (shown as circles). In either graph, wide arrows denote full connectivity via weighted links amongst all units from an originating layer up to those of its destination layer. Thin arrows denote a direct copy from a single unit in the originating layer to its corresponding unit in the destination layer multiplied by some constant (default set to 1.0 .) The two implementations differ primarily in that the activations from the neural network's hidden layer are accumulated by the memory layer for the Elman network while the memory layer in a Jordan network copies the activations of the neural network's output layer. Both neural network implementations can utilize an exponential trace memory vector with decay constant, a, for use in learning to produce desired time-varying output behavior.

very attractive in countless complex problem domains which grapple with unknown relations and functions. Also, in terms of strengths, the neural networks can be used in developing very simple models of human brain dynamics and function which can help shed light on the inner workings of the human brain. In fact, many such brain computational models have indeed been developed in attempting to capture brain phenomena documented in existing neuro-biological literature. These same computational models can serve as effective tools in developing understanding and treatment for afflictions of the brain ([46], [47])

2.2 Neural Network Sequential Processing

Neural networks have traditionally been used in learning tasks in which one input vector should yield a single output vector. However, in some domains, the desired output would be in the form of a series, or sequence, of vector outputs which vary over the course of discrete time steps. In order to achieve this result, recurrent links can be introduced within a neural model between neural elements in such a way that, even if the input vector should be kept static, a neural element can yield a different output value with each subsequent time step. Figure 2.4 shows examples of such neural network architectures.

There are various methods researchers have used in attempting to create neural models which take into consideration a history of states in order to determine the subsequent output. Some architectures attempt to "parallelize" time by placing simultaneously in the input layer a finite number of previous network inputs, outputs, and/or states which can then be processed by a subsequent hidden or output layer. An example of such a recurrent neural network architecture is the NARX (non-linear autoregressive with exogenous inputs) network in which a history of the previous q inputs, $\{u_n, ..., u_{n-q+1}\}$, and q network outputs, $\{y_n, ..., y_{n-q+1}\}$, comprises the input layer which is presented to a multi-layered perceptron to eventually yield output y_{n+1} ([8],[37]). In this manner, the NARX model can be trained to consider unmistakably the history of input/output pairs which transpired previously in order to determine the subsequent output. This architecture, however, can lead to increased complexity of the learning task as the input space increases linearly with input and output vector lengths through user-specified history length, q.

One well known recurrent network architecture is the Jordan network [23] which has recurrent links from the output layer to a memory layer that is situated at the same level as the input vector and has its own set of weighted links to the next hidden layer (see Figure 2.4). Neural elements in the memory layer generally have self-recurrent links which utilize a decay $0 \leq \alpha < 1$ term which has the effect of accumulating a history of its actions over time. Such a grouping of memory processing units can be referred to as an *exponential trace memory*.

Giving initial memory $x(0)$ some known initial assignment such as $x(0) = 0^n$, for instance, the output dynamics of a simple two-layered Jordan network may be characterized by the following equations :

$$h(t) = f(W_u u(t) + W_x x(t)), \tag{2.5}$$

$$y(t+1) = g(W_h h(t)), \tag{2.6}$$

$$x(t+1) = y(t+1) + \alpha x(t). \tag{2.7}$$

where *x* is the exponential trace memory vector, *y* is output of the recurrent network at discrete time step, *t*, and *h* is the hidden layer. Functions *f* and *g* are the activation functions for the hidden and output layers, respectively. Terms $W_u, W_x,$ and W_h describe vectors corresponding to weighted connections emanating from the input, memory, and hidden layer vectors, respectively, to the appropriate subsequent layer. This type of recurrent network architecture is appealing in that varying length output histories can be retained and considered in estimating the desired output at subsequent time steps without having to increase the dimensionality of the memory in the input layer.

The Elman network is yet another instance of a recurrent neural network which effectively uses an exponential trace memory vector in the input layer. Where this architecture differs from that of a Jordan network is that the exponential trace memory is used to store a history of activations from some intermediate, or hidden, layer of processing units as opposed to the output layer (see figure 2.4).

Similarly, the output dynamics for a simple Elman network can be described as follows:

$$h(t) = f(W_u u(t) + W_x x(t)), \tag{2.8}$$

$$y(t+1) = g(W_h h(t)), \tag{2.9}$$

$$x(t+1) = h(t) + \alpha x(t). \tag{2.10}$$

In the case of exponential trace memories as they are used in Jordan and Elman networks, input space size is not as significant an issue as it is for the NARX architecture and models like it.

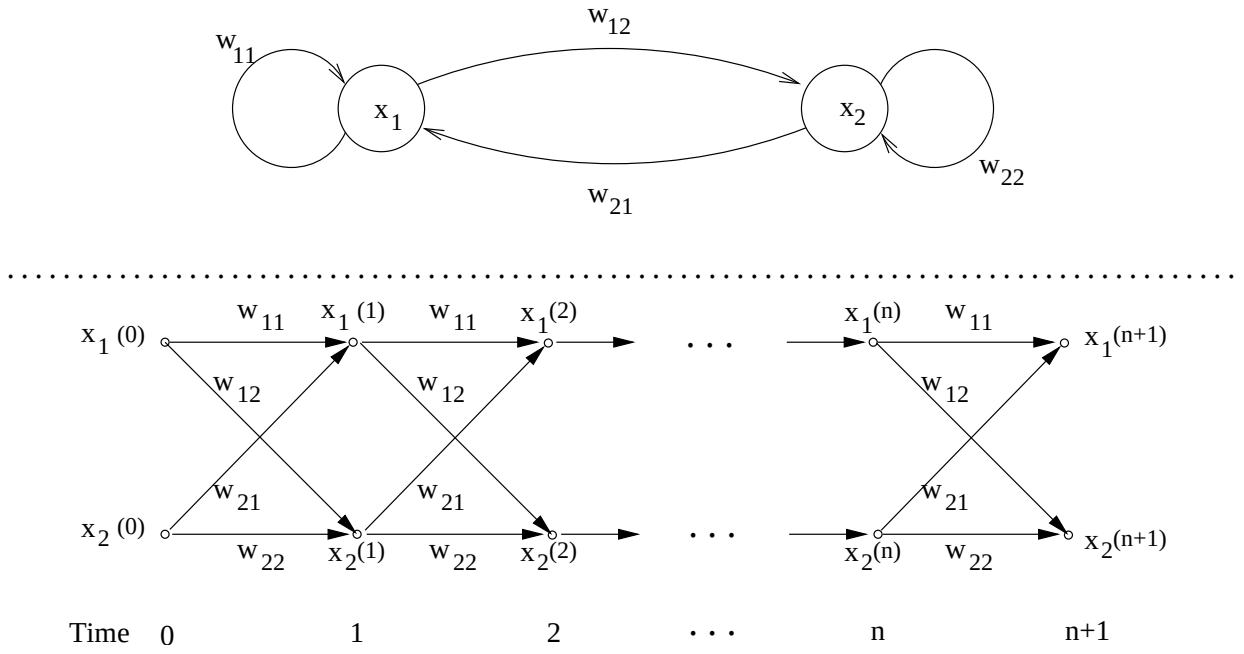


Figure 2.5: Recurrent network unfolding example provided in Haykin [18]. (Top) Simple recurrent network composed of two nodes having weighted connections to themselves and each other. (Bottom) Equivalent non-recurrent multi-layered feedforward network capable of producing sequences of length n. Consequently, modern back-propagation techniques can then be derived for the latter network to yield back-propagation in time learning rules.

However, to what history length the exponential trace memory vector can be effective in producing the remainder of a target sequence can be an issue. This is because the effects of states stored from previous time steps can vanish very quickly as the exponential term is continually applied to the memory vector. In addition, this type of memory vector is quite limited as to its ability to recall the sequence of states it was given to store.

2.2.1 Training Methods for Sequential Neural Networks

Methods for training recurrent neural networks such as those described previously have been developed and refined for years. One method training recurrent neural networks is known as back-propagation in time [64]. By “unfolding” a network’s recurrent links and transforming it to re-

semble a standard, single pass multi-layered feedforward neural network, very effective weight change rules can be inferred in much the same way as those developed for less dynamic, yet more heavily studied non-recurrent neural network architectures (figure 2.5). More specifically, back-propagation methods initially used exclusively for feedforward networks can be extended for training recurrent networks. Variations of back-propagation in time methods are described in greater detail by Williams et al. [68].

Methods have also been developed to improve existing sequential network learning techniques. Teacher forcing ([67],[69]) is one such method. Here, the "teacher" can clamp onto a layer of processing nodes (i.e. the memory vector), when available, the desired activation at that discrete time step, t, rather than the erroneous activations that occur amidst the early stages of training. This process can be implemented by supplanting Equation 2.7, the original memory update equation for Jordan networks, with following equation:

$$x(t+1) = y^*(t+1) + \alpha x(t). \tag{2.11}$$

where $y^*(t+1)$ is the target output vector at time t+1 provided by the supervisor as opposed to the actual output, y(t+1), from the recurrent network itself supplied via its recurrent links. Using this method during training in the described manner tends to assist the recurrent network to converge faster and more readily. A new form of teacher forcing I develop is introduced in the methodology in section 3.4.

2.2.2 Time Delay Memory Structures

In addition to the exponential decay memory structures introduced previously, another popular form of memory structure exists in delay line structures used early in recurrent network design. Using this architecture, at the current time step, t, the set of activations from some pre-determined set of nodes (generally some hidden or output layer in a multi-layered feedforward recurrent network) are copied directly to some memory module of nodes. The resulting module can then be used at the subsequent time step, t+1, as input to the network through trainable weighted connections

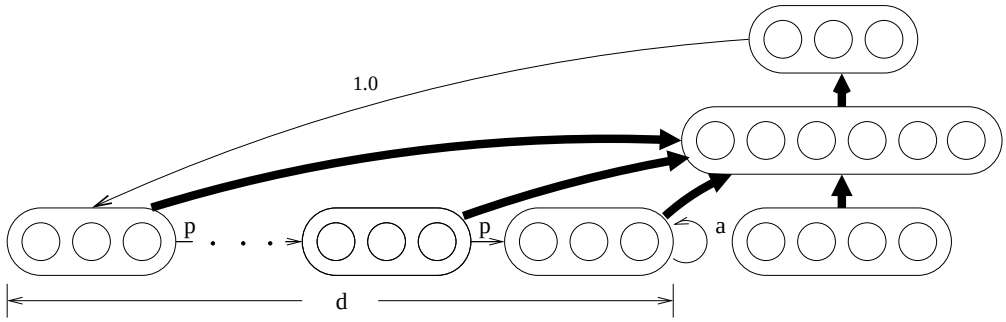


Figure 2.6: A recurrent Jordan network using d time delay layers. The node activations at memory delay module, k, is determined at each discrete time step as the product of the contents of the previous delay layer (k-1) and the propagation term, $0 < p \leq 1$. In addition, the final delay layer here uses a decay rate, a, such that the memory structure retains exponential trace history of actions once the k-sized window is exceeded. Setting α = 0 restricts this memory mechanism to being a sliding window of size d, which is very common amongst memory delay recurrent neural networks in prior studies.

along with the already present input vector.

Multiple memory modules can be incorporated into the recurrent module as well, separated by delay lines from a prior memory module of the immediately previous time-step. Here, memory contents from the (t-i)th set of activations are copied to the next memory module representing the prior (t-i-1)st time step of activations before itself receiving the set of activations contained in the module representing subsequent time step (t-i+1). This series of delayed activations can be made arbitrarily long based on the goals of the recurrent neural net designer. What results, unlike in the case of the exponential decay memory vector for a delay window length, $d > 1$, is an absolute record of previous actions is taken which can be utilized by the recurrent neural network with a greatly reduced risk of ambiguity or misinformation to within d prior times steps.

One problem that results, however is the window length, d, of memory observation is always restricted to some finite number, and any memory activations recorded d+1 time steps prior will be lost to the recurrent neural network, essentially falling off edge of the proverbial “sliding window” of action history. One way this could be addressed is to make the final (t-d)th memory module

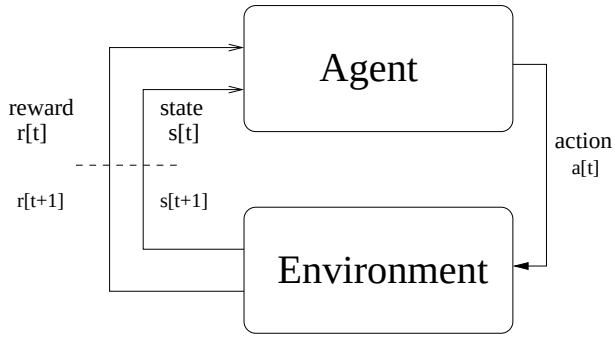


Figure 2.7: Reinforcement learning framework.

an exponential decay memory vector just as previously discussed. In this manner, the recurrent network readily remembers and can act on outputs it made prior to the t-d-th time step in fostering better subsequent decision-making as opposed to forgetting that information entirely (Figure 2.6.)

2.3 Reinforcement Learning

Reinforcement learning is generally the method of choice when training agents to acquire good-to-optimal behavior in an external environment. In this framework (see Figure 2.7), an agent, once presented with the current state, generates an action in the environment. The environment then returns some numeric score to gauge the effectiveness of the action performed. The controller must then modify its own internal state based on this reward/penalty signal such that, during this learning stage, it would be more apt to select this action given the same input if a high score (i.e. reward) was achieved. Similarly, it should be less apt to select this action if a low score (i.e. penalty) resulted. The goal of the learner is to construct an optimal policy which it could use to generate behavior which would eventually yield the optimal or desired outcome at some point in the future.

Many successes have resulted in the use of the reinforcement learning techniques. Two very early successes include Samuel's checker playing program [53] and the pole balancing solution [31]. One of the more famous successes is the TD Gammon program which, in playing itself over

one million times, has learned to play backgammon at an extremely high level and has gone so far as to significantly change the way the game is played by backgammon professionals and masters due to novel ways it has found to win [61].

Shortcomings do exist, however, with the reinforcement learning paradigm. For one, there is currently a variety of issues such as the credit assignment problem [33] and the exploration / exploitation dilemma which make this a difficult method to master for just about any complex learning task. The credit assignment problem is significant in that it deals with the issue of assigning credit or blame accurately to each action taken by an agent in the environment. There are potentially countless combinations of actions an agent can take in the environment and it is often very difficult to reward or penalize an act based on the end result of a sequence of actions. As such, many beneficial actions can be unfairly penalized while counterproductive actions may be rewarded just because of how well the sequence of actions to which they belong scores using the environment's evaluation function. Many methods have been proposed to help solve this issue but it is still a concern and an active topic of research within the field of reinforcement learning.

The exploration vs. exploitation dilemma is also an issue encountered often in reinforcement learning implementations. A reinforcement learning agent, *exploiting* only the best sequences of actions it has encountered, could ensure convergence to some solution but, without further exploration of the space of actions, cannot guarantee optimal or even good solutions. To *explore* the action space of the learner would increase the likelihood of finding good action sequences through searching and evaluating the entire action space. However, without exploiting the good solutions found, the agent runs the risk of never converging and even possibly "forgetting" the good action sequences previously discovered.The most significant hurdle, however, unlike traditional supervised learning techniques, is that a controller is not guaranteed to find an optimal, or even a good, policy using many of the popular forms of reinforcement learning.

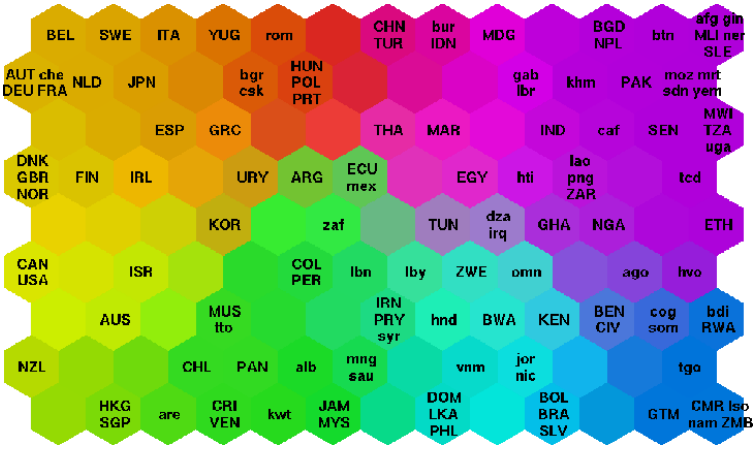


Figure 2.8: SOM which examines worldwide poverty by region. (taken from http://www.cis.hut.fi/-research/som-research/worldmap.html)

2.4 Self Organizing Maps

2.4.1 Description

Self organizing maps (SOMs), inspired by map formation phenomena found to occur in the primate cortex, are very effective tools for clustering unknown data as well as being an effective method for visualizing groupings of high-dimensional input data in two dimensions. The design of the underlying dynamics of these self-organizing maps was motivated by the way neurons are believed to form associations with other neurons in the brain. The *Hebbian rule* suggests that when two neurons fire simultaneously after being presented with some input stimulus, their connection is strengthened ([]). Similarly, in SOMs, connections between computational neuronal elements in the input and output layers are strengthened when they fire simultaneously in much the same manner observed in cortical neurons of the brain. This rule, called the Hebbian rule, forms the basis for very powerful neurally-inspired unsupervised learning methods.

2.4.2 Hebbian Learning

A self organizing map is designed to have a number of output neural elements, or nodes, which take input from all values in input vector X. The output neural computational elements are subject to a neighborhood function which dictates how neighboring nodes are adjusted based on proximity during training to the winning node. Each neural element j has associated with it some weight vector w_{ij} where $1 \leq i \leq n$ (n being the number of inputs) and $1 \leq j \leq m$ (m being the number of nodes in the SOM). Each weight vector that corresponds to a neural element lies in the same vector space that the input vectors are in. The weight vector can be considered a representative vector of the node with which it is associated. "Training" in a SOM essentially consists of conforming all weight vectors to represent in the two-dimensional lattice regions in the space of input data.

There are various ways to select winning nodes in a SOM. One way is to employ a winner-takes-all approach ([26]). Using this rule, the input vector or stimulus is tested against the weight vector of every neural network in the SOM lattice. The node whose weight vector is closest to the input vector causes the corresponding vector to be the winner. Consequently, the output at the winning node is set to be 1.0. All other nodes in the lattice are set to be zero.

Now in training, Hebbian learning dictates that the vector corresponding to the winning node be made marginally closer to the input vector presented to it. In addition, the proximity of nodes in the lattice of output elements from the winning node determines how other nodes should be brought closer to the input vector as well. The proximity information of nodes is generally defined when initially designing the SOM by specifying which nodes neighbor each other. A very common scheme would be to set up a two-dimensional lattice of nodes where each element is attached to up to four neighbors that can influence each other through the unsupervised training process (In Figure 2.8, a SOM lattice of nodes is demonstrated which actually gives every node up to six neighbors as opposed to four). Over an extended period of training, where neighborhoods are made to decrease gradually over time, entire areas of the high-dimensional input data space can be denoted by a group of similarly classified neurons in close proximity to each other.

And, much like in the feedforward multi-layered neural network described previously, a learn-

SOM Training Algorithm
1. Initialize SOM weights randomly.
2. Retrieve a sample input vector, x, from the input training data.
3. Calculate winning node : $i(\mathbf{x}) = argmin_j \|\|\mathbf{x}(n) - \mathbf{w}_j\|\|, j = 1, 2, ..., l$
4. Update weight vectors of all appropriate nodes (including winning node and other nodes in neighborhood $\eta(n)$) : $\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n) h_{j,i(\mathbf{x})}(n)(\mathbf{x}(n) - \mathbf{w}_j(n))$
5. Repeat from step 2 until feature map stabilizes.

Table 2.2: Procedure for training a self-organizing map

ing rate is utilized. The Hebbian learning update rule for updating the weight vector $\mathbf{w}_j$, of a winning node, j, can be described as follows :

$$\Delta \mathbf{w}_{ji} = \eta * (\mathbf{x}_i - \mathbf{w}^{ji}) \tag{2.12}$$

$$\mathbf{w}_{ji} = \mathbf{w}_{ji} + \Delta \mathbf{w}_{ji} \tag{2.13}$$

There are many ways in which a SOM can be trained. The standard procedure for training a Kohonen self-organizing map is shown in Table 2.2. Note that a SOM can take tens of thousands of epochs or more to complete training.

The neighborhood functions can be designed to take the form of all sorts of proximity information and characteristics. They can be defined by such characteristics as shape over an area (e.g. box), by distance function (e.g. euclidean distance, manhattan distance.) One of the more popular neighborhood functions, the gaussian neighborhood, is not a boolean indicator like those described previously, but an indicator, $0 < h \leq 1$, of the current node's proximity to the winning node. What will then result over time is that regions of SOM nodes will ultimately cluster and represent high dimensional input data in the form of a two-dimensional lattice.

Upon completing the training procedure, a mapping should result where regions of neighboring SOM nodes are shown together which can be taken to represent clusters or categories of the input data. What will occur after training is that the ordering of the set of neurons can visually suggest

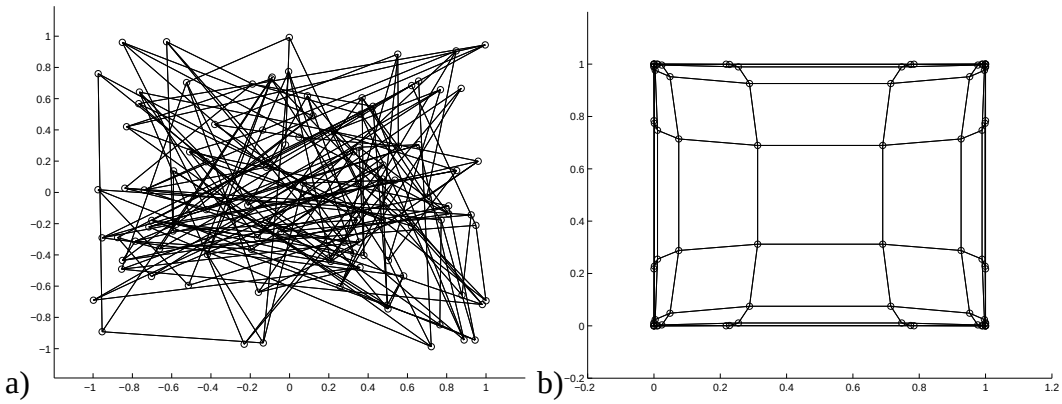


Figure 2.9: These graphs demonstrate before-and-after snapshots that signify the training of a standard SOM designed with a 10x10 lattice of output nodes. Note that output nodes that neighbor each other in the lattice are shown connected by a line. Plot a) is a snapshot of the weight vectors plotted in $\Re^2$, each representing an output node, that comprise the SOM prior to training. Plot b) demonstrates the self-organization that occurs following 20,000 epochs of training using the standard SOM training algorithm of Table 2.2. The training data consisted primarily of vectors from the set $\{(0,0)^T, (0,1)^T, (1,0)^T, (1,1)^T\}$ which would explain why so many output nodes cluster around those points near the corners.

clustering information to the trainer even in the presence of vast vector input spaces. In addition, all weight vectors representing the SOM map nodes converge to some highly-ordered spatial organization in the input space as a result of the neighborhood restrictions imposed on them (Figure 2.9.)

2.4.3 Applications

Self-organizing maps have been used to assist in many areas of technology. These uses range from the creation of cognitive models of cortical map activation ([45], [48]) to the visualization of high dimensional spaces from unordered, un-clustered data ([32]). Using a SOM, the clustering of data inputs thought previously to be unrelated can occur, causing groupings of all types of input data to be confined visually into a rectangular space (or map.) This map would primarily comprise

the activations of the two-dimensional lattice of interconnected neurons in the output layer of the SOM. When training has been successfully completed, some nearest neighbor groupings can be formed from which similarities or categorical information can be inferred or concluded.

Some would call this visual data mining. The advantage of searching or seeking groupings in this manner is that it is very efficient, but also that it is confined to whatever sized 2D lattice the trainer wants to define for it. So, in other words, the groupings can be visualized on a 5-by-5 lattice SOM or a 500-by-500 lattice SOM. The larger one may be able to provide visually more information or insight into the input data and may be able to classify and map much more data than the smaller map. Yet, the smaller map would take some order of magnitude less training than the larger proposed map. Groupings can be viewed once the SOM is fully trained just much like those shown in Figure 2.8.

The application of SOMs in the main work described in subsequent chapters is to use one as a very simple model of associative memory storage. From this model, the processing and subsequent comparison of resulting map sequences generated by incoming auditory phoneme streams to those already stored in the SOM model can be made possible.

The SOM can also be used to take input data and pre-process it as input to other systems. In other words, it can be used to cluster input data which was previously unclassified and take the resulting mappings and redirect them as inputs to other systems. In one such application, which will be described at great length in a future section, one researcher has a robot use a SOM in order to ground into itself a sense of the layout of the room in which it is expected to operate ([60]). The robot can then use its “understanding” of the area it is attempting to travel and make good judgments as to where it is and how to proceed next in order to get to its optimal goal position in the room.

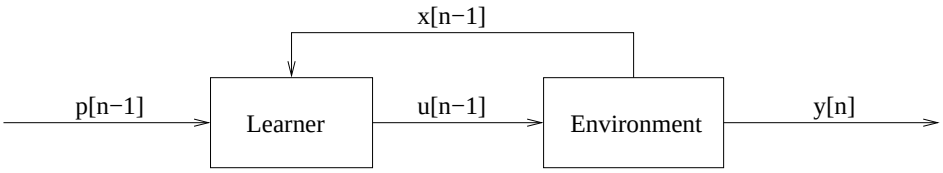


Figure 2.10: Basic setup for the distal learning problem. At time n, the learner accepts as input some intention p[n-1] and current state x[n-1] and must generate an action u[n-1]. The environment then transforms that action in output space to vector y[n] and returns the resulting next state, x[n].

2.5 Distal Supervised Learning

In the classical supervised learning paradigm, target outcomes are presented explicitly by the teacher to the learner for the purpose of training. In the case of distal supervised learning (Figure 2.10), however, the teacher is only capable of providing desired target vectors which are distal in nature to the learner and may only be realized by the learner through its proximal interactions in an external environment. Proximal target values which are generally provided by the teacher in the classical supervised learning framework must now be discovered by the learner in order to minimize the performance error, J, over the entire system of learner plus environment. Here, the learner, which produces proximal action u, can be characterized by the function $u = h(p, x, w)$, while the environment accepts the learner's proximal action, u, and produces the actual distal output, y. Here, x is defined as the current state information used to guide the learner and w represents the learner's weight vector.

One such example of a distal learning problem in which only distal target outputs are available is provided in Jordan [23]. He describes a scenario of a basketball player who intends to shoot a ball through a hoop. The correct series of proximal actions (in this case, arm muscle commands) must be learned in order to propel the ball through the air and environment into the hoop. Only the distal end result of the player's actions ("the sights and sounds of the ball entering the hoop") is accessible from the environment for calculating performance error. An appropriate proximal sequence of motor commands to achieve the desired goal is not available for training from the

teacher. Ideally, providing the desired distal target result of the sensation of the ball going through the hoop along with the input of the current position of the ball in space used together with the intention to shoot the ball into the hoop must suffice for the player to acquire the desired proximal behavior.

In order to train the neural network in this setting using the supervised learning paradigm, Jordan et. al [24] introduces the idea of training an additional neural network to model the environment. Once trained, this additional neural network, also given the term *forward model*, can then be used in conjunction with the system's performance error to train the learner. This forward model can be described by the function $\hat{y}= \hat{f}(x,u,v)$, where v is the weight vector of the forward model and x represents the current state. Once the forward model is sufficiently trained so that its predicted output, $\hat{y}$, is within some acceptable error of the actual output, y (i.e., when $\hat{f}$ is capable of approximating the environment closely) effective training of the distal learner can be achieved (Figure 2.11). To train the forward model, any number, m, of random actions can be generated, $\{u_i|1 \leq i \leq m\}$, from the proximal output space of the learner and run on the environment. The resulting outputs in environment space, $\{\hat{y}_i|1 \leq i \leq m\}$, can be used as target outputs to form input/output pairs $\{(u_i, \hat{y}_i)|1 \leq i \leq m\}$ to train the forward model using standard back propagation methods.

Training cannot occur in distal supervised learning using equation 2.3 as there is no way to calculate $\frac{\partial y}{\partial w}$ directly, where the environment function is unknown. However, in substituting the forward model for the environment function, we can now substitute $\hat{y}$ for y which, after applying the chain rule, yields the following learning rule:

$$\nabla_w J_n = -\frac{\partial u}{\partial w}\frac{\partial \hat{y}}{\partial u}(y^*[n] - y[n]), \tag{2.14}$$

Here, $\frac{\partial u}{\partial w}$ refers to the gradient of the learner's output, u, with respect to its weight vector, w. The term $\frac{\partial \hat{y}}{\partial u}$ refers to the gradient of the forward model's output with respect to its input. Equation 2.4 can then be used in the same manner to update the learner's weight vector, w.

A key component in creating a system such as this is how effectively the forward model is

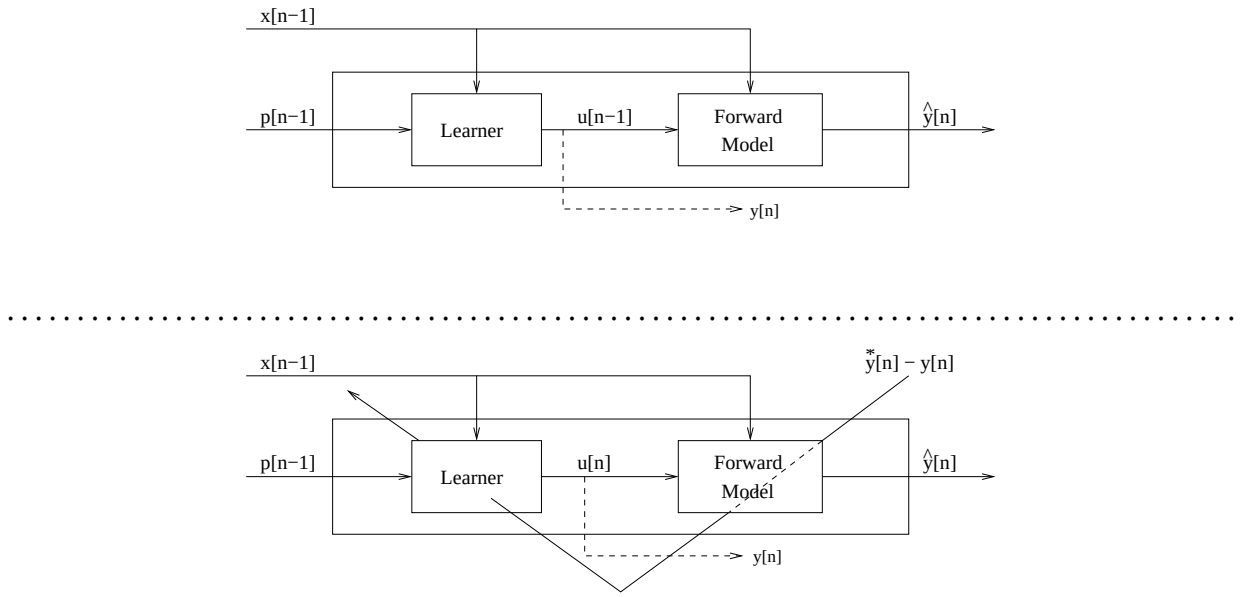


Figure 2.11: (Top) Distal supervised learning framework shown here where, once again, the intended distal learner accepts as input intention p[n-1] and, optionally, state x[n-1] from the environment and responds with action u[n] which is simultaneously sent to the environment and the forward model to generate, respectively, not only the actual output y[n] (shown in 2.10) but predicted output ŷ[n] as well. (Bottom) Training the distal learner requires propagating performance error $y^*[n] - y[n]$ back through the forward model in order to approximate the gradient direction for the sum squared error function essential for effectively updating the weight vector of the distal learner.

trained. A forward model must be sufficiently trained to predict the correct output of the actual environment to effect meaningful weight vector updates to the distal learner. However, an interesting consequence of this framework is that, even if a forward model is not completely trained, the learner can be shown to retain or even continue to learn the desired behavior throughout the distal supervised learning training procedure. This is possible since the term $(y^*[n] - y[n])$ used in training the distal learner approaches zero when the actual environmental result of the learner's proximal action(s) closely approximates the desired distal targets (i.e. correct proximal actions are being generated to produce near-optimal distal outputs). As a result, due to the error gradient

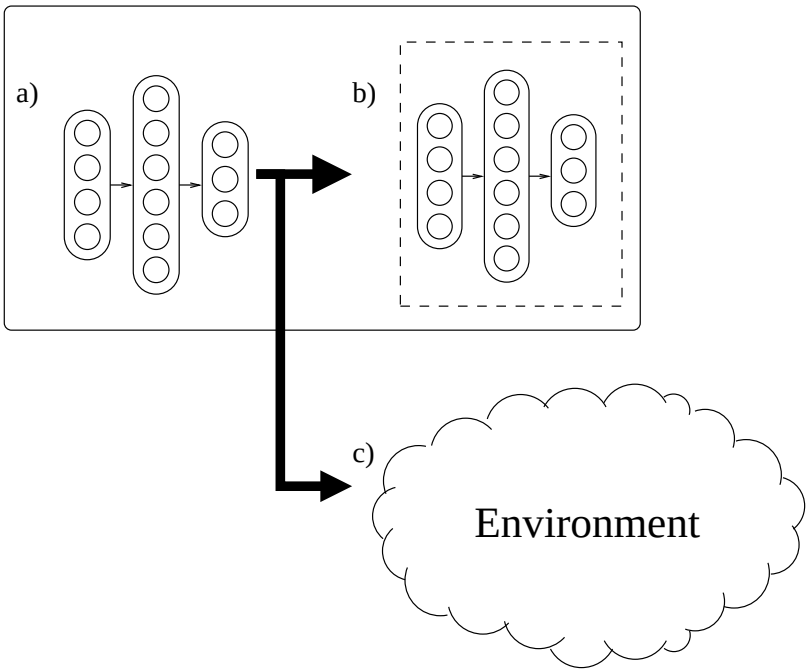


Figure 2.12: Standard setup of a distal supervised learning system utilizing feedforward neural networks for distal learner and forward model structures.

calculations of equation 2.14, the learner's weight vector remains mostly unchanged by equation 2.4 so that the learner will continue to exhibit the same correct proximal behavior. As such, the distal learner and the forward model can actually be trained simultaneously and in series with each other.

Distal supervised learning methods have been used in developing neural networks which can serve as continuous inverse mappings of environments they are placed in [24]. In addition, this method of training neural models can be quite pertinent in computational brain modeling as forward models are being shown more and more to exist in the human brain. These real life forward models, believed to exist in the cerebellum, are thought to serve very similar purposes to those used in computational distal supervised learning studies. That is, they are shown to be useful in learning to anticipate the distal consequence of proximal neural actions for use in various cognitive motor function development tasks such as motor control and speech acquisition ([3], [4], [70], [71], [72]). Developing learning agents to handle these types of problems is hardly an exact science.

Up until now, absolute success has been demonstrated in mostly simple environments and limited success shown in the more difficult environments. A substantial amount of work must still be done in making distal supervised learning a viable model of distal supervised learning problems such as cognitive function acquisition.

Chapter 3

Recurrent Distal Supervised Learning

In this chapter, a modified method of distal supervised learning is presented to address learning in sequential environments. These sequential environments are designed to accept not a single action, as in typical distal learning problems, but a sequence of actions from an agent to then, in turn, yield an equivalent-length sequence of distal consequences. Namely, the modifications entail replacing the typically non-recurrent distal learner and forward model feed-forward neural networks of the existing distal supervised learning framework presented by Jordan [24] with recurrent neural networks. These recurrent networks are capable of utilizing knowledge of past internal states and/or previous actions taken in order to better acquire and produce correct proximal sequential behavior while operating in a sequential environment, even when current state information is not present. Also presented is a version of teacher forcing I modified for use in assisting the learning process of a recurrent distal learner. Lastly, the effectiveness of the proposed system is demonstrated on a sample case of recurrent distal supervised learning using a sequential environment which is designed to be predictable and easy to comprehend for analyzing purposes.

3.1 Motivation

In most studies involving distal supervised learning, the current state is provided by the environment at every time step to the distal learner. This current state vector typically summarizes where the distal learner is as the latter acts progressively in an environment en route to potentially accom-

plishing the end distal goal through its progression . For instance, consider the ball-tossing distal supervised learning scenario provided by Jordan [24] where a person sets out to learn how to propel a ball into a basketball hoop. The single distal target goal sought by this learner in this scenario entails the sensations which accompany the ball entering the hoop. The proximal actions here provided by the learner comprise the series of arm commands required to propel the ball through the air. The current state information required by the learner from the environment throughout this task would be the position of the ball in the learner's visual field that results after each arm motion is performed.

Note that the current state provided at every time step should be distinguished from the distal sensation or result occurring in the environment. The current state is merely information used to assist the learner in acquiring and generating the correct proximal behavior and, technically, can be potentially considered optional and done without (e.g. shooting the ball into the hoop with closed eyes) if the input vector is dynamic and ever-changing throughout the task. Conversely, there will always be a distal consequence in the environment which follows as a result of one or more proximal actions from the learner.

However, if such current state information is not available to be presented to a typical distal feedforward neural network which utilizes a static and unchanging input vector, learning to produce meaningful proximal actions would be hindered tremendously. In other words, given a single static input stimulus, training a standard neural network to produce a series of differing actions in order to produce a desired output sequence in environment space would be nearly impossible. With no current state information with which to tell where it is in deciding on the correct sequence of moves to enact, the neural network will not be properly equipped to provide differing proximal actions over time to eventually realize the desired distal path. The lone exception could result if a single proximal action produced repeatedly could correctly yield the desired series of distal consequences in the environment.

Some method could be developed which would enable a "sight-less" neural network to consider

its own "memory" of actions taken up to this point,

$$\Lambda_{t-1} = \{u_1, u_2, ..., u_{t-1}\},$$

in order to better identify an appropriate subsequent action, u_t, en route to devising some correct series of commands,

$$\Lambda = \{u_1, u_2, ..., u_n\},$$

needed toward achieving the distal goal. For some time, recurrent neural networks have been developed and refined extensively to do just this. However, supervised learning methods for recurrent neural network architectures in distal problem domains required to operate in complex external environments had never been previously addressed.

In addition, there exist problem domains where some account of the previous actions taken must be utilized in the learning of the task. In Ziemke [73], for example, the author demonstrates that recurrent neural networks, in their use of contextual internal information, are better suited than standard feedforward neural networks in many domains requiring sequential outputs. It is therefore natural to wish to extend these capabilities to the distal environment interaction domain, where many very difficult yet pertinent problems exist.

The purpose of the work presented in this chapter is to demonstrate procedures I developed which are capable of training recurrent neural networks to produce a discretized series of correct learned actions from a single intention which will ultimately cause a very specific series of desired consequences to result in the environment. In adding recurrency to the neural networks used in distal learning for this purpose, the idea is that these well-studied sequential generators will be considerably more effective in achieving such behavior (Figure 3.1).

3.2 Forward Model as a Recurrent Neural Network

In a distal setting, the recurrent neural network will require the ability to, given a single input stimulus, produce appropriate sequential behavior which could only be evaluated in the space of

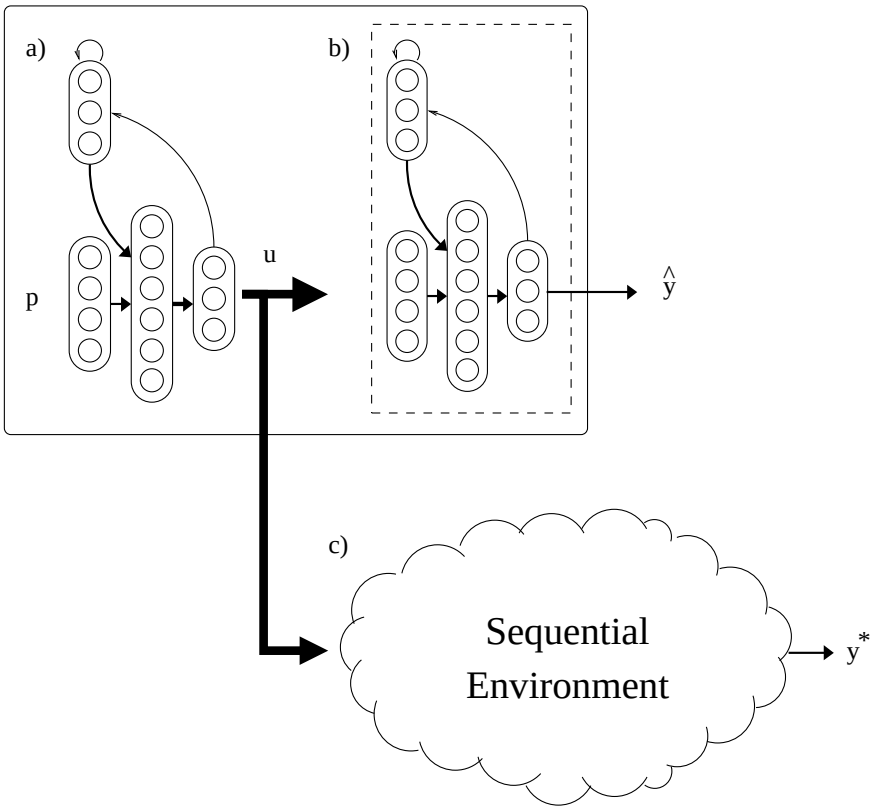


Figure 3.1: A more telling visual depiction of recurrent distal supervised learning. Given a static single intention, **p**, as input, the recurrent distal learner (a.) will look to generate an action sequence, **u**, of n vectors. This action sequence is accepted simultaneously by the environment (c.) and the forward model (b.) attempting to model the environment. What results are output vector sequences ˆy and **y** from the forward model and the environment, respectively. These sets of vector sequences are compared to the set of desired distal vector sequences, $\mathbf{y}^*$ (not shown here), and effect parameter changes of both distal learner and forward model to eventually yield an effectively trained recurrent distal learning neural network.

the external environment in which it operates. Its corresponding forward model, precisely as the environment it looks to emulate, must be able to accept a sequence of proximal actions and map it into a distal sequence as accurately as possible for it to be effective. Standard feedforward network architectures are currently not sufficiently equipped to do this effectively. Just as the sequential environment used must both accept temporal sequences (i.e., proximal action sequences from the learner) and produce temporal sequences (i.e., distal output sequences in the environment), the forward model whose purpose is to emulate the latter must also be designed as a recurrent neural network which both accepts and generates temporal sequences. However, since the particular distal recurrent learner studied here accepts only a single input as opposed to the sequence of vectors accepted by the forward model, two different recurrent neural network designs are addressed.

Using recurrent forward models in distal supervised learning is not a new concept. Tani [60] used recurrent forward models to learn traversal trajectories in training a robot to learn to get to some goal location from an arbitrary point in a room. Jordan [24] suggests the use of recurrent forward models in training a standard feedforward distal neural network guided by current state information to learn to reproduce specific distal trajectories effectively. Neither model, however, addresses generating correct discrete proximal sequential behavior minus current state updates as both continue to rely heavily on receiving streams of correct state information in their design.

In this work, recurrent forward models can take the form of a Jordan network, an Elman recurrent neural network, or even possibly a hybrid of the two (Section 2.2.) The task of the recurrent forward model will be to learn to approximate as closely as possible the sequential mapping of the actual environment. Toward this end, the recurrent forward model should take in sequential actions and, ideally, should return as distal sequences precisely what the environment would. When it is trained sufficiently to do this reasonably well, the recurrent forward model should be able to assist the distal recurrent neural network in learning to produce the correct set of proximal actions needed to yield the series of distal outcomes the trainer is seeking. Current standard neural network gradient descent methods are all that is required to train the recurrent forward model here (Section 2.2.)

Ideally, should the forward model be capable of modeling the environment relation entirely and correctly, the correct proximal behavior of the distal recurrent learner from a single static input can be learned more readily. However, the combination of environment relation and current state function can become exceedingly complex and, hence, extremely difficult to learn. In this case, as long as the forward model can learn to produce the correct distal desired consequences when given the correct, though previously unknown, proximal output sequence, it should be better equipped to train the distal recurrent network.

Training the forward model sufficiently to, in turn, get the learner to generate the correct proximal behavior is still a subject of study. Experimentation can be used to determine things such as recurrent network type (Jordan/ Elman), length of training time, hidden layer size, neuron output functions, best gradient descent training method, etc. Care must be exercised in ensuring the forward model is not overtrained and can generalize as best as possible to the environment relation. To be ultimately successful, as mentioned before, the forward model should be able to map closely the sought-after proximal sequences to the desired distal sequences provided by the trainer in order for it to provide accurate error signals in training the learner. This accuracy desired of the forward model can actually be achieved either in training before or simultaneously while training the distal recurrent learner.

Let U_i^* be some action sequence in the learner's proximal output space which would yield sequence, Y_i, the i-th target distal sequence provided in environment space:

$$Env(U_i^*) = Y_i^*.$$

The goal of the recurrent distal learner is to adjust its weight parameter sufficiently such that it can produce sequence U_i^* to within some acceptable root mean squared error (RMSE) once presented with single vector p_i as input. Note that, if the environment function is not one-to-one, many action sequences can potentially be mapped to the same desired distal trajectory. However, any given forward model can guide the distal learner to only one winning solution. Conceivably, with unlimited time and resources, the forward model could eventually make its way to obtaining the

correct target mapping from U_i^* to Y_i in a variety of ways. Ideally, the forward model can go about doing this by learning to generalize the target mapping through its training from arbitrary proximal / distal trajectory pairs obtained via random sampling or produced from the learner.

For truly complex environments for which generalization may be difficult, actually being capable of mapping the unknown yet sought proximal action sequence, U_i^*, and mimicking the target mapping that way could suffice. To one extreme, one could just ensure that the forward model knows to transform the "correct" proximal behavior to the distal sequential desired outcomes by representing them as input / output pairs somewhere in its training data. This is under the assumption that the correct proximal sequential behavior is available for training a priori, which is often not the case and sometimes defeats the purpose of developing such a system.

In addition, to aid the recurrent forward model in learning the environment mapping, teacher forcing ([67], [69]) can potentially be employed if the Jordan architecture is utilized. In this case, since the desired sequential outputs for the forward model are known already (they are merely the actual sequence of distal outcomes resulting in the environment from the same proximal actions used as inputs), the forward model can be trained in that manner.

3.3 Training the Recurrent Distal Learner

The distal recurrent learner is trained in much the same way as the standard feedforward distal learner. The recurrent learner is trained through interaction with environment and forward model just as it is for the non-recurrent case. The primary differences lie in the structures of the learner and forward model, which both require exponential memory vectors (i.e., context or state layers) for tracing the history or action path taken thus far. The memory vector can reflect an exponential trace, meaning a decay term may be applied to the memory vector at a subsequent time step before adding the latest action to it. In the case of exponential trace vector, a limited amount of previous action taken can be reliably considered in making subsequent action, much like in the non-distal case described in Section 2.5. The only difference would be the existence of the forward

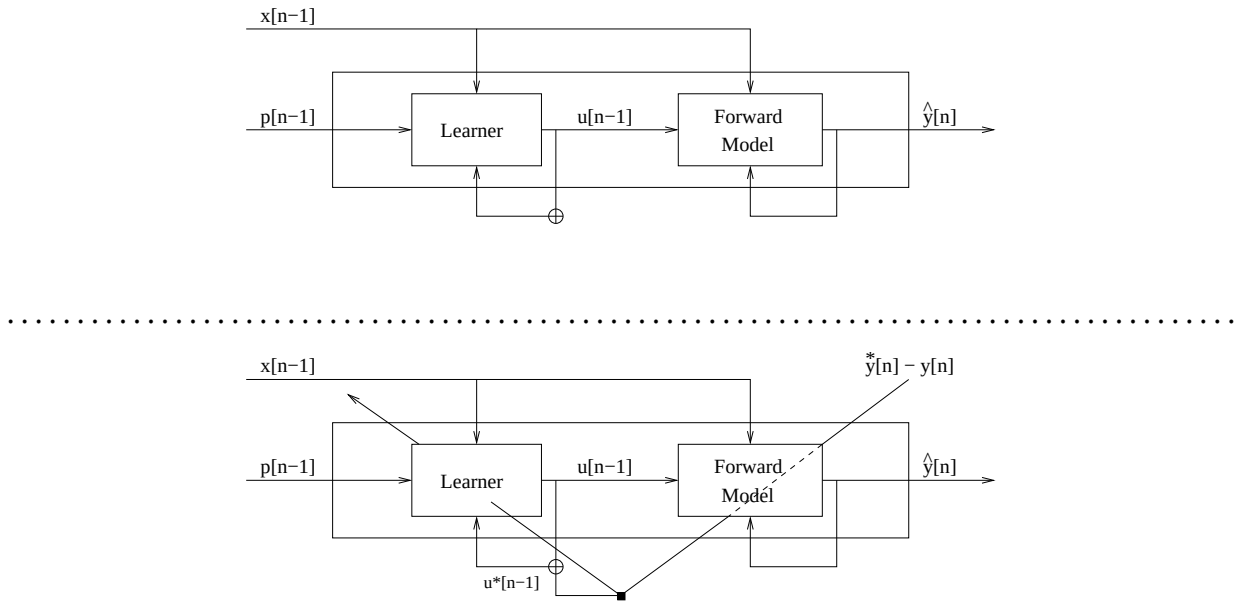


Figure 3.2: (Top) Distal supervised learning framework for training a recurrent neural net to learn proximal sequences which ultimately yield desired sequential outcomes in the environment. Here, the forward model is also a recurrent neural network. (Bottom) Proposed training procedure for the recurrent distal learning paradigm

model necessary for training in the distal setting. The forward model can be used to transform errors from the distal variable space of the environment to the proximal action space of the distal recurrent learner. This can be done efficiently much like the standard, non-recurrent case by propagating these differences between desired and predicted sequential outcomes back through the forward model. However, since the forward model is known to be recurrent as well, the backpropagated error signals need to consider what was output previously in order to propagate back the correct information. Here, the memory module can take in the previous internal state or memory activations and utilize that in order to propagate correctly the right error.

One issue that arises in training forward models stems from the difficulty that standard neural network architectures have in retaining previously learned mappings or trained behavior while adopting new ones. In this case, storing previously seen training instances for continued training in ensuring an appropriate amount of retention of the environment function landscape can be a

good remedy. In training the forward model repeatedly not only on new actions produced by the learner but in retaining recent and promising proximal actions, effective training can be ensured. Here, once again, caching these training instances in developing an efficient forward model may be key to training the distal learner in complex environments and in no way compromises the task of having the latter determine on its own the correct set of proximal actions to take. As the correct answers are not given directly to the distal recurrent learner but to the forward model, the training task is still a very difficult one.

Considering the memory trace vector, **x**, the distal supervised learning procedure can now be modified by redefining the parameterized function of the distal learner to accommodate recurrent links and trace memory from Equations 2.5-2.7 for Jordan networks and Equations 2.8-2.10 for Elman networks. In training the recurrent neural network in this fashion, much of the same methods and formalisms identified in Jordan[24] remain intact. What is needed in order to expand the existing procedure from the non-recurrent case (single input/single distal output) to the recurrent neural network case (single input / multiple distal output) case is to use the recurrent forward model to interpret the distal error into proximal error at each discrete time step of the distal desired sequence. This is a very challenging goal. For the purpose of these initial studies, the distal recurrent learner knows the length of the desired distal trajectory and is, hence, confined to only producing that same number of proximal actions. There are other ways in which the distal recurrent neural net may be trained to execute the correct number of actions (Radio [44]) which will be addressed in subsequent chapters. For now, it should be sufficient to use the length of the desired output sequence as the number of proximal outputs required from the distal recurrent learner to yield the correct behavior. This can be done by assuming that a new action is necessary for a new distal outcome to result in the environment. This assumption can be made valid if no major changes in distal consequence can occur without the learner's direct intervention with action.

Every distal training pair in this particular study is assumed to associate one fixed input stimulus, **p**, with some varying length distal desired sequence, Y^*. In contrast, in standard distal learning studies, such as those proposed in Jordan[24], training pairs only have a single input, **p**, associated

with a single distal output textbfy*. In order for this to resemble the standard distal learning architecture, it will be sufficient to first "unfold" the single input vector to the recurrent neural network into a comparable multi-vector sequence of inputs, each corresponding with one known output of the distal target trajectory. Each of these new input vectors would now include the corresponding contents of the memory vector at that particular time step, whether implementing a Jordan or Elman architecture, as well as the original fixed input vector. The combination of input and memory vector contents from the i-th time step makes for a new input vector which can be uniquely associated to the environmental outcome at the same time step in the desired distal output sequence. In addition, as implied previously, they should number to as many vectors as there are in the target trajectory. As a result, the distal recurrent learner should be able to differentiate between stimuli while keeping in mind the memory trace of previous actions taken up until this point.

When concatenating the context history vector, $\mathbf{x}_t$, to the single input vector, $\mathbf{p}$, at every time step, t, a new sequence of input vectors, $P = \mathbf{p}[1], \mathbf{p}[2], ..., \mathbf{p}[l]$, can be constructed for training the recurrent distal learner. The input sequence, P, will number in length the same as the desired distal output sequence, $Y^* = \mathbf{y}^*[1], \mathbf{y}^*[2], ..., \mathbf{y}^*[l]$. Each newly concatenated input vector, $\mathbf{p}[t]$ in the newly constructed input sequence can be defined as follows:

$$\mathbf{p}[t] = [\mathbf{p}, \mathbf{x}_t], 1 \leq t \leq l. \tag{3.1}$$

where l is the number of vectors in desired distal output sequence, Y^*. As a result, all corresponding input / output pairs $\langle \mathbf{p}[t], \mathbf{y}^*[t] \rangle, 1 \leq t \leq l$, can then be used for training using the standard distal supervised learning procedure (Section 2.5).

3.4 Approximated Teacher Forcing

In implementing a recurrent network, it is known that all previous outputs of the network have a hand in determining the network output at the next step. Hence it follows naturally that if any previous network output is erroneous, learning of any subsequent outputs will be seriously hindered. Until the network output $\mathbf{y}(t)$, $1 < t < l$, of sequence length l is produced correctly, acquiring the

correct mapping to subsequent outputs $\mathbf{y}(t+1), \mathbf{y}(t+2), ..., \mathbf{y}(l)$ becomes increasingly difficult. Implementing a learning scheme in which the teacher can fix the actual output $\mathbf{y}(t)$ to, instead, be the desired output $\mathbf{y}^*(t)$ before learning desired output $\mathbf{y}^*(t+1)$ could potentially be significant in alleviating this problem. Doing this allows for learning in parallel of all vectors of a target output sequence simultaneously rather than having to wait for vector outputs $\mathbf{y}(0), \mathbf{y}(1), \mathbf{y}(2), ..., \mathbf{y}(t-1)$ to be sufficiently correct before training on output $\mathbf{y}(t)$. Such a scheme is often referred to as *teacher forcing* ([39]). Note that here the Jordan recurrent architecture is used, as opposed to the Elman network, as only the external outputs are required and recorded in the exponential trace vector of the Jordan network. Teacher forcing would hardly be possible in an Elman network as there would be no way in advance to know what the actual intermediate layer activations at any arbitrary time step t should be en route to acquiring correct sequence generation capability.

Teacher forcing is a powerful tool which greatly assists in the training of recurrent neural networks. The trouble is that teacher forcing as discussed previously cannot readily be used to benefit the training of a recurrent neural network in a distal setting. Namely, knowledge of the correct proximal output sequences for the recurrent neural network is required in order to provide accurate trace memory vector contents to significantly hasten training. By definition, this information cannot be made available to any distal learning framework for training of a recurrent distal learner.

What can be done, however, is some approximation of the correct proximal sequence can be developed to substitute for the actual, though unknown, correct proximal sequence, U^*. En route to deriving this approximation to U^*, the following set of equations restate the derivation of weight changes for a standard feedforward neural network from the error calculation, J_n, at time step n (Equations 2.1-2.4.)

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

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

Ultimately, the learner's weight vector, $\vec{\mathbf{w}}$, is updated as follows:

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

When training a distal learner, calculation of the weight update above is restated as Equation 2.14,

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

but since the environment function, $\vec{\mathbf{y}} = Env(\vec{\mathbf{u}})$ is unknown, the gradient term $(\partial y/\partial u)$ cannot be calculated directly. However, according to Jordan [24], the gradient term $(\partial \hat{y}/\partial u)$ can be computed for a forward model neural network trained to mimic that environment and taken as an approximation of $(\partial y/\partial u)$ thereby yielding the distal learner update rule,

$$\nabla_{\vec{\mathbf{w}}} J_n \approx -\frac{\partial u}{\partial w}^T \frac{\partial \hat{y}}{\partial u}^T (\vec{\mathbf{y}}^*[n] - \vec{\mathbf{y}}[n]) \tag{3.2}$$

Here, I define a new term, $\Delta\hat{\mathbf{u}}$, used to describe the error correction obtained once the performance error vector, $\Delta\mathbf{y} = \vec{\mathbf{y}}^*[n] - \vec{\mathbf{y}}[n]$, is propagated through the weighted connections of the forward model,

$$\Delta\hat{\mathbf{u}} = \frac{\partial \hat{y}}{\partial u}^T \Delta\mathbf{y} \tag{3.3}$$

Now the distal learner weight update can be expressed as,

$$\hat{\nabla}_{\vec{\mathbf{w}}} J_n = -\frac{\partial u}{\partial w}^T \Delta\hat{\mathbf{u}} \tag{3.4}$$

If we do indeed consider $\Delta\hat{\mathbf{u}}$ as a sufficient estimate of the difference between the recurrent distal learner's output and the correct, yet unknown, proximal action at that time step, a fair approximation of some correct proximal sequence, U^*, can be defined as $\hat{U} = \hat{\mathbf{u}}(0), \hat{\mathbf{u}}(1), \hat{\mathbf{u}}(2), ..., \hat{\mathbf{u}}(t-1)$, where :

$$\hat{\mathbf{u}}(i+1) = (\mathbf{u}(i+1) + \Delta\hat{\mathbf{u}}(i+1)). \tag{3.5}$$

Here, $\Delta\hat{\mathbf{u}}(i+1)$ is the vector of predicted proximal error obtained by propagating distal performance error $(\mathbf{y}^*(i+1) - \mathbf{y}(i+1))$ back through the trained forward model. This vector, known as the error vector used in effecting weight updates in the recurrent distal learner, can be thought of

as an approximation of the difference between the erroneous proximal output, $\mathbf{u}(i+1)$, given by the learner and the correct but unknown output, $\mathbf{u}^*(i+1)$. Assuming the forward model is trained effectively, their sum should come close to the correct proximal action required at time i+1.

Therefore, though desired proximal output sequence U^* is not directly known in order to conduct true teacher forcing in the context layer of the recurrent distal learner, its effect on the trace memory vector can be approximated as follows:

$$\mathbf{x}(t+1) = (\hat{\mathbf{u}}(t+1)) + \alpha\mathbf{x}(t) \tag{3.6}$$

$$= (\mathbf{u}(t+1) + \Delta\hat{\mathbf{u}}(t+1)) + \alpha\mathbf{x}(t) \tag{3.7}$$

where $\mathbf{x}(0) = \vec{\mathbf{0}}$. In other words, the idea is that approximated teacher forcing (Equation 3.4) can be used in the place of standard teacher forcing (Equation 2.2.1) even when given the situation where desired proximal output sequences are not available for training. This hypothesis will be tested and shown to be effective in the various recurrent distal supervised learning applications covered in this work. The entire algorithm for training a recurrent neural network is listed in Table 3.4.

3.5 Use of Time Delay Memory Structures in Recurrent Distal Supervised Learning

In looking to utilize past output history in computing subsequent actions, one can potentially utilize delay-line memory structures instead of, or in conjunction with, the exponential trace memory input vectors described previously. Like exponential trace memory vectors, the use of such delay-line memory structures would be a straightforward extension of what was described already in Section 2.2.2. In merely copying the contents from the appropriate hidden or output layer to the first delay-line memory vector and propagating those activations one-by-one with subsequent discrete time-steps, one can potentially arrive at the same benefits as those one would expect in a

Training Procedure for a Recurrent Distal Learner
RDL(g, h, Env, **p**, Y*)
1. Pre-train forward model
2. Single-input / single-output re-assignment -
Given : • training pair - $< \mathbf{p}, Y^* >$
• Input - **p**
• Distal Output Sequence, Y* = **y***[1] **y***[2] ... **y***[k]
• Initial memory vector - **m**(0) = $\vec{0}$
3. For each distal target **y***[i], $1 \leq i \leq k$
4. Update input $\mathbf{p}_i$ with memory **m**(i-1): $\mathbf{p}_i = concat(\mathbf{p}, \mathbf{m}(i-1))$
5. **Compute:**
• recurrent learner output sequence, **u**(i) = h($\mathbf{p}_i$,**w**), given input $\mathbf{p}_i$ and recurrent learner's weight vector **w**
• distal output, **y**(i) = Environment(**u**(i))
• estimated distal output, $\hat{\mathbf{y}}(i)$
6. Compute distal error: $\Delta\mathbf{y} = \mathbf{y}^*[i] - \mathbf{y}[i]$
7. Estimate learner (proximal) error: $\Delta\hat{u} = -\frac{\partial\hat{y}}{\partial u}\Delta\mathbf{y}$
8. Calculate and apply update to weight vector **w**:
• $\nabla_w J_n = -\frac{\partial u}{\partial w}\frac{\partial\hat{y}}{\partial u}\Delta\mathbf{y} = -\frac{\partial u}{\partial w}\Delta\hat{u}$
• $\mathbf{w} = \mathbf{w} + \alpha\nabla_w J_n$
9. Update memory layer **m**, $0 \leq \beta < 1$: $\mathbf{m}(i) = \mathbf{u}(i) + \beta\mathbf{m}(i-1)$ or $\mathbf{m}(i) = (\mathbf{u}(i) + \Delta\hat{\mathbf{u}}(i)) + \beta\mathbf{m}(i-1)$ (approximated teacher forcing)
13. Re-calibrate recurrent forward model : (train on $< \mathbf{u}(i), \mathbf{y}(i) >$)
14. Endfor (step 3.)

Table 3.1: Training procedure for a recurrent distal learner.

simpler non-distal sequential problem domain.

However, one issue that arises in this context is the use of teacher forcing ([67], [69]). Teacher forcing can be readily used in tapped delay-line memory applications in non-distal recurrent networks since the immediate desired behavior is known to the trainer and can be subsequently furnished to the first delay line module to effect training speedup in learning the desired sequential task. However, in the distal recurrent supervised learning domain, once again, the desired proximal behavior is probably unknown to the trainer. In this case, approximated teacher forcing can be utilized in the training of the recurrent distal learner to what should amount to improved performance over much of the run. Here, given the estimated proximal error provided by the forward model, the desired proximal action can be approximated and placed on the delay-line memory queue in the same manner as in the non-distal case. Figure 3.3 demonstrates an example recurrent distal supervised learning architecture in which the recurrent distal learner is outfitted with some number of "tapped" delay-line memory vectors in the same manner as was described in Section 2.2.2. In this particular example, the recurrent forward model is not given delay-line memory vectors to work with. It is, however, not the case that recurrent forward models could not be given this capability as well.

3.6 A Distal Sequence Generation Task Using a Simple Environment

For the initial work addressing supervised recurrent network learning from distal target sequences, a simple system is demonstrated. Here, a sequential neural network is trained in a simple environment whose characteristics and properties are well understood. This distal recurrent neural network learns to generate varying length discrete action sequences when given single static input vectors. These action sequences ultimately yield the desired distal target sequences provided by the distal teacher when executed in the environment.

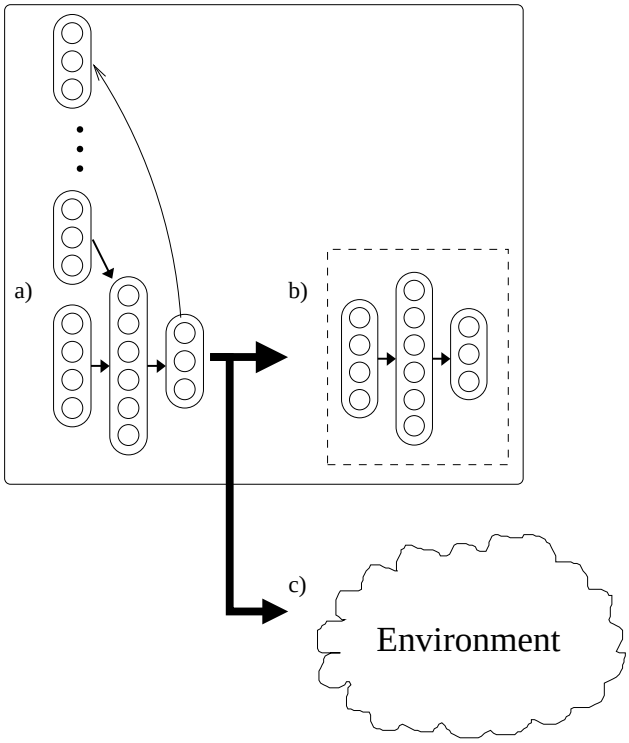


Figure 3.3: An example setup of delay memory layers in use by the recurrent distal learner. Note: delay memory modules can be added to either or both recurrent distal recurrent learner and forward model structures as required. In the case shown here, only the recurrent distal learner is given delay-line memory layers.

3.6.1 Simple Sequential Environment for Preliminary Study: Concatenation

I sought to identify initially a less complex environment which could serve as a first test to verify that the proposed approach to recurrent distal supervised learning would perform as hypothesized. Such an environment would preferably possess these properties:

1. There is an intuitive series of outputs given a sequence of input vectors.

2. There is a one-to-one relationship between the input sequence and the output sequence space. In other words, given a valid sequence of outputs from the environment, only one possible input sequence could generate it.

The environment mapping, f^*, used here (illustrated in Figure 3.5) is merely one which accepts a sequence of input vectors $\{\vec{x}_1, \vec{x}_2, ... , \vec{x}_k\}$ and produces a corresponding list of output vectors $\{\vec{y}_1, \vec{y}_2, ... , \vec{y}_k\}$ where each $\vec{y}_i$ is a vector consisting of a concatenation of the inputs seen thus far plus a series of trailing 0's to fill the remainder of its contents, if any. This can be described as follows:

$$f^*(\{\vec{x}_1, \vec{x}_2, .., \vec{x}_k\}) = \{\vec{y}_1, \vec{y}_2, ..., \vec{y}_k\}, \quad where \;\; \vec{y}_i = \begin{pmatrix} \vec{x}_1 \\ \vec{x}_2 \\ . \\ . \\ \vec{x}_i \\ 0 \\ . \\ . \\ 0 \end{pmatrix}, 1 < i < k < c. \tag{3.8}$$

Here, k denotes the number of vectors in the input sequence, m denotes the length of any input vector, and c denotes the maximum length allowable for an input sequence to the concatenation environment. Each input vector $\vec{x}_i$ is a column vector such that $\vec{x}_i \in \Re^m$ while the resulting output vector $\vec{y}_i$ will be a column vector such that $\vec{y}_i \in \Re^{(m \times c)}$.

The resulting output vector will always have length equivalent to the product of the length of the input vectors and the maximum sequence length possible. Any entries in the vector which are not filled in through the concatenation operation are merely set to zero. The length of the resulting output sequence from this environment will equal the number of vectors in the input sequence presented to it. This constructed mapping is demonstrated in the example of Figure 3.5 for a maximum possible sequence length of 4.

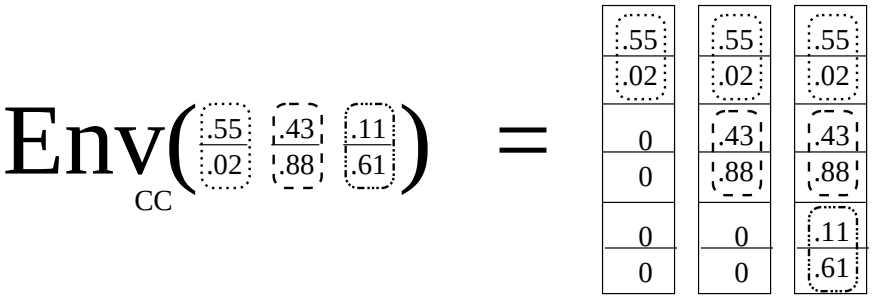


Figure 3.4: A simple illustration of the sequential concatenation environment. Above, the environment function is shown taking each vector in the input sequence in order at each time step and concatenating it to all previously seen input vectors to form a new vector in the output sequence. Varying line-styles (dotted, dashed, and dot-dashed) are employed to

One key property of this environment is that there is only one input sequence which can yield any legal output sequence. This property greatly simplifies the learning task of the recurrent neural network situated in the environment aided by the forward model. This is because the forward model will be able to propagate back to the learner only information which it can use to learn the precise sequence it needs to produce. If it were possible to have many potential input sequences yield the same desired distal sequential outcome in the environment, the forward model could assist the learner in learning to reproduce just one such proximal sequence. However, it would be very possible for the produced sequence to be something other than the desired proximal set of actions should a very specific proximal output be expected. This is only an issue in this setting because, in this particular exercise, proximal accuracy is key in measuring success for this method. The main properties of the environment ensure us that the specific proximal outputs needed to produce the desired distal sequences are readily derivable for use in measuring performance. In many other domains which utilize a distal supervised learning framework, one-to-oneness from an environment's input to its output space is much less of an issue.

Shown in Figure 3.5 are sample input/output sequence interactions of the concatenation environment mapping, f^*, (shown as black arrows) used to demonstrate the effectiveness of the recurrent distal learning architecture. On the bottom are three example discrete input vector sequences

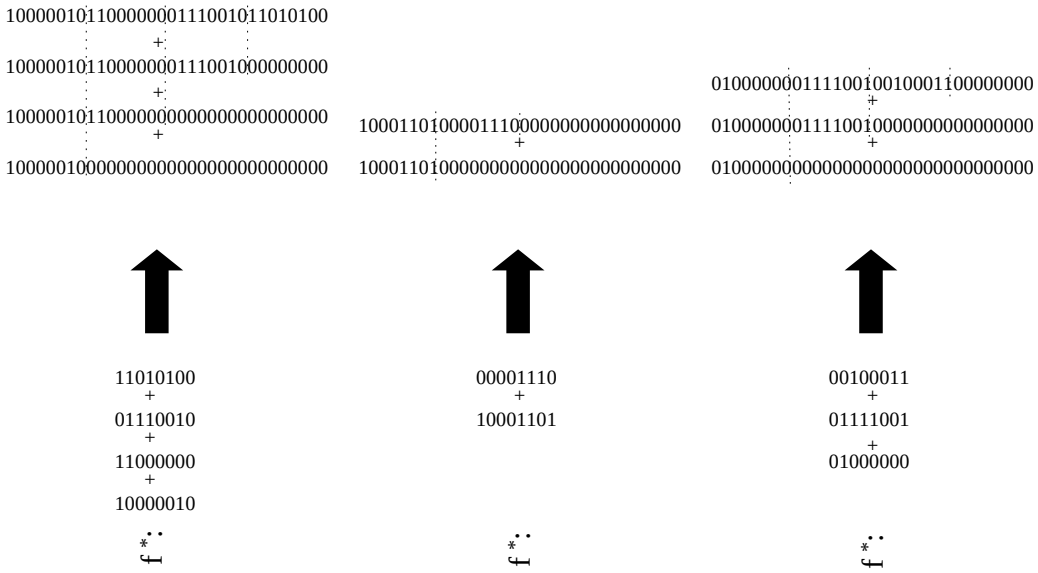


Figure 3.5: Three of the ten input / output sequence pairs used in training the recurrent forward model for the distal concatenation experiments of Section 3.6. Just like the example mapping of Figure 3.4, the concatenation environment (shown as the black upward arrows) accepts each of the three sequences of vector inputs, each of which being a binary vector of length eight, and transforms them into corresponding concatenated vectors of the same sequence length but containing vectors of length 32. Dotted lines are used to delineate the concatenated inputs within the resulting output vectors

each having vector lengths of eight but varying in sequence lengths of four, two, and three, respectively. The arrows denote the mapping (described in Equation 3.6.1) of these input sequences by the concatenation environment to distal output vector sequences having the same sequence length but all containing vectors of length thirty-two. The three proximal / distal sequence pairs shown above are examples picked from the actual ten used in the preliminary experiment outlined in Section 3.6. To successfully accomplish this distal sequential learning task, ideally the recurrent distal learner will learn to produce the correct proximal output sequences (left) when presented with the single static vector (not shown) associated to the target distal output sequence (right). Performance results of the model are shown in Figure 3.6.

3.6.2 Experiment

The distal recurrent supervised framework shown in Figures 3.1 and 3.2 is used in this initial experiment where the distal learner and forward models, both recurrent Jordan networks, are set in series with each other and assigned random initial weights. The external environment is the concatenation mapping as described in Section 3.6.1. Ten varying length vector sequences are generated randomly in the output space of the learner and recorded as the desired proximal output sequences for testing the accuracy of the learner throughout the training process. These ten action sequences are then mapped by the environment to ten distal output sequences, each having the same sequence lengths as their proximal counterparts, which are stored and used as the desired distal outputs for the study. Ten static input vectors of the form $[\underbrace{0, ..., 0}_{(j-1)}, 1, \underbrace{0, ..., 0}_{(n-j)}]$, where j is the jth input vector and n is vector length 10, are associated to the ten distal output sequences as input / output pairs. The task is to see if the distal recurrent neural network can learn to produce the original ten generated action sequences which would yield through the environment the desired distal output sequences given the ten static input vectors using the proposed framework.

To begin the simulation, the forward model is first trained for 1000 epochs on 1000 varying length input / output sequence pairs, 990 generated randomly plus the ten generated sequence pairs discussed previously. The idea is that the better the recurrent forward model is trained to model the concatenation environment, the more efficiently the recurrent distal learner can be trained. Then the distal learner, presented with a static input vector, produces a vector sequence which is submitted to the environment to yield the *actual output sequence*, y. The same vector sequence is also submitted to the forward model to yield the *predicted output sequence*, $\hat{y}$. Both outputs can then be used with the desired output sequence, y^*, to yield predicted error (y^*-$\hat{y}$) and performance error (y^*-y). The predicted and performance errors can then be used to effect weight vector updates of the forward model and distal learner recurrent neural nets, respectively. The predicted error, which merely measures the accuracy of the forward model over the input / output sequence pairs, can be used to modify the forward model weight vector using standard gradient descent methods. This can then

be repeated for all ten static inputs to complete the epoch.

The results shown in Figure 3.6 describe key characteristics of the best training run for recurrent distal learners in this learning task. This top-performing recurrent distal neural network itself used a hidden layer of 30 units while the forward model it utilizes works with 25 units in its own hidden layer (indicated as $< 30, 25 >$ above both graphs.) Three error curves are shown together to demonstrate the various interactions occurring throughout the training of this recurrent distal learner (namely the forward model error, the distal learner error, and the distal performance error..)

First, similar to the practice used in standard distal supervised learning, the recurrent forward model is trained for 1000 epochs before training of the recurrent distal learner is initiated. This stage is often referred to as the *b*abbling stage and enables the forward model to acquire behavioral characteristics of the environment so that it can more aptly propagate effective error signals back to the recurrent distal learner. Also note that, again in a similar manner to standard distal learning, training of the forward model continues throughout training of the recurrent distal learner. The interaction between the recurrent distal learner and the environment provides a steady supply of training examples which the forward model can use to train on en route to better mimicking of the environment mapping.

The varying length sequential outputs from the recurrent distal learner, produced when given the set of static input vectors, are compared to the set of desired proximal output sequences throughout training to yield a proximal error training curve which closes with a RMSE of just over 0.05 (Figure 3.6 a.) The desired proximal outputs can be found in this domain since, by design, the dynamics of the sequential environment are so well understood that its inverse is easily determined. In most complex domains, however, the proximal desired targets for the learner cannot be known a priori and, hence, this measurement usually cannot be determined for analysis.

The distal performance error curve, computed throughout training as the RMSE between actual distal outcomes resulting from the learner's interaction in the environment and the desired distal sequential outcomes provided by the teacher for training purposes, is shown to converge to an RMSE of just under 0.05.

As stated previously in Section 3.4, the error propagated through a sufficiently trained forward model from a desired target sequence can be taken as an estimate of the difference between the learner's desired proximal output and its current output. Hence the sum of the learner's current "incorrect" output and the propagated error should yield some approximation for the correct desired proximal outputs. The estimated action sequence error is the RMSE between this sum and the actual desired proximal outputs. Figure 3.6 b. is merely a demonstration of the utility of the propagated error which is itself used to modify the existing distal supervised learning rule for this work. Plotting together the training curves e graph shows that the current output plus the propagated error is even closer to the known desired proximal outputs than just the current output alone.

Figure 3.7 offers further proof in support of the thesis that using the propagated error for improved memory layer updates can improve training of the recurrent distal learner in sequential environments. This figure superimposes the training curves of two recurrent distal learners attempting to handle the same learning task described previously while operating in the concatenation environment. The initial weights and training data were kept the same between the two runs shown to ensure that approximated teacher forcing alone, or the lack thereof, could be the contributing factor to improved training of either recurrent distal learner. Here, Figure 3.7 shows the learner using approximated teacher forcing indeed produced the better distal performance errors, converging at an RMSE of .0571 while the learner that did not use approximated teacher forcing was shown to converge to .0689.

3.6.3 Conclusions

In summary, the figures of Section 3.6.2 verify the usefulness of the work described here by demonstrating the successful training of a sample recurrent distal neural network capable of replicating the desired distal outcome sequences in a sequential environment, namely the concatenation environment, from single static input vectors. In Figure 3.6a., the diminishing RMSEs of the recurrent forward model, recurrent distal learner, and of the results of the latter's proximal sequential actions in the environment in an example recurrent distal learning system are charted throughout train-

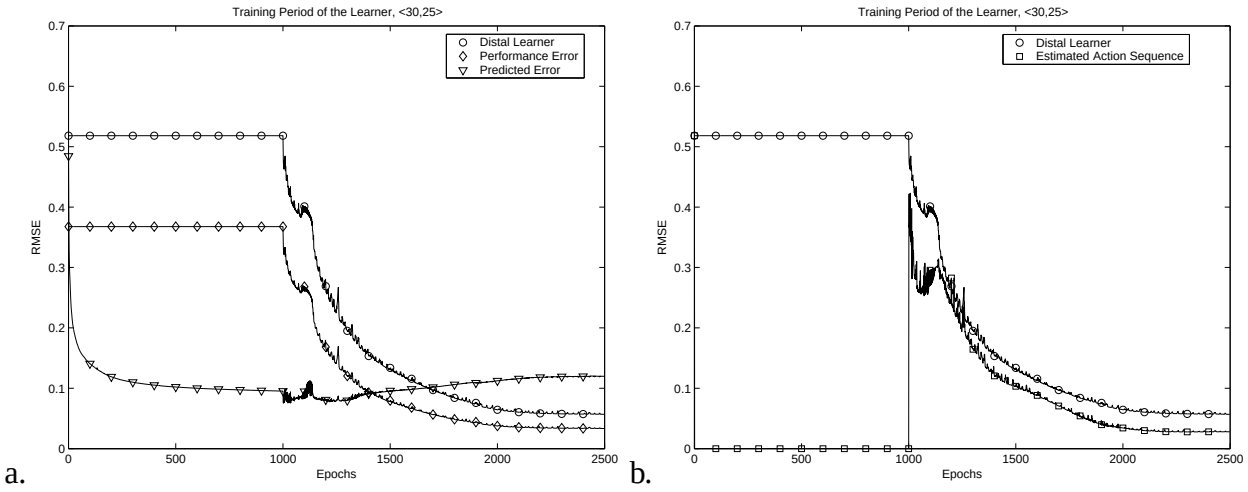


Figure 3.6: Training performance charts of the recurrent network using distal target sequences.

ing. Figure 3.6b. charts the RMSE of the proximal sequential outputs of the same recurrent distal learner against the RMSE of the same proximal sequential outputs plus the approximated error attained through use of the forward model. Essentially, this chart demonstrates that even as the proximal actions given by the recurrent distal learner improve in accuracy as training progresses, the same proximal actions added with the error correction provided by the recurrent model are shown to be even more correct throughout training. This demonstrates that the sum tracked by this curve would be a more viable output to incorporate into the context, or memory, vector to enable more efficient training. Lastly, Figure 3.7 verifies that using the sum of the learner's less-than-accurate proximal output at any point in its action sequence with that estimated error correction attained from the recurrent forward model at that time step to update the learner's memory layer does indeed tend to lead to better distal learner training than when the proximal output alone is used.

Despite this initial success, this experiment helped to bring some concerns to light:

1. **Forward Model Training** - Preliminary experiments seemed to suggest sufficient training of the forward model is absolutely essential to the training of the recurrent distal learner. This may become difficult in more complex domains and needs to be studied further.

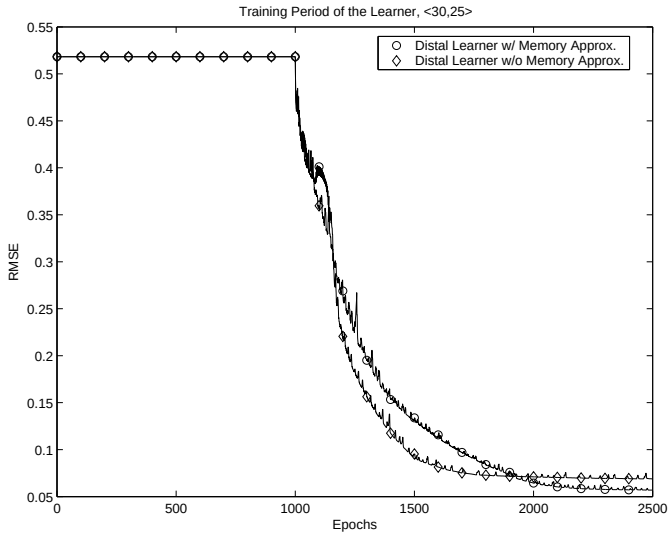


Figure 3.7: Approximated teacher forcing, or using error signals propagated through the forward model to better approximate previous proximal output states for more effective exponential trace memory updates, is shown above to assist the distal recurrent learner to converge better than when it is trained without it.

2. **Scalability** - The relatively high computational effort required to accomplish learning in this not-so-complex sequential environment could imply tremendous difficulty if this modified architecture is used to train recurrent networks in truly large and complex environments. This new system of recurrent distal supervised learning must be validated in much tougher sequential environments to judge how effective it can truly be. A tougher environment is indeed introduced and used for evaluation purposes in Chapter 5.

3. **Ambiguity** - In many complex distal domains, the method found by the learner to yield the end distal target output sequences is more or less irrelevant as long as it is reached. In an environment where multiple sequential paths (sequences) can be used to arrive at the same distal target output, the forward model will essentially ”select” one viable sequence to guide the learner to acquire. In certain learning tasks, however, a very specific action sequence is preferred for the learner to acquire. In a domain such as this, methods need to be developed

through which the forward model can be used to guide training of the recurrent distal learner towards that desired proximal learned behavior.

4. **Varying length sequences** - This preliminary distal supervised sequential learning system assumed a priori knowledge of the length of the desired proximal sequences which the distal learner must be trained to produce. This is neither desirable nor practical in many truly complex sequential environments. One idea to achieve the desired behavior is to train the forward model to produce an 'End of Sequence' (EOS) vector once a correct sequence has ended. It would then be possible to train the distal learner to output the EOS vector after outputting the correct number of outputs in a sequence. Something similar to this was demonstrated in Radio et. al. [44] but not in a distal learning framework such as this.

Ultimately, these results demonstrate for the first time that, given a single, unchanging input stimulus and a corresponding sequence of desired distal outcomes, acquisition of correct proximal sequential behavior can indeed be attained in a sequential environment that provides no consistent stream of current state updates. Existing systems which utilize Jordan's distal supervised learning procedure to train feed-forward neural networks require constant updates from the environment, especially when provided only with static input vector, to acquire the correct learned proximal behavior and should essentially falter when such current state updates are absent. Replacing standard feed-forward neural networks in Jordan's architecture with recurrent multi-layered neural networks turned out to be a very effective method of addressing supervised learning in sequential environments. In addition, proximal error correction provided by the recurrent forward model can, in turn, further improve training by making less-inaccurate the proximal actions taken by the recurrent distal learner before adding them to its memory layer. This, in effect, helps to encourage noticeably better convergence in the training process for the recurrent distal learner. It is highly improbable that any such mechanism can be developed for standard non-recurrent distal supervised learning systems in much the same way that teacher forcing strategies are useless with regard to non-recurrent feedforward neural networks in non-distal learning tasks.

3.7 Contributions of the Chapter

The work described in this chapter extends the existing distal supervised learning framework to handle sequential learning tasks. Here, both the distal learner and the forward model which are ordinarily created as single input/ output neural networks are replaced with recurrent neural networks. Such recurrent neural networks are capable of utilizing their histories of past actions to make subsequent decisions with or without being informed of their current state in the world. In doing so, the recurrent learner can thereby acquire the ability to reproduce a set of time-varying distal target outputs in the environment from a static input vector without the need for constantly updating current state information.

To evaluate this proposed extension to the distal learning framework, I implemented a learning system that employed a sequential environment designed in a manner where its behavior was predictable and easily verifiable. The sequential environment used in this particular implementation was the concatenation environment which, at every time step, took all vectors in a sequence accepted before the current time step and concatenated them into one long vector. The goal of the system was to train the recurrent distal learner to learn to output the sequence of vectors responsible for generating the desired sequence of long concatenated vectors in the environment while presented only with a single static input vector. The system was shown to successfully train recurrent networks to accomplish the task.

The other significant contribution demonstrated here is the introduction of an approximated teacher forcing strategy to assist in the training of the recurrent distal learner. In a manner which is inspired from standard teacher forcing practices utilized in the training of standard recurrent neural networks, more accurate memory vector updates are shown to result using feedback from the recurrent forward model. This newly devised strategy is shown to enact quicker, and at times more accurate, convergence to the desired sequence of outcomes.

Chapter 4

Sequential Processing using Self-Organizing Map Models

The purpose of this chapter is to introduce a new modification on an effective method for processing input sequences in self-organizing maps (SOMs.) Currently, one of the more effective methods of utilizing a SOM to uniquely encode an input sequence is called the SARDNET method (James [21]). This method presents a very computationally effective and meaningful way of encoding an input sequence of input stimuli into a SOM. Unfortunately, at times the SARDNET procedure does not go far enough to ensure the uniqueness of any arbitrary input sequence in its SOM output lattice. In this chapter, I outline the method known as the SARDNET algorithm and then describe a modification I introduce that is capable of creating even more unique output representations for input sequences based on the proximity of each input vector to known candidate vectors. This chapter is essential in establishing a method to properly, efficiently, concisely, and uniquely represent input vector sequences so that it can be utilized as an essential piece of the very complex distal sequential learning task described in the next chapter (Chapter 6). There, the modified SOM can be treated as a viable model of associative memory in humans for use as part of a very ambitious distal learning task in a complex sequential environment, termed the phoneme sequence generation environment, in an attempt to mimic the process by which humans acquire the ability to produce words.

4.1 Background

In certain problem domains, it is conceivable that sequences of input stimuli may be required for mapping in a self organizing map (SOM) as opposed to having static stimulus patterns. In addition, much like in the static input case, it would be imperative that each sequence of inputs be mapped such that the resulting output pattern will be as distinct and different as possible from any other potential sequence of inputs. Typical implementations of Kohonen SOMs, however, lack the functionality for handling and classifying sequential input data.

In the existing literature, there are two classes of SOM models which are designed to handle sequential inputs. One approach, termed the One-Shot, Multi-winner SOM [54], takes a more biologically inspired approach to accomplishing the desired computational behavior. The other, called SARDNET [21], accomplishes the goal using a more computationally efficient method. In this chapter, I develop a modification of the SARDNET architecture, namely in its output dynamics, such that, rather than output a 1.0 at winning nodes as most SOM models do, map nodes output a value which serves as an indicator of 1) how close the input vector in the sequence truly is with respect to any of the anticipated, or "candidate", input vectors to the SARDNET SOM as well as 2) how close the current map node is to the actual winning node.

4.2 SARDNET

The SARDNET architecture [21] allows for a very efficient classification of input sequences, each identified almost uniquely by its series of map node activations. In this architecture, many rules developed for the Kohonen Map remain intact in the SARDNET SOM. However, in creating an output map, once a winning map node is selected for an input vector in a given vector sequence, that map node is marked never to be used in that sequence again. The map node would then be given an output of 1.0. Once done, all previous activations would then be decremented by multiplying each one by some decay constant, $0 < d < 1$. This is then repeated for the length of the input sequence. The tendency of each output map produced en route to forming the final SARDNET

The SARDNET Training Procedure
Initialization: Clear all map nodes to zero. MAIN LOOP: While not end of sequence 1) Identify unit whose weight vector that best matches the input. 2) Adjust weight vectors of other nodes based on user-defined neighborhood function (e.g. gaussian) using standard Hebbian learning. 3) Exclude the winning unit from subsequent competition. 4) Decrement activation values for all other active nodes.
RESULT: Sequence representation = activated nodes ordered by activation values

Table 4.1: The SARDNET Training Procedure

output pattern using this procedure is that only one unique input sequence that could be responsible for producing each map. Training of the SARDNET SOM similarly involves marking winning nodes as it traverses through the input sequence. The actual training algorithm is listed in Table 4.1. Subsequently, the procedure used for producing an output pattern in a trained SARDNET SOM from an input sequence is listed in Table 4.2.

Figure 4.2 demonstrates two plots of the weight vectors of a 10x10 SARDNET SOM in which the input vectors, as well as the weight vectors, one for each node in the output lattice, are two-dimensional vectors. Each input sequence ranges from two to four vectors in length and are comprised solely of some combination of the following four candidate vectors, $\{[00]^T, [01]^T, [10]^T, [11]^T\}$. Connecting lines are shown to designate adjacency between output nodes in the output lattice, each of which corresponds to some 2D weight vector. In the weight plot of Figure 4.2a., the weight vectors of the SARDNET SOM are randomly initialized and demonstrate no organization prior to training. The weight plot of Figure 4.2b., however, is a snapshot of the weight vectors after training for thousands of epochs. Here, organization of the weight vectors given the neighborhood function is immediately apparent. Also note that most node vectors look to accumulate around the

Output Dynamics of a Trained SARDNET SOM
1) initialize all node outputs to 0
2) select x_i in sequence X,
3) multiply output of all marked nodes by $0 < \mu < 1$
4) determine closest *unmarked* (winning) node and set its output to 1.0
5) mark winning node
6) repeat from 2) until sequence X is completed.

Table 4.2: Outline of the procedure for producing output maps in the SARDNET SOM once presented with input vector sequence, X = $\{x_i | 1 < i < n\}$.

four candidate vectors from which the list of input sequences was solely created. Also note the relatively even distribution of weight vectors surrounding the four candidate vectors implying an even distribution of the candidate vectors throughout the input data. An output node corresponding to any weight vector in close proximity to one of the four candidate vectors will be among the first to be selected and turned on once that candidate node is seen by the SARDNET SOM as input.

In addition to this procedure being very fast, it turns out that it is extremely memory and computationally efficient as well. James et al. [21] point out that the SARDNET SOM can classify p^{nl} sequences utilizing only lp^n nodes in it's output lattice, where *p* is the number of possible values of an input, *n* is the length of an input vector, and the maximum length of a vector sequence is represented by the variable *l*. Many other previously suggested sequential SOM architectures would tend to map each sequence to a separate map node, potentially requiring p^{nl} map nodes.

The SARDNET architecture provides a great tool for producing potentially unambiguous activity patterns for finite lists of input vector sequences. However, ambiguity among activity patterns in the output maps can still occur. Truly unambiguous activation patterns result primarily when any input vector seen anywhere in one of the set of training input vector sequences can be mapped uniquely to one specific winning output node in the SOM. In other words, this outcome can be ensured only if no two input vectors can be mapped to the same winning node. If potential vector

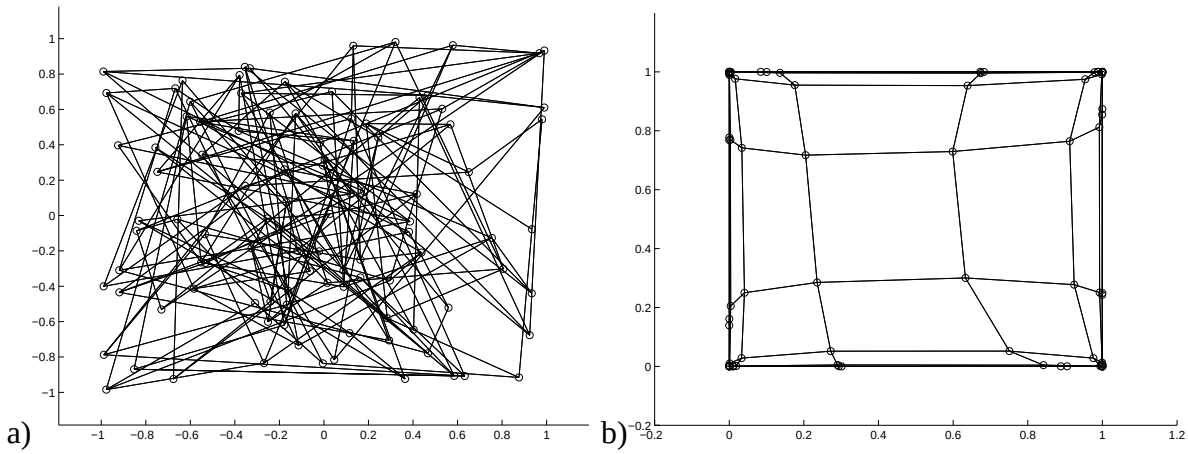


Figure 4.1: A plot of the weight vectors used to characterize a SARDNET SOM utilizing a 10x10 lattice of output nodes. Here, the SOM is used in an unsupervised learning task of two-dimensional input sequences, each ranging from two to four vectors in length. Plot a) shows the initial configuration of random weight vectors of the SOM as plotted in two dimensions. Plot b) shows the same SARDNET SOM after being trained using the SARDNET procedure outlined in Table 4.1.

inputs are selected solely from some finite alphabet, or set of *candidate vectors*, this property can generally be expected in a reasonably-sized, well-trained SARDNET SOM. However, where vector contents can take on not just some finite number of values, p, but any of an infinite number of values (e.g. real valued), unique output map creation cannot be guaranteed.

To demonstrate this, let X and Y each be vector sequences of length k used as input to SARDNET SOM SD_{EX} such that X = $\mathbf{x}[1]$, $\mathbf{x}[2]$, ..., $\mathbf{x}[k]$ and Y = $\mathbf{y}[1]$, $\mathbf{y}[1]$, ..., $\mathbf{y}[1]$. We construct sequences X and Y such that they comprise the same vectors from position 0 up until next-to-last position, k-1, in each respective sequence (i.e., $\mathbf{x}[i] = \mathbf{y}[i], 0 \leq i \leq (k-1)$.) As such, the series of output maps produced by the SARDNET SOM SD_{EX} will certainly be equivalent whether given X or Y up to vector k-1 of either. An issue can easily arise if vectors x[k] and y[k] both are closest to the weight vector of the same output node but $x[k] \neq y[k]$. In this scenario, this will likely result in the same output value, 1.0, being output at the same winning node, leading ultimately to equivalent output map representations between the two input sequences even though the sequences are not equivalent (i.e., $SD_{EX}(X) = SD_{EX}(Y)$ but $X \neq Y$.)

The problem occurs because the same output map node is selected and the same output value is pre-determined even though the input vector at that time step is different. Ideally, rather than just having the winning map node produce the same pre-determined output value when it wins, a more descriptive output score than 1.0 could be calculated and produced which could most probably be different for two differing input vectors, even when they select the same winning node.

By knowing a priori the set of anticipated, or candidate, inputs expected to be seen by the SOM, more informative map node activation values for the SARDNET SOM can be developed. Such a modification in its own right could potentially offset the effect of output map ambiguity substantially in the standard SARDNET SOM.

4.3 Candidate-Driven SARDNET

As a response to this issue of prevailing ambiguity in SARDNET SOMs, I devised a more informative output node dynamic which allows for more telling real numbered output node activations than just the standard 1.0 output suggested by James et al. ([21].) Suppose it is known a priori the entire set of possible input vectors, termed candidate vectors, seen somewhere in any input vector sequence anywhere in the training data. Let C denote the set of candidate vectors and **x**[t] denote the input vector at discrete time step, t, of the current n-length input vector sequence, X = **x**[1], **x**[2], ..., **x**[n].

First, note that the training procedure remains unchanged from that used for single-winner SARDNET SOMs described in Table 4.1. Some winning node o_j, associated to weight vector w_j, can be found in the same manner as is detailed in the original SARDNET output scheme. However, in calculating the output of a winning node in this modified version of the SARDNET SOM, rather than use the algorithm outlined in Table 4.2, the following variables must first be calculated,

$$\mathbf{c}_x = argmin_j ||\mathbf{c}_j - \mathbf{x}_t||, 1 < j < m \tag{4.1}$$

$$\mathbf{w}_c = argmin_k ||\mathbf{w}_k - \mathbf{c}_x||, 1 < k < n \tag{4.2}$$

where m is the number of candidate vectors in C and n is the total number of nodes in the SOM lattice. Vector **x**[t], again, denotes the single input vector at time step t of the current input vector sequence, X, to the SOM while vector $\mathbf{w}_k$ can then be defined as the weight vector which corresponds to output node o_k. Hence, the variable $\mathbf{c}_x$ signifies the closest candidate vector in C to the input , **x**[t], at time t of the current input vector sequence. Vector $\mathbf{w}_c$ is therefore the weight vector of the trained SARDNET SOM which most corresponds to that best candidate, $\mathbf{c}_x$.

The following equations calculate gaussian, or radial basis, measures ranging from 0 to 1 indicating the proximities of the winning node to the predicted candidate vector (eq. 4.3) as well as the current node to the winning node in the output lattice (eq. 4.3):

$$g_{ci} = e^{-\frac{||w_c - x_t||^2}{2\delta^2}} \tag{4.3}$$

$$g_{cn} = e^{-\frac{||o_c - o_n||^2}{2\gamma^2}} \tag{4.4}$$

where $\delta > 0$ and $\gamma > 0$ are radius terms which each determine width for their respective gaussian curves listed above and $||...||$ indicates Euclidean distance. Vector o_c denotes the (i,j) lattice position. By combining these two terms, a new, more meaningful real-valued output can be produced at a SOM map node which can be treated as a gauge for its closeness to the intended candidate vector :

$$Output(o_c) = g_{cn} * g_{ci} \tag{4.5}$$

See Table 4.3 for the entire candidate-driven SARDNET SOM output procedure.

One way of looking at this new candidate-based output scheme is that the g_{ci} term indicates the proximity of the weight vector of the output node closest to the winning candidate is the actual input vector. A perfect match, where the candidate output node has a weight vector equivalent to the t-th input vector of X (i.e., $\mathbf{w}_n = \mathbf{x}_t$), will yield a g_{ci} of $e^0 = 1.0$. Alternately, the further a candidate output node's weight vector is from x_t, the closer the term approaches 0. The second

Output Dynamics of a Trained Candidate-Driven SARDNET SOM
1) initialize all node outputs to 0
2) select input vector **x**[i] in sequence X,
3) multiply output of all marked nodes by $0 < \mu < 1$
4) determine closest *unmarked* winning node and set its output to $g_{ci} * g_{cn}$ (Eq. 4.3)
5) mark winning node
6) repeat from 2) until sequence X is completed.

Table 4.3: Outline of the procedure for producing candidate-driven outputs in the SARDNET SOM once presented with input vector sequence, X = **x**[1], **x**[2], ..., **x**[n].

term, g_{cn}, indicates how far the node currently being looked at is from the weight vector closest to the winning candidate. If the current node has a weight vector equivalent to the candidate, this term will work out to be $e^0 = 1.0$ as well. In the event that both cases are true, the terms together yield an output of $g_{wc} * g_{wi} = 1.0 * 1.0 = 1.0$ just like in the standard SARDNET procedure. Hence if certain weight vectors of a SARDNET SOM end up being made equivalent to the set of candidate input vectors, the resulting candidate-driven output scheme can be reduced to the standard SARDNET output scheme.

The scale of this output given at any node is now a much more descriptive indicator of the closeness of a node to the input vector with respect to the set of expected vector inputs than in the original SARDNET model. Hence, the SOM does not fall into the same pitfalls demonstrated in the previous SARDNET example, which is content to merely place a '1' as output to any winner. Though outputting ambiguous maps using this format is still somewhat of a possibility, it tends to occur at a much reduced rate.

Following training, there will tend to be one node in the candidate-driven SARDNET SOM's output lattice whose corresponding weight vector is closer than any other to any given candidate input. In this case, if this candidate input vector's "best node" has a weight vector that is not equivalent to itself, the calculated output at that node when selected may approach, and yet never equal,

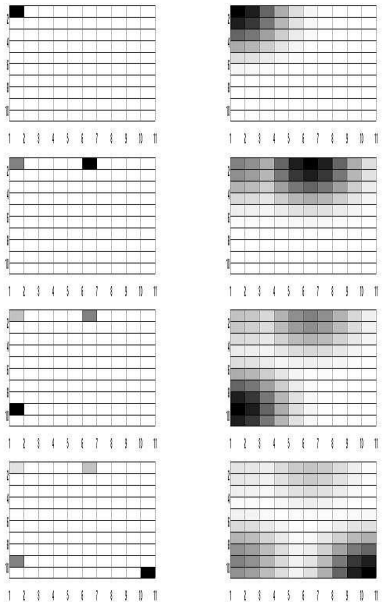
Output Dynamics of the Candidate-Driven SARDNET SOM
1) initialize all node outputs to 0
2) select **x**[j] in sequence X,
3) multiply output of all marked nodes by $0 < \mu < 1$
4) for all nodes, y[i], in SARDNET SOM, SD,
- set node output at y[j] to $g_{ci} * g_{cn}$
5) repeat from 2) until sequence X is completed.

Table 4.4: Procedure for producing multi-node output maps in a candidate-driven SARDNET SOM once presented with input vector sequence, x[1], x[2], ..., x[n].

1.0 due to the manner in which Equation 4.3 was constructed. As an additional, yet optional, step one can elect to take at the close of the initial training phase of the candidate-driven SARDNET SOM, one can choose to find the closest node to each candidate and set its corresponding weight vector equivalent to that same candidate input vector. This would serve to force outputs to be set precisely to 1.0 once inputs presented to the system belong precisely to the set of expected candidate vectors. Such behavior would once again closely resemble that of the standard SARDNET procedure outlined in the previous section.

This variation on the standard SARDNET SOM output procedure is most ideal for domains in which the number of expected, or most sought after, input vectors are countably finite and available for training. However, if such a candidate input vector set is not available or is infinite, this method would be seriously compromised.

This map node output scheme fulfills the desired characteristics described previously and looks to differentiate all different input vectors that seek to select the same winner. This, however, still does not completely guarantee uniqueness, but it comes significantly closer than that of the original SARDNET architecture.



a) b)

Figure 4.2: This figure illustrates the contrast between two differing forms of candidate-driven SARDNET SOM output schemes. Specifically, two snapshots above demonstrate outputs produced by the same trained candidate-driven SARDNET SOM using a) the standard output scheme of Table 4.3 and b) the multi-node output procedure outlined in Table 4.4. Top to bottom, both pictures show the respective output generated by the trained SOM at each time step when presented with each vector of the same four vector sequence as input (section 4.3.1.)

4.3.1 Multi-node Candidate-Driven Output Mapping

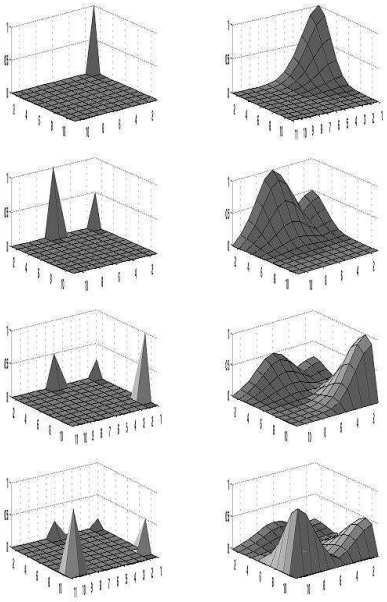
One other benefit to using the candidate-driven version of the SARDNET architecture is that this is a method by which the SOM can be used to produce output not only from nodes which have won, but by which all nodes across the entire SOM lattice may be used to produce outputs (see procedure in Table 4.4.) The standard SARDNET output procedure only allows for outputs at past and current winners. What tends to result as output maps is reminiscent of gaussian mounds centered around winning nodes (Figure 4.3.1).

The terms g_{wc} and g_{wi} combined allow for the formation of Mexican hat or gaussian bell curve

structures in output maps. Each Mexican hat structure can be seen to emanate from the winning nodes outward across the SOM lattice. The g_{wc} term can be regarded as the initial height of each gaussian mound. So if the g_{wc} term ends up equaling .5, a gaussian bell curve with a height of 0.5 should result centered at the winning node outward to the rest of the SOM lattice. This phenomenon of Mexican hat activations over a map of competing neurons is often observed in actual neuro-biological studies of the human brain ([16], [12]). The capability of the candidate-driven SARDNET SOM to output such Mexican hat phenomena across multiple SOM nodes can potentially be useful in providing more realistic models of sequential map formation in the human cortex among competing neurons than the standard SARDNET algorithm.

Take Figures 4.3.1a. and b., for instance. Both figures are meant to signify an example of the progression of activity patterns on a candidate-driven SARDNET SOM en route to generating a final output map to uniquely represent the input sequence. The SARDNET SOM consisted of a 10 x 10 output lattice of map nodes, each of which is represented as a square in a 10x10 grid of outputs. The outputs of the map nodes are represented on a grayscale, where the color black signifies a map node output of 1.0, a white square signifies no output, and the intensity of a gray square indicates a map node's output value to either extreme. In other words, light gray would signify a value closer to 0 while a very dark gray may signify an output value very close to 1.0.

In Figure 4.3.1 a., the normal progression of activation patterns on a trained candidate-driven SARDNET SOM is shown when given a four-length vector input sequence. Notice here that only one new map node, the winning node, is allowed to give an output at every new time step when a new input vector in the sequence is introduced. Figure 4.3.1b. shows the resulting activation patterns from the same SARDNET SOM presented with the same exact four-length input sequence but in using the multi-output scheme of Table 4.4 in which all map nodes have the opportunity to produce outputs. What differentiates these two sets of candidate-driven SARDNET SOM activity patterns lies in determining *w*hich map nodes are allowed to produce output values: winning map nodes only or all nodes in the SARDNET SOM's lattice of output nodes. Figure 4.3 merely shows the same series of SARDNET map activations from Figure 4.3.1b. but in three dimensions (i.e.



a) b)

Figure 4.3: These figures illustrate the same contrast of candidate-driven outputs as shown in Figures 4.3.1a-b. Rather than represent the real candidate-driven SARDNET SOM outputs in grayscale, however, they are plotted in a third dimension to better illustrate the formation of Mexican hat output structures as is often observed in neuro-scientific studies of cortical activation.

representing map node output values in the Z-axis as opposed to grayscale.) Here, the spreading Mexican hat activations described previously as what the multi-output SARDNET activation scheme is capable of producing becomes more visually evident.

4.3.2 Demonstrating the Utility of the Candidate-Driven SARDNET Enhancements

The major improvement of this modification to the SARDNET SOM is that the new modification lends itself to fewer occurrences of ambiguity.

Here I define three similar input vector sequences, I_1, I_2, and I_3:

$$I_1 = \langle [1.0, 0.0], [0.0, 1.0] \rangle, I_2 = \langle [0.9, 0.31], [0.18, 0.65] \rangle, I_3 = \langle [0.79, 0.02], [0.23, 0.85] \rangle.$$

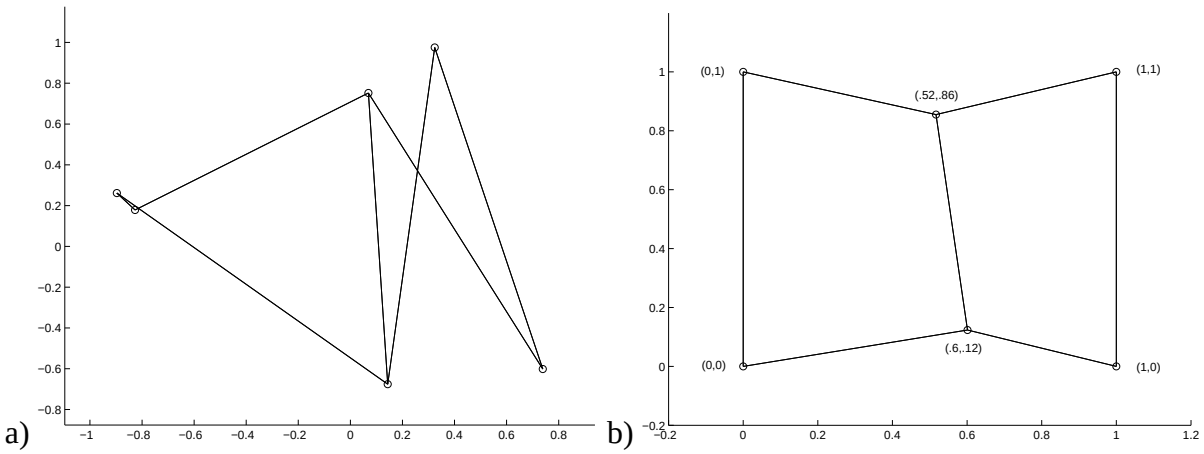


Figure 4.4: a) The initial plot of a 2x3 SARDNET SOM discussed in Section 4.3.2 before training. b) Plot of the same SARDNET SOM after training on two-dimensional sequential vector data made up entirely from vectors of candidate vector set $\{[00]^T, [01]^T, [10]^T, [11]^T\}$.

Let $SD_{4.4}$ denote the original output scheme for an example Candidate Sardnet SOM using a 2x3 lattice of output nodes which was previously trained on a number of input sequences ranging from two to four vectors in length, one of which being sequence I_1 listed above. The corresponding before-and-after training weight plots are shown in Figure 4.4. When presented with input sequences I_1, I_2, and I_3, all three final resulting 2x3 output patterns come out looking exactly identical:

$$SD_{4.4}(I_1) = SD_{4.4}(I_2) = SD_{4.4}(I_3) = \begin{array}{|c|c|c|} \hline 0.5 & 0.0 & 0.0 \\ \hline 0.0 & 0.0 & 1.0 \\ \hline \end{array}.$$

This is because, though they may be noticeably distinct, the input sequences trigger the same winning nodes and, hence, yield a 1.0 output at the same nodes regardless. The candidate-driven output scheme, however, takes into consideration proximity of the winning node to the closest candidate vector in determining its final output activation pattern. As such, given similar input sequences I_1, I_2, and I_3, identical final output patterns are far less likely when using the same 2x3 SARDNET SOM but with the modified output dynamics (denoted by $CDSD_{4.4}$):

$$CDSD_{4.4}(I_1) = \begin{array}{|c|c|c|}\hline 0.0 & 0.0 & 0.5 \\ \hline 1.0 & 0.0 & 0.0 \\ \hline \end{array}, CDSD_{4.4}(I_2) = \begin{array}{|c|c|c|}\hline 0.0 & 0.0 & 0.404 \\ \hline 0.734 & 0.0 & 0.0 \\ \hline \end{array},$$

$$CDSD_{4.4}(I_3) = \begin{array}{|c|c|c|}\hline 0.0 & 0.0 & 0.457 \\ \hline 0.86 & 0.0 & 0.0 \\ \hline \end{array}$$

The potential for significant reduction in the size of SARDNET SOMs using the candidate-driven modification presented here is important as well. To offset ambiguity in the standard SARDNET SOM architecture, increasing the number of map nodes tends to reduce the occurrence of ambiguous output maps. This is because with an increased number of map nodes comes much improved opportunity for differing input vectors to activate differing nodes based on proximity to their respective weight vectors.

Using this modification, however, one would be harder pressed to find two input vectors which activate the same winning node in the SARDNET SOM with the same output activation. As such, in looking to create SOMs which give more unambiguous outputs, more compact map architectures with fewer nodes, and hence, fewer calculations, can be designed. Since now two similar vector inputs can be represented differently by the output of the same winning node, as opposed to merely outputting a 1.0 both times, even fewer output nodes than the already reduced number cited by James [21] can be used to uniquely encode an input sequence.

4.4 Contributions of the Chapter

The primary contribution presented in this chapter is the modification I made to the SARDNET self-organizing map, a neural model designed to accept and uniquely classify sequential input data, enabling it to produce more unique representations of input sequences. The SARDNET self-organizing map, although designed to output unambiguous map activations for distinct input sequences, is shown by example to generate non-unique output maps in similar situations. Using

my modification, more meaningful node outputs are produced which consider, among other things, the proximity of an input vector to the intended vector it was supposed to resemble in calculating its output rather than indiscriminately producing a 1.0 value as suggested in [21]. As a result, the modified candidate-driven SARDNET SOM tends to yield more unique output maps than the standard version. If the single winner-take-all selection is set aside for the multi-output scheme in which all output nodes are capable of firing, interesting Gaussian mounds become apparent in output maps reminiscent of Mexican hat formations described in the neuro-scientific literature regarding spreading cortical activation in the brain. This modified candidate-driven SARDNET SOM holds promise in being a potentially useful tool for capturing sequential cortical brain behavior for use in time-varying computational cognitive behavior studies.

Chapter 5

Recurrent Distal Learning in Modeling the Acquisition of Phoneme Sequence Generation Behavior

In this chapter, the effectiveness of the recurrent neural network modifications made to the existing distal supervised learning framework introduced in Chapter 3 is demonstrated on a very complex application. Namely, an experiment is designed in which a recurrent neural network is created to undergo the same complex process that humans are believed to go through en route to acquiring the ability to produce or generate sequences of phonemes to articulate words. Distal supervised training of a recurrent neural network is demonstrated despite it operating in a very complex composite mapping of two non-linear functions, one constructed using the smooth mapping procedure discussed in Appendix B and the other being a Candidate Driven SARDNET SOM (Chapter 4) which is designed to take on the role of associative memory as it is thought to be utilized in the phoneme sequence acquisition process in humans. The charts shown at the end of the chapter demonstrate that not only does learning occur in such a difficult sequential environment, but that there is indeed a strong case for utilizing approximated teacher forcing (also introduced in Chapter 3) to improve memory layer updates and, subsequently, acquisition of sequence generation behavior in distal settings.

5.1 Phoneme Sequence Generation

Phoneme sequence generation refers to the process by which humans manufacture very deliberate and specific strings of individual minimal units of spoken language, or phonemes, through motor activity in vocal organs in order to communicate with other humans. The acquisition and ongoing use of this cognitive behavior is certainly not well understood and many researchers continue to struggle to explain and model the inner workings of the process (Roelofs ([50]), Dell [10], etc.)

Previous attempts at computational simulation of phoneme sequence generation vary significantly in approach and in motivation. Dell [10] developed Spreading Activation Theory (SAT) for speech production which is favored by many and has been very influential. In it, Dell details a connectionist model employing nodes working, initially, in parallel and, subsequently, in serial through four levels of speech word form classifications.

The WEAVER (**W**ord-form **E**ncoding **A**ctivation and **VER**ification) model (Roelofs [50], Levelt, Roelofs, Meeyer [30])expands on Dell's model of spreading activation and addresses some of its shortcomings to create a more encompassing 6 level model of speech production. Neither model, however, addresses the process by which this cognitive function is acquired over time. In particular, neither model attempts to define the role of internal models or even the role of memory retrieval from associative memory in the human cortex in acquiring this function.

Guenther ([17]) designed a very telling model of single phoneme production which dealt with the mapping from motor phoneme to orosensory sensation (i.e. the tactile sensation of the phoneme being uttered.) His study proved to be very enlightening as he was able to replicate various commonly known traits or phenomena generally observed in the production of learned phonemes. Among the phenomena he was able to demonstrate was co-articulation, in which the sound of a phoneme depends directly on the previously articulated phoneme. His model, much like the model presented here, conducted a "babbling" stage to properly set the initial parameters of the system.

The fundamental difference between Guenther's model and the work discussed here is that, primarily, his model was designed to produce single phonemes in the study utilizing orosensory

inputs. The phonemes his model produced had a local, not distributed, representation scheme (i.e. a single unit being on uniquely identified a particular phoneme.). Also, he did not at all represent stored distributed cognitive representations of phonemes in associative memory as was done in this study.

In addition, there was no attempt to represent an internal model for speech production in Guenther's simulation of phoneme acquisition. Internal models, such as motor programs believed to exist in the cerebellum of the brain [72], seek to correctly imitate the mapping from motor commands to their respective cognitive representations. There is a growing body of evidence touting the existence of internal models in the brain which, through continued interaction with the external world, acquire the ability to forecast the consequence of a series of motor actions. This internal model is now considered key in acquiring all types of higher level cognitive motor function capabilities such as moving limbs and speech acquisition tasks ([70], [71]). The model discussed here incorporates all of these aspects in its present form.

In discussing the development of previously constructed phoneme sequence acquisition models, it must be made clear that the task has generally been attacked in pieces, not as a whole. For instance, the storing of heard words in associative memory, the producing of phonemes and sound due to commands to the motor cortex emanating from Brocas area, the way sounds enter the ear and stimulate the auditory cortex, etc. - each is so complex as to be studied and modeled separately by researchers extensively over the years. As such, the attempt made here to create a model of phoneme acquisition sequence as a whole is quite an ambitious task. In order to create such a model, it was required that the task be simplified to some extent.

5.2 Single Phoneme Production Model

5.2.1 Model

First, a model of acquiring the ability to generate a correct single phoneme (e.g. /b/, /ae/, or /t/) from its intent using the expected auditory phoneme was designed. This model is implemented by

using a standard, non-sequential distal supervised neural network where there is a standard non-recurrent feedforward neural networks for both the distal learner and forward model. This was done in order to gage how difficult the harder, more complex, sequence acquisition task would be. Also, in creating this simpler setup, the environment function, to be discussed later, could be tested for validity and effectiveness in training the distal learner. Details of the challenges encountered in attempting to model these ambitious tasks are outlined in the upcoming sections.

This neural model was intended as a preliminary step in a very ambitious attempt to create a system inspired by the complex process through which people: 1) accept, process, and store language phoneme sequences of a heard word as a series of neural firings in the auditory cortex and associative memory, and 2) subsequently produce the correct motor phoneme sequential response via interactions between Broca's area and the brain's primary motor cortex. The sounds produced as a result of the latter interaction, after passing through the environment (air, environmental noise, auditory system etc.), will again evoke the intended neural representation in associative memory after being processed by the auditory cortex.

The model, inspired by the organization of the centers of a human's brain responsible for speech production, is presented with some intended phoneme input stimulus and its known auditory phoneme representation. Ultimately the goal of this exercise is to create a neural model capable of learning the mapping from phoneme intent to the corresponding motor cortex response which will eventually yield the desired activations in the auditory cortex. In turn, this exercise is meant to imitate the human brain's ability to learn to produce single intended speech sounds from memory en route to the eventual acquisition of phoneme sequence, or full word, skill.

The portion of this model discussed here will make use of a more standard form of non-recurrent distal learning in order to complete its learning task. The distal learner must learn to produce the correct motor phoneme activations in the primary motor cortex given a unique static phoneme intent vector as input such that, when transformed by the environment, this will correspond distally to the desired auditory phoneme representation in associative memory. This is done by having some neural connections attempt to model the external motor to auditory phoneme

transformation and using those same connections to assist in updating the weights of the learner. This internal forward model can be trained by generating random motor responses and associating the ensuing neural firings in the primary auditory cortex to that motor response. As discussed in Section 2.5, there is some evidence which suggests such forward models do indeed exist in the brain (likely located in the cerebellum ([4],[72]).)

A source of inspiration for this approach is that, when looking at speech development in infants, the 'babbling' a baby does in the early stages appears to be a necessary process for the development of the forward model responsible for predicting the outcomes of various motor actions involving his/her speech organs. Here, the infant, who one might suggest "just likes to hear herself", makes arbitrary noises through motor commands and can eventually associate a particular heard sound to the motor commands that it resulted from. Once this "mapping" is ascertained, the baby can thereby reproduce that sound whenever he/she intends to. Formation of an effective forward model for producing phonemes, however, is generally not completed by the time an infant's intent surfaces to duplicate known auditory phoneme sequences. Over time, a cycle of producing increasingly improved, though incorrect, motor action of an intended sound based on what is stored in associative memory must be repeated continuously to achieve the desired result. Intent to repeat new words and phonemes heard spoken from adults will increase the infant's set of intended phonemes.

5.2.2 Environment

The environment used in this study makes use of the table in Appendix B which lists the component features that make up motor and auditory phonemes needed to construct a smooth mapping from the former to the latter. It is important that this mapping be smooth and differentiable to help facilitate the learning in this model's forward connections. The manner in which this mapping is constructed, as well as the many considerations which must be addressed, is discussed in Appendix B.

The training method used in this computational model is the standard form of the distal super-

vised learning method discussed in Section 2.5 to train the internal model and motor output area together in series as if they were one four-layered neural net but to propagate different deltas to the appropriate components to achieve the desired results (Figure 2.11).

All input vectors to the distal learner used in this particular study take on the form $[\underbrace{0.1, ..., 0.1}_{(j-1)}, 1, \underbrace{0.1, ..., 0.1}_{(n-j)}]$, where $1 \leq j \leq n$ and vector length n varied based on the dimensionality of the static input, learner's output, and environment output:

1. **S**tatic input to phoneme generation area (phoneme intent) : vector length (n=39) corresponds to the number of possible phonemes, with one unique bit set to one and all others set to minimum value 0.1.

2. **P**roximal output from phoneme generation area (motor phoneme) : vector length (n=20) corresponds to the number of features through which distinct motor phonemes can vary (see Appendix C). Each bit is set to .1 or 1, where .1 corresponds to a '.' and a 1 corresponds to a '+'.

3. **E**xternal environment/internal model response (auditory phoneme) : vector length (n=34) corresponds to the number of features through which distinct auditory phonemes can vary. Each bit is set to .1 or 1 (see Appendix C).

The sets of motor and auditory phoneme feature vectors used in this preliminary study are listed as tables in Appendix B. Twenty-four consonantal and fifteen vocalic motor phoneme feature vectors were merged together to form the 39 total motor phonemes used to formed the basis of the input space for the motor-to-auditory smooth environment mapping constructed in the manner described in Appendix B for this distal supervised learning task. Likewise, the 39 auditory phoneme feature vectors are gathered in a similar manner to form the basis of the environment's distal output. At the same time, the 39 auditory feature vectors are used as distal target outputs for use in training the distal learner. Minimum values of .1 are substituted for zero values in each phoneme vector used here as zero target output values have been shown to be problematic in the

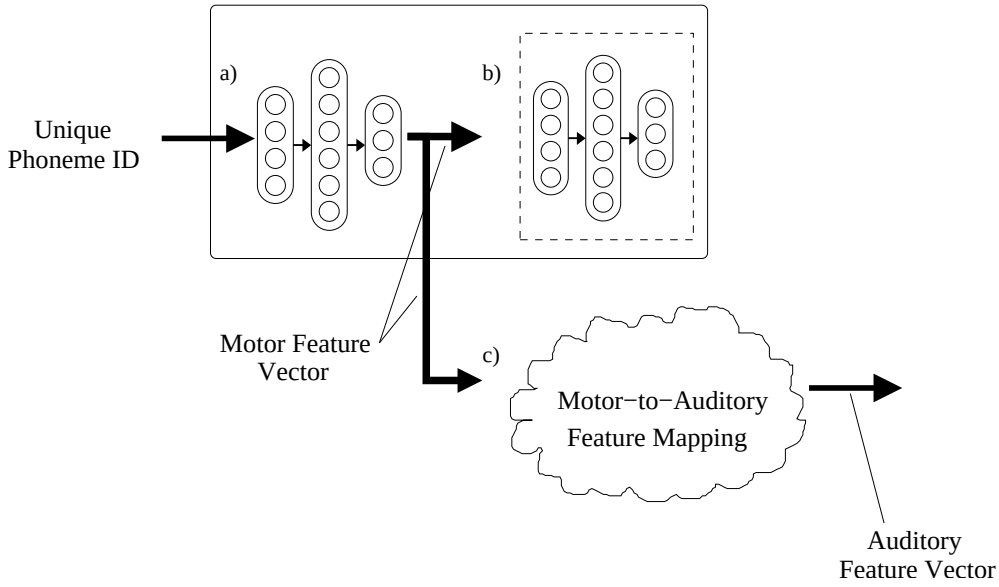


Figure 5.1: This figure demonstrates the setup for the single phoneme acquisition model described in Section 5.2. Here a distal learning neural network (labeled as a)), with the assistance of the forward model (b)), is designed to learn to reproduce the correct motor phoneme vector when provided only a unique phoneme intent vector and its corresponding distal auditory phoneme vector. This distal learner produces motor phoneme vector outputs and obtains auditory vector outputs while operating in the motor-to-auditory phoneme transformation environment mapping (c)) (section 5.2.2).

training of neural models using sigmoid activation functions in their output layers. Also, these minimum values are used to assist in creating the smooth environment function using the phoneme tables of Appendix C to offset difficulties encountered when introducing zero values to the smooth mapping algorithm discussed in Appendix B.

5.2.3 Distal Learner / Forward Model Designs

This preliminary neural network model has the following capabilities :

- various gradient descent methods such as adaptive learning and momentum.
- one hidden layer (size determined experimentally)

- sigmoidal output at hidden and output layers

The forward neural model is a standard two layered neural net which is trained primarily using the adaptive learning rate gradient descent method. The algorithm in Appendix A outlines the procedure for training the motor output area and the forward model. Figure 5.1 is a diagram of the architecture in which the distal learner and forward model work in tandem to handle this particular distal supervised learning task.

5.2.4 Results

The model described here has exhibited good success in handling this particular learning task. Despite having to learn in an environment function which maps actions from one sizable domain to another (i.e., Motor Output:$\{0,1\}^{39} \rightarrow \{0,1\}^{20}$) in the absence of the teacher to implicitly provide proximal target output values, the model is capable of learning the phoneme intent to motor phoneme mapping task at a RMSE of just under 0.1. In actuality, because of the amount of stochasticity inherent in the model (e.g. random assignment of distal learner and forward model weight vectors and in the random selection of environment interaction generated to facilitate forward model training to simulate babbling), RMSE tends to vary from .09 to .22 where a mean run terminates with an RMSE of approximately 0.15.

The current model uses the following parameters:

1. Distal neural model of motor output: Hidden Layer size - 125

2. Forward model: Hidden Layer size - 54

As you will see in section 5.3, the next step in this study involves expanding this model to accept a single static word intent vector, encoded to uniquely represent some phoneme sequence stored in associative memory, and output the appropriate motor phoneme sequence required to generate that word. By expanding on the distal learning paradigm of section 2.5, I have developed a method of training recurrent neural networks to accomplish just such a complex task (section 5.3).

5.3 Framing the Distal Recurrent Learning Architecture for the Phoneme Sequence Recurrent Task

5.3.1 Setup

The phoneme sequence generation model is loosely inspired by the way it is generally believed that a human learns to produce spoken words [5]. A vastly simplified process that humans go through in acquiring phoneme sequence generation capability is illustrated in Figure 5.2. From here on the "learner" does not necessarily refer to the human learning to speak but, rather, the cognitive region or machinery used to accomplish the acquisition of phoneme sequence generation behavior. First, a single unchanging intent or idea of a word results in the recall of the correct series of activation patterns in associative memory that the learner will try to duplicate. As such, the learner commences to generate some time-varying sequence of motor responses largely using his/her own speech organs. These motor commands cause some series of noises to result in the external world which are conducted via vibrating air molecules, along with external noise, back to the person's hearing organ. Each acquired sound is processed by the auditory cognitive region before being streamed to the associative memory region, where a very distinct series of activation patterns results.

The goal of any learning process used here would be to, wherever necessary, change the makeup of the learner's own neural connectivity such that the learner will make steady progression towards eventually producing the desired series of neural activity patterns in associative memory. Ultimately, the learner should acquire the capability to produce some series of motor commands which would be responsible for reproducing the recalled set of desired distal memory activity patterns retrieved at the beginning of the learning process. Notice that, in this particular setup, the only input provided to the learner required to produce the series of correct proximal motor behavior is the single, unchanging phoneme sequence intent stimulus.

In an attempt to develop a simulation of this approximated cognitive learning process, the re-

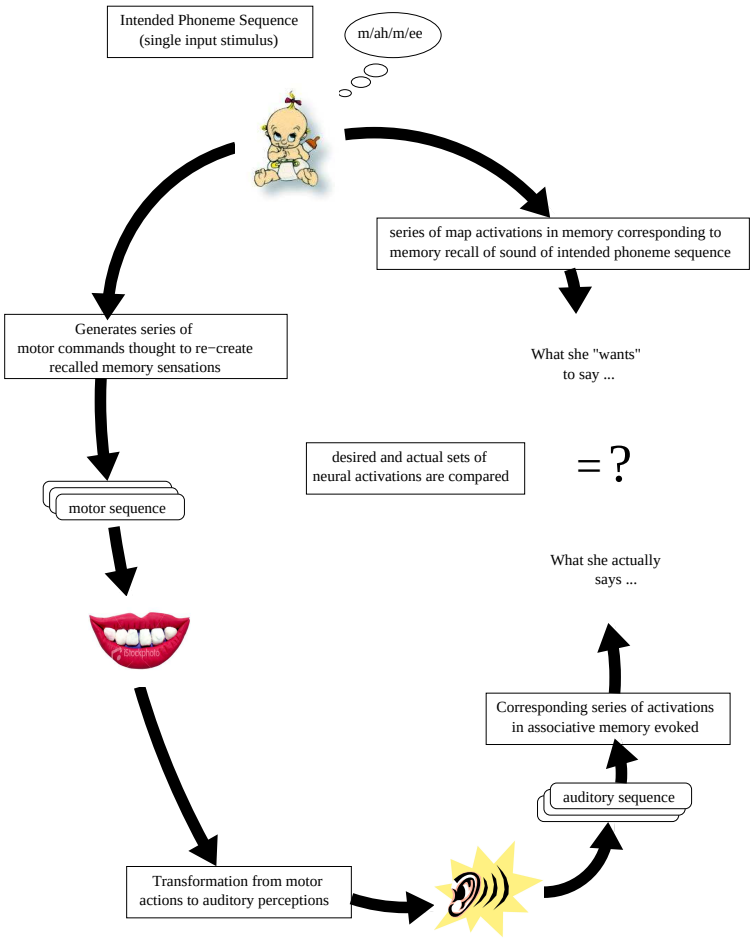


Figure 5.2: Illustrating the Phoneme Sequence Generation Domain.

current distal supervised learning architecture illustrated in Figure 5.3 was devised. In it, some learning agent is presented with a single static input stimulus which corresponds to a unique and deliberate, yet initially unknown, sequence of motor phoneme commands. What is available regarding this phoneme sequence intent input stimulus is the series of self organizing map (SOM) activations known to uniquely correspond to it. In other words, these map formations are meant to signify the stored representation of the intended word in ”memory” that the distal learner would like to duplicate. For this exercise, the task for the intended distal learning module is to then generate some sequence of vectors, in which each vector represents a motor command whose contents signify motor phoneme features that yield some unique utterance or sound. The duration of this motor vector sequence will always be assumed to be equivalent to the length of the target distal

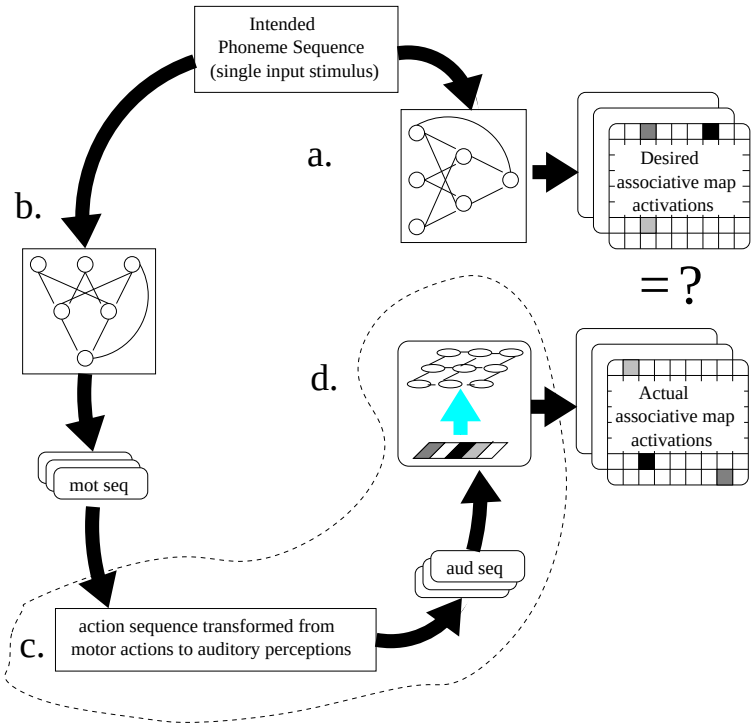


Figure 5.3: An illustration of the setup for the Phoneme Sequence Generation distal learning task. The previously-trained memory recall network (a.) provides the sequence of target memory activity patterns required to train the recurrent learner. Ultimately, the task given to the recurrent motor phoneme generating model (b.), given only a single static input word intent vector, is to learn to generate the correct sequence of motor feature vectors that, once transformed into a phoneme sequence of auditory feature vectors (c.), yields a series of activation patterns in associative memory (Candidate-Driven SARDNET SOM d.) matching those produced in memory recall.

output sequence provided for training. This sequence is then presented to the environment, which transforms this motor phoneme sequence into a corresponding sequence of "auditory" phoneme vectors which are based in auditory distinctive features (see Appendix C.) Finally, this series of auditory vectors then produces some series of neural activations to occur in associative memory that are unique to those vectors (Figure 5.2).

In this task, the motor-auditory mapping and the Candidate-Driven SARDNET SOM memory model together make up the Phoneme Sequence Generation sequential environment (signified by

the enclosed dotted area in Figure 5.3.) The purpose of such an exercise is to enable the intended recurrent neural network to learn to transform the single static intent stimulus into the appropriate sequence of motor phonemes which would ultimately and uniquely yield the target sequence of output memory activations made available at the beginning of the training run. The recurrent distal learning architecture designed to approximate the process illustrated in Figure 5.3 is shown in Figure 5.4.

5.3.2 Phonemes and Phoneme Sequences for Experiments

In developing the phoneme sequence generation model, I looked to identify: 1) a subset of key phonetic features used to describe many commonly used English phonemes, 2) a subset of the listed phonemes in the English language using this reduced feature set, and 3) a list of phoneme sequences that a distal learner could conceivably acquire and learn to generate. This reduced feature set decided on consisted of the following characteristics (note: feature categories known to be complements of each other are paired together to reduce the parameter space to be searched): 1. vocalic/consonantal, 2. strident/nasal, 3. voicing on/voicing off, 4. continuant/stop, and 5. height (high/low).

Likewise, nine binary variables were determined which could adequately address ten of the features listed in Appendix (C) believed to completely characterize the auditory reception of any English phoneme. This reduced auditory phoneme feature set includes: 1. continuant, 2. interrupted, 3. duration (on/off), 4. terse, 5. lax, 6. $F_{2,VH}$, 7. $F_{2,L}$, 8. $F_{1,L}$, and 9. $F_{1,HM}$. The terms of the form F_{1,x_1} and F_{2,x_2} refer to varying intensities of formants f_1 and f_2, respectively. Formants are peak acoustic frequencies which result from the resonance of the human vocal tract [66]. Formants f_1 and f_2 can be particularly helpful in characterizing differing vowel sounds. Variables x_1, x_2 consist of values from the set $\{L, HM, VH\}$, where 'L' means "low", 'HM' means "high medium", and 'VH' means "very high".

In ascertaining which phoneme features to use, there are certain features that are discussed heavily in the phoneme generation literature that are deemed to be very pertinent (e.g. vocalic/consonantal.)

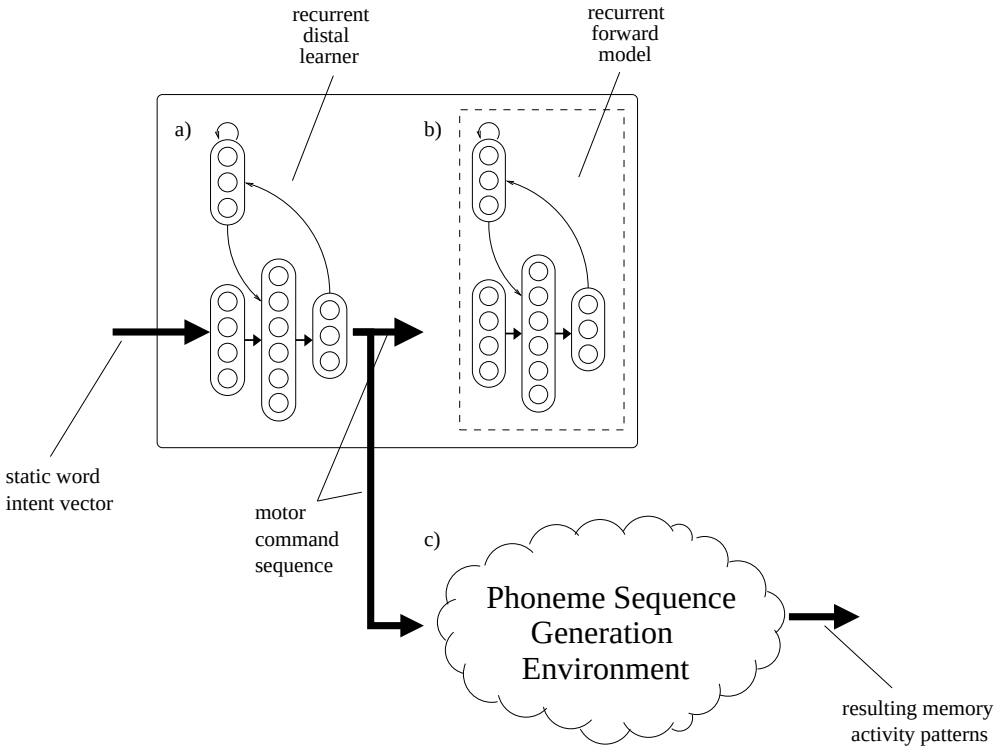


Figure 5.4: Recurrent distal learning architecture used to model the Phoneme Sequence Generation framework of Figure 5.3.

Also, high preference was given to those features that were binary in nature or were the exact complement of another feature across all phonemes, vowels and consonants alike. In other words, the presence of one feature signified the absence of another (e.g. voicing on/off, continuant vs. stop, etc.)

Lastly, there were phonemes I deemed important (in particular, 's' and 't') that could not be described without the use of certain very specific phonetic features. I wanted to use these phonemes since they are included in so many viable English phoneme sequences. I also wanted to use them because of their high capacity for clustering with other consonants like s/t and t/r. This describes a phenomenon in which two phonetic consonant sounds can be compounded together without the use of vowel sounds in between (e.g. "/s/+/k/", "/n/+/t/", etc.)

I then grouped all phonemes which could be described in the same manner through the chosen subset of features and picked the phoneme in each group which I believed could help construct the

Motor Features	/s/	/t/	/r/	/aw/	/ih/
Consonantal/Vocalic	1	1	1	0	0
Voicing (on/off)	0	0	1	1	1
Continuant/Stop	1	0	1	1	1
Strident	1	0	0	0	0
Height (high)	0	0	0	0	1

Table 5.1: Reduced List of Phonemes and Their Corresponding Distinctive Motor Features.

most phoneme sequences with which to do this study. The group of phonemes I assembled is listed by its corresponding binary feature sets in Tables 5.1 and 5.2.

Table 5.1 lists the five phonemes that the recurrent distal phoneme sequence generator is expected to learn and utilize in order to successfully replicate the list of phoneme sequences stored in this study in a simple neural model of associative memory. Note that nine features are mostly listed as pairs of complements. In order to shrink the search space of the distal recurrent learner, five binary parameters are instead used in which both ’0’ and ’1’ values hold significance. Hence, for example, phoneme /s/ can be described as consonantal, strident, and continuant, where the ’1’ denotes the presence of the first in their corresponding paired binary features. In addition, however, it also contains non-voicing characteristic, as the ’0’ entry denoting the absence of the first of a paired parameter (in this case, voicing) implies the presence of its respective complement (the second of the paired features.)

Likewise, Table 5.2 lists the same five phonemes shown in Table 5.1 but described using auditory characteristics. Understandably, the auditory characteristics which make up each phoneme uniquely are mostly different than those used in the motor feature listing. Here, however, there is no need for paired complementary binary features. Not even duration (on/off) could be classified as a strictly complementary feature as vowel phonemes do not use either. These auditory feature vectors were used to construct the phoneme sequences listed in Table 5.3 which were used

At the culmination of this process, a small subset of the known English phoneme alphabet

Auditory Features	/s/	/t/	/r/	/aw/	/ih/
Continuant	1	0	1	0	0
Interrupted	0	1	0	0	0
Duration (on)	1	0	0	0	0
Duration (off)	0	1	1	0	0
Tense	1	1	0	0	0
Lax	0	0	0	1	1
$F_{2,VH}$	0	0	0	0	1
$F_{2,L}$	0	0	0	1	0
$F_{1,L}$	0	0	0	0	1
$F_{1,HM}$	0	0	0	1	0

Table 5.2: Reduced List of Phonemes and Their Corresponding Distinctive Auditory Features.

was determined for use in this study. Five phonemes, three consonants (s/ t/ r) and two vowels (ih / aw), were deliberately selected which could be uniquely represented by the reduced set of pertinent phoneme features chosen. Using these very common phonemes, a list of 15 phoneme sequences (Table 5.3) was compiled from the English language, each possessing anywhere from 2-5 phonemes. Some of these fifteen sequences contained phonemes which repeat at some point in the sequence to increase the challenge and authenticity of the study. These phoneme sequences were ultimately used as training data for a the Candidate-Driven SARDNET SOM that was created to represent associative memory for the distal sequential learning task. Following training, their resulting SARDNET output associative activations then became the only representation of this list of phoneme sequences used anywhere in the remainder of the simulation (i.e. these phoneme sequences were never again seen or used during training.)

The disconnect between motor and auditory feature space could be accomplished by a smooth mapping technique I developed for the purpose of this study, which is capable of transforming a finite mapping into one which is smooth and continuous for all inputs (Appendix B.)

S / IH / T
R / IH / S / T
IH / T
R / AW / T
S / T /AW / R
T / AW / S / T
S / AW / S
AW / T
S / T / R /AW
R / IH / T
S / IH / S / T
S / AW / T
S / T /IH / R
R / AW
R / IH / S / T / S

Table 5.3: List of Target Phoneme Sequences.

5.3.3 Memory Recall of Associative Map Distal Target Sequences

I employ a neural network to supply the target sequence vectors necessary for training the distal recurrent learner. Knowing that the human brain does not explicitly store physical target distal sequences, this neural network is supposed to represent the memory recall of the series of associative memory map activations which occurs when a phoneme sequence is decided upon. This mechanism is what provides the associative memory activations which serve as distal target sequences used to drive training of the distal recurrent learner. This neural network employs a self-halting mechanism which allows it to output varying length vector sequences depending on the input stimulus, which, in this case, is the single intended phoneme sequence vector. It is trained to produce a predetermined halting vector when it decides to end production of the sequence. Although the

self-halting mechanism was used here successfully for this standard recurrent neural network, the same feature proved to be more problematic to employ for the distal recurrent neural network of interest in this study. More research is needed on determining how to more effectively implement this feature for state-less distal sequence generation tasks.

Once the sequence was generated and the recall done, it could be used as the desired distal targets employed to drive training of the distal recurrent learner. Successful training of the distal learner can now be defined as the extent to which the learner is capable of producing the correct series of motor phonemes which will ultimately yield these memory associative maps through interaction with the environment.

In this setup, there is an environment much like that described in the previous single phoneme generation preliminary study. What is different is that the environment accepts not one, but a sequence of motor phoneme commands supplied to it via the distal learner.

The environment in this study is a composite mapping comprising two main components: 1) the smooth mapping procedure which exists to transform motor phoneme feature vectors into some corresponding equivalent auditory feature vector in auditory feature space, and 2) the self-organizing associative memory model trained, a priori, to uniquely map the fifteen chosen phoneme sequences (Figure 5.5). The composite sequential mapping referred to here as the phone sequence generation environment ultimately takes in as input a sequence of real-valued motor phoneme vectors, maps them into some corresponding sequence of auditory phoneme vectors, and outputs a corresponding sequence of activity patterns in the associative memory model. The idea is that, throughout training of the recurrent distal learner, associative memory activation maps resulting from the learning agent's proximal motor command sequences could be compared to the target associative memory map sequences generated by the neural network representing memory recall.

5.3.4 Environment

Please note that the "environment" as described here does not solely comprise the *e*xternal environment which maps individual sounds uttered by the learner into heard auditory phonemes. That

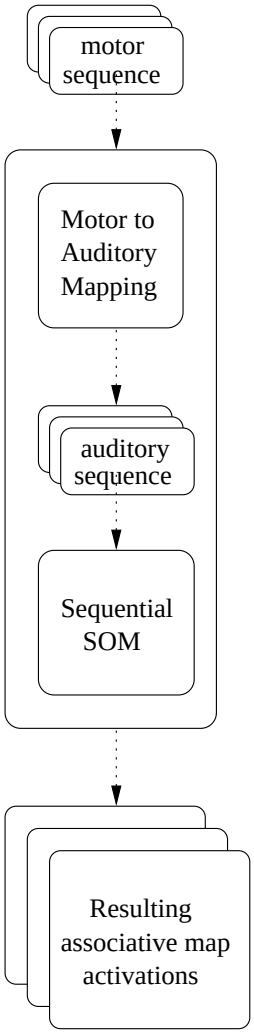


Figure 5.5: The Phoneme Sequence Generation Environment.

portion of the environment which physically lies external to the learning agent is just the first component of the entire distal sequential environment used in this application. As the environment in a distal setting is required to map proximal actions (in this case, sound-generating motor commands) to distal outcomes (in this case, associative memory activity patterns), the second component mapping must be incorporated in order to process the results of this initial external function before the desired sequential mapping can me fully manifested.

This was a very challenging environment in which to test, particularly because of the variety and complexity of the components in the environment set to work in serial. To add to the complexity of this sequential composite environment mapping, there is no mechanism provided for

explicitly informing the learning agent as to its current state or plight. In other words, there is no other way for the learner to take into account where or how far its prior history of actions has taken it en route to accomplishing the distal sequential learning task than for it to remember what it had done. In acquiring the ability to generate time-varying proximal behavior given a single, unchanging input stimulus using the standard distal supervised learning framework, such a current state mechanism would be essential to ensure any measure of success.

As such, to train the distal recurrent learner to blindly produce any correct sequence of motor commands in so complex an environment without the benefit of receiving constant updates of its own current state would truly be an accomplishment. Updates to the recurrent distal learner on its new current state, separate from the environment's distal outcomes, assist the agent by giving it a reference point as to where the series of actions taken prior to that point in time has guided it. For instance, the visual location of the ball could be used as an indicator of the agent's current state in the basketball shooting example illustrated in section 1.1. The key issue of a problem domain such as this is that, unlike other attempts at distal supervised learning, information on the learner's current state is unavailable for reliable guidance and usage. As there is no such stream of current state information to be provided in this domain, most previous standard distal supervised learning models would be ill-equipped to work well operating in this environment. This is a result of the fact that standard distal systems rely so heavily on using their incoming current state information to guide them to their next step.

The phoneme sequence generation environment is broken into two separate components. On the one hand, there is a segment which maps the set of motor phonemes, which emanate from the primary motor cortex, into the set of auditory phonemes, which are received by the primary auditory cortex. This component will take the form of a smooth mapping procedure developed just for this application and described in greater depth in Appendix B. This smooth mapping procedure accepts as parameters two countable vector sets, the first being the *domain* set and the second being the *range* set. The domain set, $A = a_1, a_2, ..., a_v$, contains vectors of length m and is considered a subset of a much larger domain $\Re^A \equiv \Re^m$. Likewise, the range set, $B = b_1, b_2, ..., b_v$, with vectors

of length n, is considered a subset of range $\Re^B \equiv \Re^n$. Both sets A and B should contain the same number of vectors, v, for the purpose of assuming the existence of a finite mapping, f, where $f(a_i) = b_i, 1 \leq i \leq v$. Smooth mapping, as outlined in Appendix B makes it possible to construct a new mapping, f^*, such that a different input $a \in \Re^A$ but $a \notin A$ will have a corresponding value $f^*(a) \in \Re^B$ based on the proximity of a to members of set A.

Obviously, the actual real world mapping from motor phonemes of the primary motor cortex to heard auditory phonemes in the primary auditory cortex has little to do with this demonstration. Indeed, many factors go into this actual mapping, including interaction with air molecules, external noise, etc. which are either not completely understood or are too complex to model for the context of this work. I maintain that, solely for the purpose of this particular study, all that is needed is some continuous smooth mapping, f^*, which can map any vector in $\Re^A$ reasonably to some corresponding output in $\Re^B$ (i.e. $f^* : \Re^A \rightarrow \Re^B$) and which reasonably interpolates a finite mapping, f, for all vectors in A (i.e. $f^*(a_i) = f(a_i) = b_i, 1 \leq i \leq v$. If the environment function can display these properties, the recurrent forward model can effectively approximate it and help drive learning of the distal recurrent learner.

The other component of the environment in use here is the storage of auditory phonemes in associative memory. The storage mechanism here will be a SARDNET SOM capable of accepting sequences of phoneme inputs and, once trained, outputting a corresponding unique output map in the output lattice of the SOM. This is to represent the unique pattern of neural activations thought to occur in the associative memory once a human senses or recognizes a previously sensed input stimulus.

The purpose of the environment is to output the corresponding sequence of SOM output maps once presented with some sequence of motor phoneme inputs. In other words, the environment first accepts a sequence of motor phonemes representing the distal recurrent learner's stimulation of the primary motor cortex and then sends this sequence through the smooth mapping process to be mapped to a corresponding sequence of auditory phonemes signifying the appropriate stimulation of the primary auditory cortex. The resulting auditory primary sequence will then be accepted

by the SARDNET SOM representing associative memory and will ultimately yield some ideally unique series of neural activations used to represent stimulation by the primary auditory cortex in recognition of stored representation. Figure (5.5) demonstrates visually how this environment operates.

One primary issue encountered, which is accepted as standard in distal learning architectures as presented in Jordan [24], is that a properly trained forward model can guide a distal learner to converge to one, and only one, correct proximal action out of potentially many. If there is truly only one correct proximal set of actions to take in arriving at the desired distal outcome, or if merely arriving at the desired distal outcome by any means is sufficient, then there is no issue. However, in designing an architecture to simulate phoneme sequence generation similar to that demonstrated by the human brain, an analog of a very specific response of the primary motor cortex in the brain is sought of the distal learner which should correspond closely to what is documented in existing neuro-biological studies (i.e., motor responses demonstrating features listed in Singh [56]). In other words, unlike most other previous distal supervised learning studies, a very specific proximal response is required of the distal learner given a single input stimulus in order to yield a particular desired distal trajectory. As such, the environment to be used in this study must be carefully constructed to be as one-to-one in nature as possible, as opposed to the various many-to-one environment mappings used in previous distal learning studies. Consequently, the recurrent forward model designed to learn this particular one-to-one environment mapping can be trained with the purpose of guiding the distal recurrent learner to that specific proximal course of actions. The intent is to develop a neural model which learns a very specific one-input-to-many action mapping whose outputs can be verified as correct based on expected data possessed by the teacher.

5.3.5 Forward Model

Various properties of the proposed system seem to hold true across simulations and problem domains. One very important observation is that the proper training of the forward model is

paramount to success of this or any system like it. The motivation for even providing a forward model is to come up with a parameterized approximation of the unknown environment which could be manipulated in order to guide and assist in the training of the distal sequential agent. This can be done initially by taking random sequential walks through the environment's input space, mapping it using the environment to its corresponding distal sequential outputs, then using the resulting training pairs to train the forward model even before the training of the distal learner gets underway. This portion of training a distal learner is often referred to as babbling. It is named as such since it is analogous to that stage of seemingly random, but essential, stumbling through vocal sounds in a young infant's early language development.

An issue arises in looking to address where the sample input data should come from and in how much such data should be used for training of the forward model such that it could best assist in the training of the distal learner. Since the input space of many complex real-valued multi-variate domains is, for all intents and purposes, infinite in range, the desired environment mapping may never be fully characterized by the forward model.

One way to do this is to actually take, if available, the actually proximal sequential outputs which would ultimately generate the desired distal output sequences, pairing them with their respective target outputs, and including them in the training data for the forward model. The idea is that if the forward model trained in this manner knows directly how to map the correct, yet "unknown proximal answers", then it should be more capable of training the distal learner to arrive at these proximal answers. In other words, the forward model would be in a better position to provide correct error training signals to the distal learner if it understood the requisite mapping between answers and desired distal outputs.

In using the phoneme sequence generation environment, the most successful runs were conducted such that the desired proximal behavior was expressly used as babbling data in the initial training of the forward model along with their desired targets before beginning actual training of the distal learner. In other words, during this babbling phase, the recurrent forward model explicitly was trained using the desired distal sequential map representations as target output sequences

and the phoneme sequences that were responsible for generating them as their respective input training sequences. During the actual training of the distal recurrent learner, however, in addition to the initial babbling forward model training data, sequential outputs of the learner were provided to the forward model to be trained on along with their resulting sequential environment outputs. In this manner, the forward model could be trained simultaneously with the distal recurrent learner so that it could continue to learn to mimic the environment using the training data being generated naturally through the interaction of recurrent distal learner and the environment.

Surprisingly, even when they are available for training, expressly providing the expected proximal answers to the forward model, though helpful, often does not yield a distal learner which fully acquires the desired proximal behavior in its entirety in many complex distal sequence generation tasks.

5.3.6 Simulation of the Phoneme Sequence Generator

As a preliminary to any training, some architectural features must be selected for both the recurrent distal learner and the recurrent forward model. These can lead to important ramifications during the simulation. Some of the more important architecture choices are: 1) size of both hidden layers, 2) recurrent network type (Jordan or Elman), and 3) number of delay lines memory modules. Once this is done, the system's parameters can be initialized, including that of the random setting of the weight vectors for both neural models.

As previously discussed, the forward model goes through a babbling stage before training the distal recurrent learner to mimic the environment mapping. There are two types of training data used in this study for training the forward model during this phase. Randomly created data may be used here as well as the actual desired proximal sequential answers known to yield the distal target sequences, assuming they are available to the trainer which is often not the case.

In the case of randomized babbling, generated training data is constructed as 40 randomly generated sets of vector sequences. One half are vectors made up of real valued entries x_{ij} s.t. $0 < x_{ij} < 1$, while in the other half of the babbling random data, the vector sequences comprise

solely randomly generated vectors of 0s and 1s.

At this point, after babbling, training of the distal recurrent learner commences in the manner outlined in Section 3.3 in conjunction with the recurrent forward model. The recurrent forward model will continue to be trained to learn the sequential environment mapping using the output action sequences generated by the distal learner as inputs and their resulting distal outcomes as target output sequences. Note also that, in addition to these output action sequences, whatever data were used during the completed babbling stage to train the forward model are generally cached and re-used continually by the latter throughout training of the distal recurrent learner in addition to these output action sequences. This is because the forward model will tend to forget the mappings learned during babbling, making that practice futile. The training of the distal recurrent learner in the phoneme sequence generation environment is set to run, post babble stage, for 10,000 epochs or until the distal performance error (i.e., the RMSE between actual and target distal sequences occurring in the environment) becomes lower than .05. The training procedure referred to here is just as outlined in Section 2.5.

5.3.7 Simulation Results

Four sets of numerous simulations each were run using the phoneme sequence generation input / output data and environment. In each set of experiments, 11*11 = 121 training sessions were run, where every combination of even numbers between 40 and 60 were used as hidden layer sizes for both the recurrent distal learner and recurrent forward models (both designed as Jordan networks). What varied primarily across experiments was which of the two recurrent networks (1-2. either, 3. both, or 4. neither) were set to do teacher forcing. Recall that in teacher forcing the precisely or approximately correct target outputs, as opposed to the initially erroneous outputs of the untrained neural network itself, were inserted into the memory layers in attempting to encourage quicker learning of the training data.

Out of the total number of runs done for this study, only the top 5 runs of each set of simulations were examined and their learning curves matched up and examined. The training of each type was

recorded (in steps of 20 epochs from epoch 0 through epoch 10000) and averaged over all the 5 best runs of each type to yield an average learning performance curve to represent the efficiency of that type of architecture.

In each of the charts shown in Figure 5.6, the performance chart of the runs where absolutely no teacher forcing (approximated nor standard) was utilized was plotted against each of the other three types that utilized a teacher forcing strategy throughout training for either or both recurrent distal learner or recurrent forward model. Across each of the three graphs, the darker line represents the same averaged training curve tracking distal error of recurrent distal learners trained without use of any teacher forcing strategy over a number of runs. Here, one can readily compare the averaged run of the no-teacher-forcing architecture against the averaged runs which utilized teacher forcing in a) recurrent distal learner only, b) both recurrent distal learner and forward model, and c) recurrent forward model only. Note that the models which utilized approximated teacher forcing in the recurrent distal learner clearly demonstrate a better capacity to learn up until a point, then diverge inexplicably late in the run.

The charts of Figure 5.7 are similar to the charts shown in Figure 5.6 except to track proximal error, averaged runs for non-teacher forced architectures are plotted against those for architectures which employed some teacher forcing strategy in a) the recurrent distal learner only, b) both recurrent distal learner and forward model, and c) recurrent forward model only. The proximal error is generally not trackable as it is here as the desired proximal sequential behavior is typically unavailable to the trainer. Only due to the nature of this problem, where the trainer merely wants to produce sequential behavior already known to the former, can we actually calculate RMSE performance over the course of a run.

What seems to occur consistently in these graphs is that any simulations which utilize the approximated teacher forcing in the distal recurrent learner seem initially to actually learn more quickly than those which do not employ that scheme. Unexpectedly, however, the graphs in both Figures 5.6 and 5.7 seem to suggest that standard teacher forcing done to the forward models, though it may lead to quicker training time in the initial babbling stage, may actually seem some-

what detrimental to the distal learning process. This is a truly unexpected result, and any explanation of this phenomenon would require further study.

It becomes apparent that, at least in this particular task, although using neither teacher forcing strategy tends to cause the distal recurrent learner to acquire the correct proximal sequential behavior in the slowest time, it does avoid the pitfall of diverging from the correct behavior once it is learned. Even though both sets of simulations that utilize approximated teacher forcing of the distal recurrent learner do indeed learn quicker for time (up to, on average, a point between 3000 and 4000), something occurs in which the distal performance error no longer converges. This very peculiar behavior suggests that the recurrent forward model fails to supply the correct proximal error late in runs, somehow only after the desired proximal behavior is acquired. This peculiarity can very well lie in the complex phoneme sequence generation environment, as no such behavior attributable to teacher forcing was detected in the preliminary distal concatenation sequence generation studies.

The six best performances with performance errors less than 0.06 were recorded in Table 5.4. Despite the issue with the divergence of the error curves of most simulations which include teacher forcing strategies, the best two performances, and also four of the best six performances, included architectures which used some form of teacher forcing. This observation, plus the fact that they tended to converge to those error rates quicker than those that used no such teacher forcing feature, suggests that, with work, these strategies can be indeed useful in training distal sequence generating architectures which employ Jordan recurrent neural networks.

Also listed with their best performance error are different accuracy rates of the distal learner in reproducing correct motor phoneme sequences. The first metric looks at the percentage of phoneme sequences provided by the trained distal recurrent learner that are entirely correct. In other words, suppose the recurrent distal learner outputs some motor phoneme sequence for each of the fifteen phoneme sequence intent stimuli presented to it. The percentage of these fifteen phoneme sequences which turn out to be sufficiently *similar* to the phoneme producing behavior we hope to see can be readily calculated. A vector x is considered *similar* to a vector y, where

Teacher Forcing		Number of Hidden Layer Elements		Distal Perf Error	% correct Phoneme Sequences	% correct Individual Phonemes	% correct Best Matched Phonemes
Distal Leaner	Forward Model	Distal Learner	Forward Model				
√		56	42	.053	66.7% (10)	84%	96%
√		44	52	.055	46.7% (7)	82%	94%
		58	56	.056	66.7% (10)	88%	94%
	√	46	50	.058	46.7% (7)	82%	94%
		46	54	.06	33.3% (5)	74%	96%
√		60	52	.06	53.3% (8)	82%	94%

Table 5.4: A listing of the best performing distal phoneme sequence generators indicating important architectural characteristics. These are the best of hundreds of randomly initialized runs which varied over such key characteristics as hidden layer sizes (between 40 and 60) and teacher forcing focus in both distal recurrent learner and recurrent forward models. Note that teacher forcing techniques were employed in four of the six best performing distal recurrent learners.

$x, y \in \Re^n, if |x_i - y_i| < C, 0 \leq i \leq n$, such that C is generally a real-valued constant set close to 0. For this study, C is set equal to 0.3.

Another metric measures how many phonemes generated were sufficiently similar to the respective sought after motor phonemes (i.e. how many phonemes were generated correctly.) For the last metric, each phoneme generated by the distal recurrent learner is compared to the set of five possible phonemes and replaced by the closest one. Once all phonemes generated are transformed in this manner, similar to the second metric, the percentage of all newly transformed phonemes which equate correctly with their respective desired motor phoneme counterparts is calculated and reported.

As an example, Figure 5.8 demonstrates the typical progression of a recurrent distal learner as it acquires the phoneme sequence generation behavior. In the beginning, the forward model goes through its babbling stage of learning to mimic the environment mapping before being utilized in the training of the recurrent distal learner. The recurrent forward model is trained on the phoneme sequence behavior known to ultimately evoke the desired series of sequential associative maps (Figure 5.8 a).) Once babbling is concluded, training of the recurrent distal learner, as outlined in Section 3.3 commences, while still proceeding to train, or calibrate, the forward model using the interaction between distal learner and environment as training data (Figure 5.8 b).) Figure 5.9 then shows the entire training run as it culminates after 10,000 epochs. Of interest is how it is apparent that, even when experiencing problems in the middle of the training run, the recurrent distal learner is still capable of correcting its own acquisition of correct proximal sequential behavior through interaction with environment and recurrent forward model exclusively.

5.3.8 Evaluating the Efficiency of Recurrent Distal Elman Networks

In much the same fashion that Jordan recurrent neural networks can be trained in using the recurrent neural network modification to the distal supervised learning framework, Elman networks, as discussed in Section 2.2, can be trained as well. In designing the recurrent distal learner, the recurrent forward model, or both to be Elman networks, the primary difference in the handling of the two recurrent architecture types is the source from which information is provided and stored to the respective memory layer. One issue which arises is the fact that teacher forcing strategies cannot be used for Elman networks, as activations from intermediate nodes cannot be predicted or known a priori.

As there remains some debate as to which recurrent network structure, Jordan or Elman, works best in standard, non-distal sequential learning tasks, I attempt to determine, if possible, which mixture of the two in this recurrent distal learning framework would lend itself to the creation of better distal recurrent learners. Would a Jordan distal recurrent network paired with an Elman forward model fare better than one which utilizes both Jordan distal and forward neural networks?

Experiment Label (Short form)	Distal Network Type /	Forward Model Network Type /
ee	Elman	Elman
ee (no decay)	Elman	Elman
ej	Elman	Jordan
ejt	Elman	Jordan (*)
je	Jordan	Elman
jte	Jordan (*)	Elman
jj	Jordan	Jordan
jjt	Jordan	Jordan (*)
jtj	Jordan (*)	Jordan
jtjt	Jordan (*)	Jordan (*)

(*) - Teacher Forcing

Table 5.5: List of Elman and Jordan Distal Architecture Simulations

How would the system fare if both distal and forward models were created as Elman Networks? Is there any benefit to using teacher forcing techniques to the Jordan portion(s) of any of these Jordan / Elman hybrid recurrent distal learning architectures?

A group of six new experiments of the phoneme sequence generation distal learning task, each of which included an Elman network as either the recurrent distal learner, recurrent forward model, or both, was run in order to test questions such as these. Each run comprised 121 varying length hidden layer sizes. Table 5.5 lists each of the different combinations of new Jordan/ Elman runs network uses in the recurrent distal supervised learning framework while listing their acronym or experimentation shorthand name as well. In Figure 5.10, the graph plots performances over the best five aforementioned Jordan experiments, with and without teacher forcing, as they compare to similarly trained simulations in which Elman networks were incorporated into one or both dis-

tal recurrent learner and recurrent forward model roles. The graph clearly demonstrates, oddly enough, that architectures which utilize Elman networks as either distal recurrent learner or recurrent forward model are consistently outperformed when compared with simulations which utilize two Jordan networks, whether teacher forcing is used or not. The reason for this huge disparity is not known currently. Future experimentation of this subject matter may indeed shed some light as to why there is such a clear advantage to using Jordan networks in a system such as this.

5.3.9 Implementing Delay Line Memory Constructs

In order to increase the effectiveness of the proposed memory modules added to the existing distal supervised learning architecture, the capability to directly copy and store individual proximal actions from previous time steps was incorporated into both distal recurrent learner and recurrent forward model. I determined that, rather than replacing exponential memories used effectively until now, I could add exponential decay functionality to the very last delay line memory. In this manner, the neural network being used, whether distal learner or forward model, could still consider long histories of action even when the extent of the delay line modules has been surpassed. Figure 3.3 shows a Jordan recurrent distal learner with an arbitrary number of these delay line modules, the last of which was, optionally, set up to use an exponential decay term in order to accumulate arbitrarily long output histories. With the increased faculty to clearly discern the d-1 prior actions taken in addition to the accumulation of exponentially decaying outputs at the final module, it was thought that adding this feature could noticeably improve the performance of the distal recurrent learner. Do note that the recurrent forward model utilized in Figure 3.3 does not utilize delay line memory modules. Memory delay line structures can be utilized for either, both, or neither recurrent distal learner and recurrent forward model.

5.4 Contributions of the Chapter

The primary contribution of this chapter is to demonstrate the capabilities of the recurrent distal supervised learning system in a challenging domain which employs a relatively complex environment. The Phoneme Sequence Generation environment was constructed by pairing the smooth mapping procedure (Appendix B) used to facilitate the transformation of spoken motor feature phonemes to heard auditory feature phonemes with the candidate-driven SARDNET SOM (Section 4.3) used to represent associative memory. The recurrent distal learning framework was shown capable of training a recurrent neural network, due to its cooperation with its accompanying recurrent forward model, to generate very accurate motor phoneme sequences that produced very specific desired distal output behavior in the environment. This learning occurred even when the recurrent distal learner was being presented only with a single static "intent" as input while operating in this complex sequential environment. Also, approximated teacher forcing (Section 3.4) was shown to have a very positive effect in the training of the recurrent distal learner as expected, particularly in the beginning stages of its learning.

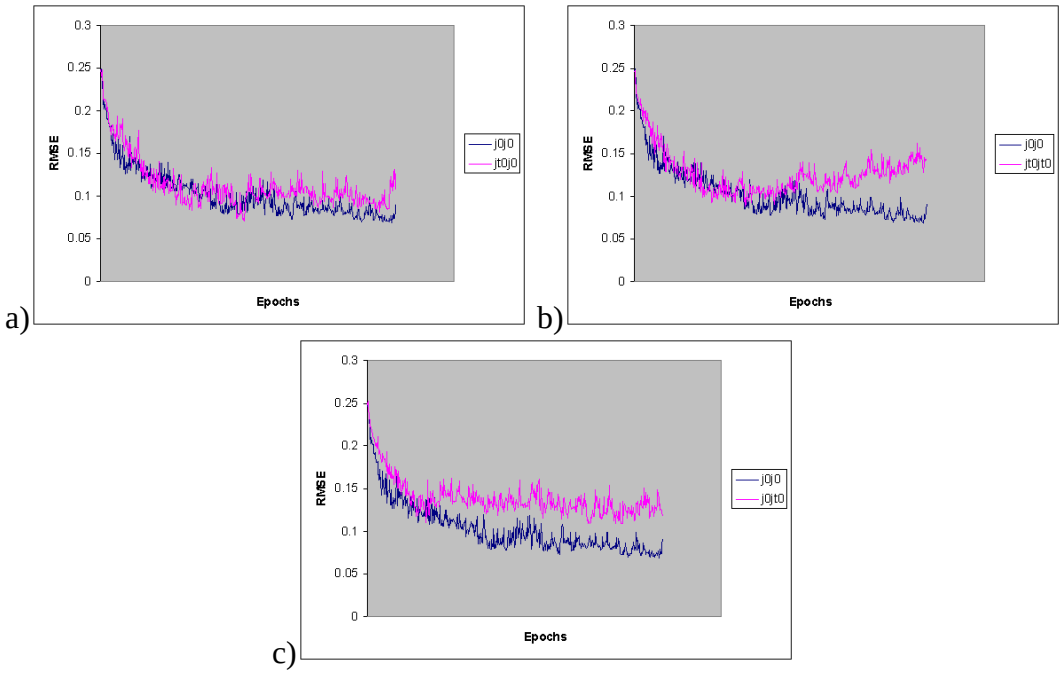


Figure 5.6: In plotting diminishing error (RMSE) against training time (epochs) over averaged runs, the effects of three separate uses of teacher forcing techniques are shown. In plot a), the averaged training run for teacher forcing used in the recurrent distal learner only (the training curve labeled jt0j0) is superimposed against a curve that signifies training of the recurrent distal learner without any form of teacher forcing (j0j0.) The remaining two plots demonstrate teacher forcing in b) both recurrent distal learner and forward model (jt0jt0), and c) recurrent forward model only (j0jt0) against the same non-teacher forced averaged training run. In all three graphs it can be seen that the teacher forcing methods demonstrate comparable, if not faster, learning in the onset of learning. Interestingly enough, though the lowest averaged learning rates can be seen in training curves in which teacher forcing strategies are utilized, divergence in learning can be seen in these same teacher forcing runs during the early to middle stages of their training.

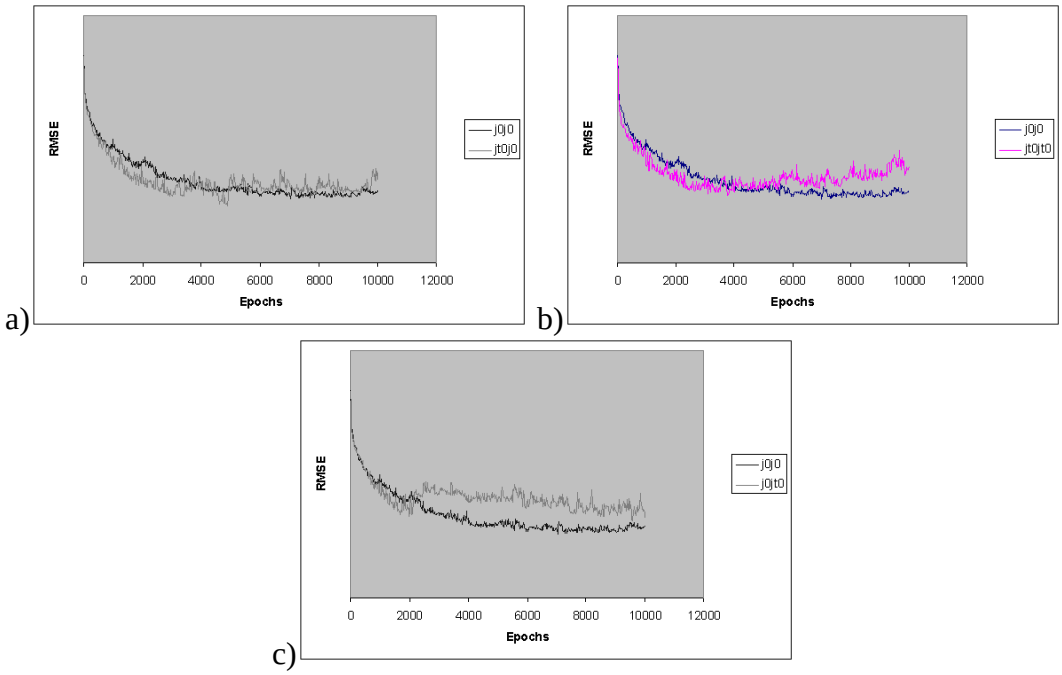
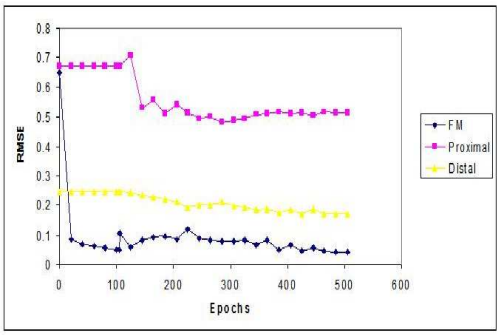


Figure 5.7: Similar charts to those shown in the charts featured in Figure 5.6 tracking the effects of teacher forcing except the proximal error of the recurrent distal learner's outputs are plotted as opposed to the distal error in the environment. Once again, the use of teacher forcing against the standard non-teacher-forced case (j0j0) is demonstrated here in a) recurrent distal learner only (jt0j0), b) both recurrent distal learner and forward model (jt0jt0), and c) recurrent forward model (j0jt0) only. A more profound positive influence is evident here early in runs as a direct result of the use of teacher forcing than when distal error was tracked.

a)



b)

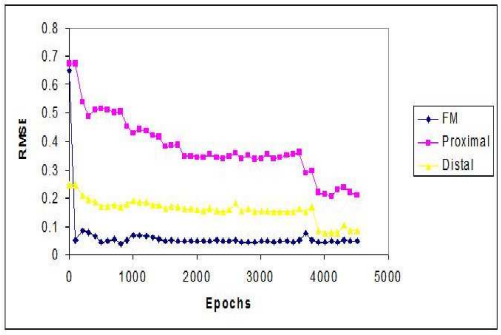


Figure 5.8: Two stages of the same training run are demonstrated for a well-trained phoneme sequence generator where diminishing error (RMSE) is tracked. In chart a) the initial babbling phase is evident in which the recurrent forward model (FM) alone is trained for 105 epochs, after which training commences for the recurrent distal learner (signified by diminishing error through epoch 505). In chart b), continued improvement in training the recurrent forward model is demonstrated by the sustained decrease of error through 4500+ epochs (including the sharp descent seen at just over epoch 3500.)

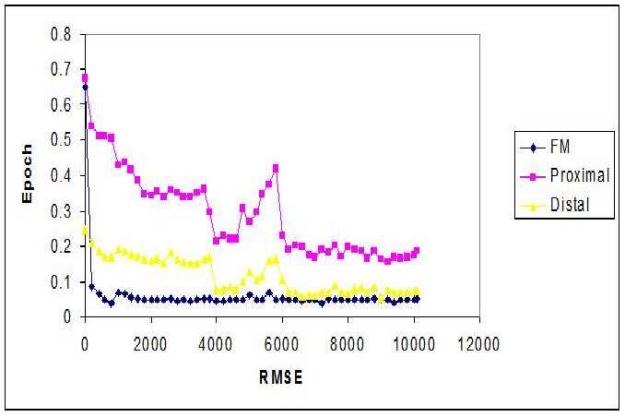


Figure 5.9: The final RMSE chart of the recurrent distal learner demonstrated in Figure 5.8 is shown here as it is trained for 10,000+ epochs.

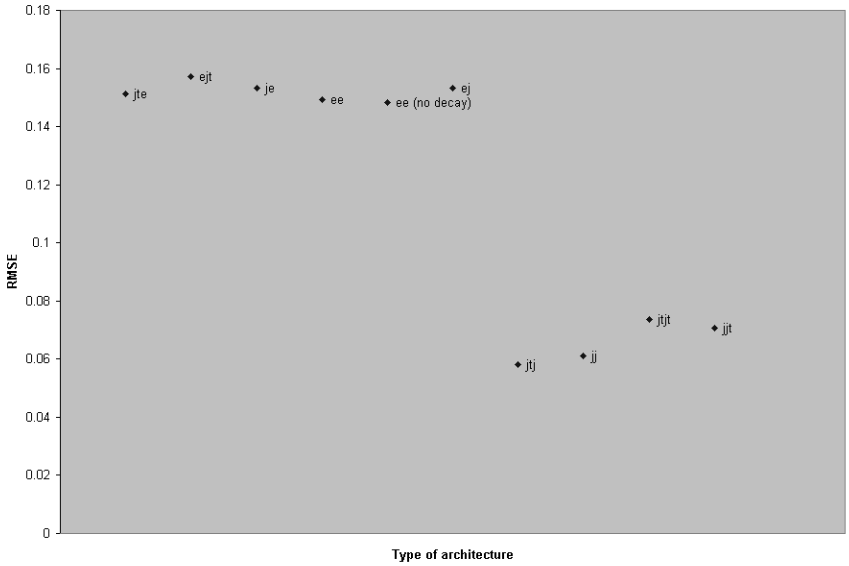


Figure 5.10: This graphic plots the performances of recurrent distal supervised architectures which utilized Elman recurrent distal learners and/or Elman forward models against performances of architectures only using Jordan architectures. The labels can be explained most efficiently by example. The point “ej” represents the mean performance of recurrent distal supervised architectures using an Elman distal learner and Jordan forward model. The point “jjt”, however, represents the mean performance of architectures using both a Jordan distal learner and a Jordan forward model (with the forward models alone employing a teacher forcing strategy to enhance its learning task.) Clearly, any architecture that utilized an Elman recurrent learner was significantly outperformed by any similar architecture that used solely two Jordan recurrent neural networks.

Chapter 6

Discussion

In this work, I demonstrate a modification of the existing distal supervised learning framework for training a recurrent neural network to produce sequences of varying length outputs which, when accepted by some sequential environment, yields the desired sequence of outcomes associated with the single static input stimulus presented to it. Moreover, it is shown that the same approximated proximal error vector supplied by the forward model to introduce effective weight vector updates in the distal learner can, in turn, be used to induce more effective updates of the recurrent distal learner's memory vector and, thereby, further improve training. This work is indeed significant in that now recurrent distal learners capable of considering its history of previous actions can be trained in environments in which the learner's current state is inaccessible. In fact, the results of these modifications are particularly distinct from those of other distal supervised learning techniques in that they allow for the effective creation of recurrent distal neural networks that are far less dependent on current state information than those distal learners trained using standard distal learning methods which tend to be heavily reliant on that information in satisfactorily making future decisions. The efficiency of the modified distal learning framework is demonstrated first on a simpler sequential concatenation environment, then later on a very ambitious phoneme sequence generation environment in which the recurrent distal learner seeks to acquire the ability to pronounce words in a similar manner as humans do. The following chapter discusses further the significance of the findings of this work as well as possible future directions for improving and extending this research.

6.1 Benefits of the Distal Sequence Generation Study

The role of neural networks with recurrent structures is becoming increasingly apparent. There are those, including Ziemke [73], who argue that there exist problems in robotic tasks in which a given state may be attained using several different action paths (e.g., the state arrived at may appear the same even though the path taken to achieve it was very different.) Learning tasks such as these can potentially lead to very difficult problems in which the current state is not sufficient to uniquely determine what the next agent action should be. Termed ”perceptual aliasing” by Whitehead and Ballard ([65]), such issues may be addressed by including mechanisms commonly used in sequential processing neural network simulations which expressly utilize past experience to more efficiently promote correct future decision making. This is but one of many potential applications which demonstrate the necessity for continued research into recurrency in neural models in all areas addressed by feed-forward networks.

Currently, there is no known work which addresses the use of recurrent neural networks in distal problem domains. However, the simulations run in Section 5.3.7 demonstrate that recurrent neural networks can indeed play a key role in creating neural models capable of learning to produce appropriate proximal sequential behavior to ultimately yield a series of desired distal outcomes while operating in a complex environment. Moreover, the fact that the distal recurrent learner does all of this while receiving no external updates of its own current state from the environment makes the task that much more intriguing.

While recurrent neural networks have been shown to be effective in managing distal sequence generation tasks, employing them to handle challenging non-sequential distal learning problems may prove to be extremely fruitful as well. Incorporating prior action history into the decision-making process by the employment of recurrent links and various memory module constructs may indeed enhance the training of standard distal feedforward neural network architectures in non-sequence generation tasks. It might even be possible to demonstrate improved training performance over standard non-recurrent distal learning systems that rely heavily on a consistent

source of current state information but utilize no concept of memory. This could potentially be the case if the current state information supplied to non-recurrent distal learners can be shown to be inaccurate, noisy, or ambiguous. More experiments would be required to determine under which circumstances the more memory-reliant recurrent distal learning systems might definitively be able to outperform standard, non-recurrent distal learning systems that rely exclusively on current state information.

6.2 Success in Recurrent Distal Supervised Learning

The architecture introduced here was demonstrated to work well in two sequential environments: 1) concatenation and 2) phoneme sequence acquisition and generation, the second of which is an exceptionally complex composite of two non-linear functions. The system was shown to work very well in the concatenation problem, which featured a less complex environment which boasted no ambiguity issues among environment outputs. The phoneme sequence generation architecture however, proved to be a much more challenging system to master. Ultimately, spanning a range of numerous simulations, when given 15 actual English phoneme sequences to acquire, the distal recurrent learner was able to produce at least 10 phoneme sequences correctly (Section 5.4).

Once again, it may be possible to incorporate the recurrent structures used in this study into existing distal supervised learning systems. Judging from the successful results seen in the distal recurrent learner training tasks of Sections 3.6 and 5.3, it is my belief that recurrent distal learners should be able to perform at least as well when substituted for feed-forward neural networks in standard, non-sequential distal learning systems developed over the years. In cases where current state updates can be ambiguous, for example, being equipped with knowledge of previous action history may be sufficient for a distal learning agent to break ties and determine what the best subsequent action should be.

As of this study, I illustrate a distal learning architecture I devised that can begin to handle distal sequence generation tasks acquired through interaction in an environment devoid of current state

information streams. Previously, all problems distal in nature required an agent which accepted some form of current state information it could use to drive its selection of a subsequent action. This reliance on ”seeing” at all times can be quite limiting and a hindrance. If the all-important state information should become noisy, inaccurate, or cease, the effectiveness of any system relying on it is significantly compromised.

There are agent situations and problem domains in which, once supplied with a single input stimulus or command, a correct sequence of actions is merely required to be executed in its external working environment. Previously, this type of problem was scarcely addressed. Distal recurrent supervised learning systems can now be constructed to ”blindly” adapt and learn to operate in external environments without receiving any information about their current state. Rather, as is typical of recurrent neural network applications, the use of self-loops and various memory structures can allow the acting agent to ”remember” arbitrarily long histories of its own proximal commands and act accordingly to accomplish the given task (section 2.2).

Of key significance is the existence of adaptive learning problems in which a given state would require different actions depending on what the agent had done leading up to that point in time. For instance, for the phoneme sequence generation task, suppose an agent intends to say ”baby” (pronounced b/ae/b/ee) and the current state information provided to it is merely the fact that ’b’ was the last phoneme uttered. The dilemma posed to the learning agent now becomes which phoneme should it utter next: the ’ae’ or the ’ee’? It was necessary in that instance for the agent to know the series of phonemes uttered up to that point before it could make an informed subsequent decision even when provided current state updates. This is termed ”perceptual aliasing” (Whitehead [65]) and there are numerous complex robot domains in which this type of phenomenon must be handled. Distal supervised learning systems up to this point have largely done little or nothing to address this type of problem. Instead, most instantiations of distal supervised learning systems tend to be content with solely using its view of the world at a given time, to decide on its subsequent action. This is not to say relying on current state inputs is a bad idea. Rather, it is the case that relying solely on current state updates can ultimately limit the capabilities of a learning agent.

By using recurrent neural networks in distal supervised problems, not only sequence generation problems can be addressed, but also systems which can benefit from having some notion of ”history” in completing their purpose. Though the system described here was shown not to need next state information in determining subsequent actions, it is not the case that it cannot utilize current state updates when they are effective. In fact, further work may reveal that the use of current state updates as employed in existing non-recurrent distal supervised learning systems, coupled with the memory structures addressed in the current work, may potentially bring about even more robust, fault-tolerant distal learners that consider where they have been in addition to where they currently are in deciding on their next move. The use of memories and histories in the determination of subsequent action is a valid step forward in the design of adaptive agents that are capable of avoiding the pitfalls of perpetual aliasing issues while learning to operate in complex environments.

Incorporating delay line structures in distal recurrent networks, just as discussed in Section 5.3.9, can be a powerful tool for generating sequences in environments. This idea of incorporating delay line structures could hold credence since it enables the recurrent learner and/or forward model to clearly discern the first few actions taken and utilize that information in order to yield the subsequent outputs or actions. In contrast, a standard recurrent neural network will tend to lose information over time when using an exponential trace memory as it continually applies the decay term to prior memory layer node activations. Further work in this area would be required to determine just how much delay line memory structures can improve upon the current recurrent distal learning architecture.

By what was just described, one might think that if one delay line memory structure can often bring about improved performance, adding arbitrarily more delay lines should continually bring about additional improvements.

Another observation of interest is that the Jordan networks, particularly those employing teacher forcing techniques, tended to outperform the Elman networks as learners and forward models in the phoneme sequence generation study. This was somewhat unexpected since it was often the case that Elman networks would converge more readily to the desired levels of performance in standard

non-distal sequential training problems than Jordan networks. Somehow, that did not translate to distal sequence generation problem domains. Again, it is unclear why this might be the case. If anything, it was believed that the Elman forward model could more capably mimic the environment than the Jordan model and be able to utilize its reuse of its own internal state representations via its hidden layer to most effectively assist in training the recurrent distal learner. This, in fact, was not the case and, ultimately, Jordan network architectures using teacher forcing strategies in the distal sequence generation domain prevailed (section 5.10.)

6.3 Issues with Training

6.3.1 Difficult Environment

Issues concerning the phoneme sequence model varied greatly. There the biggest, most significant issue was probably the challenges presented by the very ambitious and very ambiguous phoneme sequence generation environment. More study may be required in order to make such an already complex composite function of non-linear components less ambiguous for the study. As a result of the ambiguity that remained in the sequential environment, it seemed particularly challenging for the recurrent forward model to be able to guide the recurrent distal learner to produce the desired sequential proximal behavior (namely the actual motor phoneme sequences responsible for producing the target output associative maps used for training.) As such, it became very difficult to get entire motor phoneme sequences to come out as hoped. Often in distal supervised learning studies, very little is done to track the error of the proximal answers or actions of the distal learner. Indeed, **distal** error is tracked, and often used to drive training. If proximal error were to be tracked, however, it would imply that the proximal answers were indeed determinable by the trainer, and that would obviate the need for designing a distal supervised system in which desired proximal behavior is inaccessible. Success in distal supervised learning tasks is generally not measurable by error to some expected proximal behavior but by error to some desired set of distal outcomes in the environment. Even though many times in the phoneme sequence generation task the learner

would be shown to have been trained down to a RMSE performance less than 0.1, some of the motor phoneme sequences we would hope would yield this targeted distal behavior would not be the proximal sequences sought after. Rather, the resulting proximal sequential behavior exhibited by the recurrent distal learner would, due to inherent ambiguity issues, potentially be a completely different action sequence still capable of yielding distal sequential behavior very close to that targeted distal behavior.

It was largely due to the phoneme sequence generation environment in its complexity and ambiguity that the precise desired proximal sequential behavior was not always achieved. More specifically, the nature of the final map representations given by the SARDNET SOM representing associative memory served to cause the most significant challenges. Because the SARDNET SOM maps are primarily sparse, any SARDNET output maps resulting from actions of the recurrent distal learner will only show a difference in distal output from the result of its first action or phoneme by several bits at most.

The sparsity of the environment output certainly played a major role in the manner in which the recurrent distal learner could be trained, since training in this manner is driven by distal performance error. To further improve on the performance shown here, the sparsity of the SOM outputs could be kept as minimal as possible. One way to do this would be to keep the SOM output lattice dimensions to a minimum, hence reducing the number of 0 outputs as much as possible. Through trial and simulation, a SARDNET map with a 4x4 output lattice did pretty well to store the representations of 15 phoneme sequences (Table 5.3) consisting of an alphabet of the five auditory phonemes listed in Table 5.2. A 3x3 SARDNET SOM lattice could potentially suffice, particularly if repetitions of phonemes in the desired phoneme sequences stored in the maps were kept to a minimum or eliminated entirely.

Another way sparseness issues could be diminished in creating these distal output maps may involve using the Mexican hat multi-output feature covered in Section 4.3.1. This feature would allow ALL node outputs to fire, substantially limiting the number of non-firing SOM lattice nodes. Given that this Mexican hat output feature may very well be more neuro-biologically plausible in

attempting to simulate cognitive function, it may be worthwhile to see how well the distal recurrent supervised learning system would fare in using these types of outputs.

It is still quite difficult to train such a system correctly. It is a fact that there are very many methods one can use to attempt to train the system properly. Apparently, if the environment does not lend itself to easy or straightforward solutions, it can be very tough to obtain proximal correct sequential behavior on the part of recurrent distal learner. If the environment is privy to arriving at similar environmental outcomes from multiple differing proximal action trajectories, (that is, if more than one proximal set of actions can yield the same environment distal output), and if a very specific proximal answer is being sought , as in the phoneme sequence generation task, then there may be difficulty in finding the true answer.

Also, it can be quite a challenge to generate sequential environment data randomly to appropriately train the recurrent forward model in effectively sampling the environment space so that it can accurately learn to mimic it during its babbling stage and throughout the extent of the simulation run. A method for finding a good way to generate good ”random” yet directed training data which could effectively train the forward model to best enable it to assist in training the recurrent distal learner will be effectively investigated further.

6.3.2 Issues with Initial Random Setting of Neural Network Weight Vectors

Another factor which potentially restricted the effectiveness of the phoneme sequence generation model had to do with the randomness of the model. There seems to be a dependence in the manner in which the parameters are initially and randomly set once the experiments begin. If one were to run the system 10 times with the same makeup, architecture, etc. using 10 differing random seeds, the resulting behavior among them can vary greatly. The initial setting of random weights of the recurrent forward model and distal learner neural networks have much to do with how successful such a model can become. Methods can be investigated in order to determine more ways to make this issue more of a non-issue. The randomness issue is likely one that is present in many standard distal supervised learning systems and is not specific to just the augmented systems examined in

this work

6.3.3 Drawbacks Faced in Dealing with Exponential Trace Memories

One drawback to using exponential trace memories as outlined here, is the concern for the length of output sequences capable of being learned by the system. Using exponential decay memories holds the benefit of maintaining arbitrarily long histories in a very compact vector representation. In theory, they can hold potentially infinite histories without end. However, once decay terms are applied to prior outputs, it becomes more and more difficult to discern how long ago an output was first activated. For instance, if an output was set to 1.0 at time $t \geq 1$, that output is copied to the same position in the trace memory at time o_{t+1} but with diminished intensity. Assuming an exponential memory decay of .5, in producing an arbitrarily long output sequence greater than five, the output at the same position is reduced from a 1.0 at time t to $o_{t+5} = (\frac{1}{2})^4 = 0.0625$. This can be quite difficult for an untrained neural network to differentiate from the subsequent output $o_{t+6} = (\frac{1}{2})^5 = 0.03125$. As such, it is foreseeable that any Jordan recurrent network utilizing an exponential trace memory module could potentially have a problem blindly generating subsequent actions past a certain point without help from current state updates. Utilizing a mixture of exponential trace and delay line memory structures can potentially offset this issue to an extent. Also, using larger output values that will not deteriorate quite as quickly as the standard output 1.0 does may assist some in this regard. This issue would need to be addressed seriously if this feature is to be fully utilized.

6.3.4 Forward Model

Forward Model Training Data

The appropriate training of the forward model is ultimately paramount to the effective training of the distal recurrent learner. One significant challenge seems to be how one can best train the forward model to be of maximum service to the recurrent distal learner. One way of doing this, if

available, is to train the forward model, not the distal recurrent learner, using the expected proximal answers and their corresponding desired target distal sequences as input / output pairs. Of course, this is rarely useful because the point of developing distal supervised learning systems is that the proximal answers are generally not known. In the phoneme sequence generation model described here, for example, the best performance was most often obtained when the forward model was trained to efficiently map the correct proximal motor phoneme sequences to their corresponding target distal output maps. Of course, these particular distal output maps would be one and the same as those provided at the start of training and used as target sequences to train the distal recurrent learner in the first place.

One can argue that using this strategy in this fashion is justified for this particular task since the purpose of the system described in Section 5.3 is not to find correct proximal behavior previously unknown to the trainer. Rather, the goal of the proposed system is to replicate as closely as possible the process of phoneme sequence generation studied extensively in neuro-biological study. In fact, one could argue that incorporating the proximal answers in the training of the recurrent forward model can be tantamount to the visual and aural guidance coaching by some coach (e.g., teacher, parent, etc.) in teaching the pronunciation of a word, or in swinging a bat, to a child, for instance.

Alternately, one can merely generate a sufficient number of randomly created actions in the output space of the learner to be supplied as training instances for a forward model. Once the proximal action sequences are randomly generated, they can be applied to the environment to yield their corresponding distal sequential outcomes. At this point, these pairs of sequential proximal actions and distal consequences can be used to train the forward model on the resulting set of training instances. Though the latter is the easiest manner of forward model preparation, there are no guarantees that the data generated could be good or promising enough to prepare the forward model to fully and effectively train the recurrent distal learner.

A phenomenon I observed while conducting these recurrent distal learning simulations is that the forward model should at least be able to generalize the mapping of the desired proximal solutions, whatever they may be, to their corresponding distal target outputs in order to be entirely

successful. In the case of complex environments such as the one employed here, generalization in this fashion can be highly unlikely. In such an environment, the forward model would probably have to see and learn to map every set of correct proximal actions in order to even hope to train the distal recurrent learner to learn to produce them. This could potentially be done through random generation of training instances and through subsequent interactions between distal recurrent learner and the environment. But to anticipate generating enough proximal sequences to enable the forward model to properly sample the sequential input space in such a complex, non-linear environment can likely be unrealistic and can require a tremendous amount of computing power, space, and simulation time.

In the absence of extensive computing resources, supplying the forward model with some amount of correct proximal behavior up front can give the forward model a better chance to further generalize to the environment mappings necessary for effective training of the distal recurrent learner. In trying to do the phoneme sequence generation task with both types of data (i.e., both randomly constructed and also the known proximal answers to the problem) it became apparent that the simulations which employed forward models trained with known proximal answers tended to lead their corresponding distal recurrent learners to converge at a greater rate than those which utilized randomly generated data to train the forward model. Recall that in distal recurrent supervised learning experiments proximal actions need to be generated and supplied to the forward model for training purposes during babbling and training stages. Another factor that is directly manipulatable by the trainer is the size of the recurrent forward model's hidden layer. A forward model whose hidden layer is too small can be ill-equipped to sufficiently partition and, subsequently, be able to propagate effective error signals in training the recurrent distal neural network. More research can be done to determine what types of data can be best used to train the forward models of complex environment functions effectively without the use of known proximal answers.

However, although recurrent forward models tend to work better once trained on the proximal answers, such a strategy is not at all sufficient to create forward models which can effectively guide any given recurrent distal learner to learn the correct proximal behavior every time. Often in

simulations of the phoneme sequence generation task, even when trained on the correct proximal behavior down to a very low performance error, many forward models were incapable of propagating back effective error signals in the training of its corresponding recurrent distal learner. Part of the success of training a successful distal learning system apparently relies heavily on the initial random parameter settings of both the recurrent forward model and the recurrent distal learner.

Oddly enough, unlike in recurrent distal learners, incorporating delay line memory structures in forward models has not demonstrated improved performance in the distal recurrent training task. Moreover, one would think that architectures with delay line memory constructs either in the forward model or in the distal recurrent learner would outperform those that employ neither. Rather, simulations that employed distal recurrent learners that contained at most one delay line memory structure and forward models with no delay line memory structure tended to do noticeably better than any other distal recurrent supervised learning system setup.

Furthermore, it is not necessarily the case that more memory delay lines in either recurrent forward model or distal learner implies better performance over fewer delay lines. Similar experiments demonstrated that use of two or more delay line memory structures in either or both forward model or distal learner did not necessarily improve learning. In fact, in many cases learning was shown to be hindered in comparison to systems utilizing only one delay line. This result can very likely be isolated to distal sequential problem domains using environments of this type or level of complexity. Still, further study can be done to determine the cause of this phenomenon.

In noting the importance of the forward model training data in the success of training the recurrent distal learner, certain methods were developed in an attempt to improve the forward model training as it looked to mimic/model the environment of the phoneme sequence production system. One such attempt included the caching of past babbled output sequences made by the recurrent distal learner and their corresponding distal outcomes to be used multiple times in training the recurrent forward model. The idea here was to see, in the absence of more training data, if the forward model could be made to learn the sequential environmental mapping better. Experimentally, it was determined that such a strategy was not convincingly effective, whether such data was held

or cached for two or more time periods or just one (the latter being standard practice in most distal supervised learning systems.) This was just one instance of the strategy which did not work.

Also, rather than update a forward model just once on a given set of babbled data, I thought that updating or training it on the new data more than once during the same epoch could potentially help it to train better. Such an action would allow the forward model to learn more precisely what the true mapping of every randomly created or recurrent distal learner generated proximal action sequence could be, further allowing it to approximate the environment mapping appropriately. In this case, it did not work out experimentally as well as expected. Why this did not work is as yet unknown.

Currently, what works is to keep the actions generated by the recurrent distal learner in forward model training for only one epoch and to delete it before the next forward model epoch or update can begin. It seems sufficient enough for the recurrent forward model to use a recurrent distal learner's attempt at generating a good sequential proximal response, given the static input stimulus presented to it, and its corresponding distal sequential outcome as training data in one step.

6.4 Future Work

6.4.1 Improving Performance of Recurrent Distal Supervised Learning Architecture

A good deal of success was demonstrated in observing the performance of this newly proposed distal supervised learning system which employs recurrent links in both the distal learner as well as the forward model while also utilizing cumulative memory layer strategies in either. However, some aspects of the newly proposed architecture can be investigated for further improvement of this new system. One such aspect of learning which can be investigated further is the effect of varying the number of hidden layers included in either or both recurrent distal learner and forward model. If more than two hidden layers are incorporated in either neural network component, activations

from up to all hidden layers can potentially be recorded and used in exponential trace memories. The new possibilities may grant either recurrent network increased computational capability to further partition the environment mapping into segments from which more informed decisions can be made in generating good subsequent actions. Further experiments in this direction may produce even better sequence generation performance than that found in the present study.

Another potential aspect of this work which could be investigated further is the effects of different output functions to the hidden layers (and possibly the output layers) to see if further improvement can be made in training distal sequence generation neural systems. Utilizing recurrent neural networks in which layers of nodes employ the tangent hyperbolic (Tanh) output function, in particular, may enable these networks to successfully converge at significantly higher rates than those networks which employ standard logistic output functions. There may be increased benefit in using Tanh output functions just because of the increased range of output possibilities that the affected nodes can perform. In essence, the Tanh(x) has a range of $-1 \leq Tanh(x) \leq 1$ while the standard logistic function (sig(x)) has a more limited range of $0 \leq sig(x) \leq 1$. A direct result of this change in output function is that weight vectors have a larger range of possible answers, which may be good or bad.

Another significant consequence of switching to an output function with a greater range is that with Elman and Jordan/ Elman hybrid recurrent architectures, their memory trace modules will now be made to handle negative activations. This may be even less the case with Jordan networks since eventually, at least in the phoneme sequence acquisition task as described in this text, each of their output units, and hence their memory contents, would all eventually be in the range of, or very near (0,1). This modification could, in fact, have a very significant effect on the training of the learner. Future simulations augmenting recurrent distal learners and recurrent forward models alike in this manner should show just how beneficial, or detrimental, such a change can result.

One issue to be addressed in the use of Tanh output nodes is its accuracy in depicting actual neuronal behavior in neural model simulations of brain behavior. It is known that neurons tend to either be inactive (0 output) or firing (1.0 output). This makes it easy to classify neurons as

semi-binary in nature. The problem is that nodes of a neural network utilizing the Tanh hyperbolic function output can potentially output negative numbers. To my knowledge, there is no concept of negative activations emanating from neurons in the brain, just negative connections. For problem domains such as these it would seem that sticking to output node functions which produce outputs in the range (0, 1) would be most beneficial.

It is certainly the case that neurons are known to inhibit as well as excite other neighboring nodes once activated. But inhibition in neural networks is typically already addressed in the way the weights can be negative or positive. So positive activation of a node in a neural network can actually inhibit a neighboring neuron by virtue of the weight connecting the two being negative in value. By having neurons which can produce both positive and negative outputs, you can no longer express a relation that one neuron will always inhibit a particular neighbor. That is, unless connecting weights are somehow restricted to positive values, which could indeed defeat the purpose of switching to Tanh output nodes. If it is indeed the case that inhibition/excitation relations exist between neurons in the brain, such relations would potentially be nullified in corresponding neural simulations if tangent hyperbolic nodes were being used.

Yet another area of interest I could investigate would be that of the role of radial basis networks in improving the use of neural networks in distal problem domains, whether sequential or not. The update procedure would certainly change significantly as the formulation of outputs and weight updates between radial basis networks and standard feed-forward networks differ substantially. But if something were to come of this research, much could be gained in taking advantage of the radial basis nodes' ability to classify clusters. Currently, it is not clear how one would utilize the gradient of the radial basis forward model as one would the gradient of a feedforward forward model. It may be the case that only the distal learner or the forward model, and not both simultaneously, could be capable of being constructed from radial basis nodes.

6.4.2 Modeling Sequence Generating Cognitive Tasks

Another direction for future research is to progress into more advanced bi-hemispheric neural models of brain activity. In previous studies, we implemented a bi-hemispheric neural model, two feed-forward neural networks with hidden layers that contributed to each other's activation via a positively or negatively weighted pathway (Reggia, Gittens, et al. [46], [47].) The inclusion of this pathway was inspired by the corpus callosum known to connect the right and left hemispheres in the brain. The joined neural networks were capable of being trained in tandem to produce sequences of phoneme vectors in an effort to test potential factors which could attribute to the emergence of lateralization in the brain. In the study, experiments suggested that a number of factors can have a role in contributing to lateralization, including size of the hemisphere as well as plasticity and speed. From these same experiments, other observations from neurobiological studies could also be potentially inferred. For example, negative, or inhibitory, contribution on the part of the corpus callosum through which the hemispheres communicate showed evidence of mirrored activations between hemispheric hidden layers connected homotopically.

Additional lesioning studies were conducted in which activations of hidden layer neural units of either hemisphere of the bi-hemispheric model were deliberately turned off to simulate damage to the brain as a result of stroke or brain trauma, for instance. This series of experiments was designed in order to study not only factors contributing to functional lateralization in the brain but also factors which assist most in recovery of damage to the brain. It was found that the simulated corpus callosum assisted in having the non-damaged region of the hemisphere to adequately pick up function lost by the acute lesion in the damaged hemisphere. There was evidence of much of the phenomena seen in the actual studies of stroke damage in patients. For instance, for positive contributing corpus callosum, the corresponding area connected homotopically to the lesioned portion of the damaged hemisphere in the non-damaged hemisphere experienced reduced inhibition. Also, the neural contributions of neighboring neurons in the damaged model themselves lacked activation and their contribution lessened as well.

The drawback to such an initial study was the lack of feasibility of the architecture as one

to truly model the phoneme sequence, or any intelligent cognitive motor function, acquisition process. Despite the fact that behavior resembling actual neuro-biological phenomena was shown to be replicated in the test experiments, many aspects of the actual brain process evaded the original design. For one, the model used primarily a local representation of inputs and outputs. That is, inputs and outputs each specified a phoneme or phoneme sequence by a single neuron being on or off, which is unlikely. Secondly, there was little use of many processes known to have a role in the phoneme sequence generation process. There was, for instance, no existing interaction with the external environment, no distinction between motor and auditory features, and certainly no mention of stored representation of sequences in associative memory.

Once this model is completed, a more realistic, feasible, and complex bi-hemispheric model can be constructed, in which the following can be asserted:

1. Babbling can be construed as the training of the forward speech model from observations of random motor actions in the environment. This step is deemed necessary for training the forward speech model and can be introduced as a precursor and stepping stone to language acquisition.

2. Two hemispheric regions can accept as an input stimulus a distributed representation of phoneme sequence intent.

3. Each hemispheric region can have access to the forward speech model in acquiring the phoneme sequence acquisition skill following the initial babbling stage and through continued babbling during the actual distal sequential learning task. Forward models are widely believed to hold a significant role in acquiring language production skill in humans.

4. Interaction does indeed occur in an external environment that transforms motor phonemes to auditory phonemes and accesses unique activation maps of stored phoneme sequence representations in associative memory.

5. Both models of left and right hemispheres can again work in parallel and conjunctively

through use of the intermediary corpus callosum.

Another potential plan for future research would be to expand on the phoneme sequence acquisition model discussed in Chapter 5. With work, more phonemes, and hence more phoneme sequences, could be learned by the model. Also, a more biologically plausible self-organizing map such as the one-shot, multi-winner SOM (Shultz [54]) could be investigated to replace the efficient, yet implausible, SARDNET SOM which is used to represent associative memory in the model.

6.4.3 Incorporating the Self-Halting Mechanism into the Recurrent Distal Supervised Learning Architecture

Finally, it would be useful to re-visit the idea of incorporating the self-halting functionality in this recurrent distal supervised architecture. The self-halting feature proved to be very difficult to implement in an already tough phoneme sequence acquisition task. One feature which could be implemented at a later date, is the self-halting mechanism. Given time constraints, limited success was achieved in enabling the distal learner to acquire the ability to output a halting signal to stop itself from producing a sequence of arbitrary length rather than being told ahead of time how many actions to produce in a sequence. Initial success was seemingly hampered by the difficulty of having to learn to output a halting signal which was significantly different from other legal recurrent distal learner action vectors in addition to learning to operate in such a complex environment which proved to be too challenging a task at this early stage of the study.

BIBLIOGRAPHY

[1] Barto, A.G., *Reinforcement Learning*, Handbook of Brain theory and Neural Networks, MIT Press, Cambridge, MA, pp. 804-809.

[2] Barto, A.G., R.S. Sutton, and C.W. Anderson, Neuronlike elements that can solve difficult learning control problems, *IEEE Transactions on Systems, Man, and Cybernetics*, **13**:835-846.

[3] Blakemore, S.-J., Daniel Wolpert, and Chris Frith, Why can't you tickle yourself?, *NeuroReport: Review*, Lippincott Williams and Wilkins, August, 2000, pp. 11-16.

[4] Blakemore, S.-J., Chris D. Frith, and Daniel M. Wolpert, The cerebellum is involved in predicting the sensory consequences of action, *NeuroReport: Brain Imaging*, Lippincott Williams and Wilkins, July, 2001, pp. 1879-1884.

[5] Bloom, L., Language acquisition in its development context. *Handbook of child psychology: Vol. 2. Cognition, Perception, and Language.*, In D. Kuhn and R.S. Siegler (Eds.), New York: Wiley, 1998, 5th ed., pp. 309-370.

[6] Campolucci, P., A. Uncini, and F. Piazza, A unifying view of gradient calculations and learning for locally recurrent neural networks, 1997.

[7] Carlson, A.B., 1986. *Communication systems*, New York: McGraw-Hill.

[8] Chen, S., S. Billings, and P. Grant, Non-linear system identification using neural networks, *International Journal of Control*, 1990, **51**:1191-1214.

[9] D'Autrechy, C.L., and J. Reggia, 1989. An Overview of Sequence Processing by Connectionist Models, *University of Maryland Dept. of Computer Science Technical Report*, UMIACS-TR-89-82 (also CS-TR-2301), Institute for Advanced Computer Studies, Department of Neurology, UMAB and Department of Computer Science, University of Maryland, College Park, MD 20742.

[10] Dell, G., Spreading activation theory of retrieval in sentence production, *Psych. Reviews*, 1986, **93**:283-321

[11] Dennis, S., Behavior with an implicit teacher in connectionist networks.

[12] Donoghue, J., S. Leibovic, and J. Sanes, Organization of the forelimb area in squirrel monkey motor cortex., *Experimental Brain Research*, 1992, **89**:1-19.

[13] Elman, J.L., 1990. Finding structure in time, *Cognitive Science*, **14**:179-211.

[14] Fahlman, S.E., An Empirical Study of Learning Speed in Back-Propagation Networks., *Carnegie Mellon University Computer Science Dept. Technical Report*, CMU-CS-88-162 1988.

[15] Flanagan, J.R., and Alan M. Wing, The Role of Internal Models in Motion Planning and Control: Evidence from Grip Force Adjustments during Movements of Hand-Held Loads, *The Journal of Neuroscience, Society for Neuroscience*, **17**(4):1519-1528

[16] Georgopoulos, A., R. Kettner, and A. Schwartz, Primate motor cortex and free arm movements to visual targets in three-dimensional space. II, Coding of the directions of movement by a neural population., *J. Neuroscience*, 1988, **8**:2928-2937.

[17] Guenther, F.H., 1995. Speech sound acquisition, coarticulation, and rate effects in a neural network model of speech production, *Psychological Review*, **102**:594-621.

[18] Haykin, S., *Neural Networks: A Comprehensive Foundation*, Prentice Hall, Inc., 1999.

[19] Igel, C, and Michael Husken, Improving the Rprop Learning Algorithm, 2000, pp. 115–121.

[20] Jakobsen, R., G. Fant, and M. Halle, *Preliminaries to Speech Analysis: the Distinctive Features and their Correlates*, MIT Press, 1951.

[21] James, D.L., and Risto Miikkulainen, SARDNET: A Self Organizing Feature Map for Sequences, *Advances in Neural Processing Systems*, 1995.

[22] Joost, M., Wolfram Schiffmann, Speeding Up Backpropagation Algorithms by Using Cross-Entropy Combined with Pattern Normalization., *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 1998, **6(2)**: 117-126

[23] Jordan, M., Attractor dynamics and parallelism in a connectionist sequential machine, *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Englewood Cliffs, NJ: Erlbaum, pp. 531–546, 1986.

[24] Jordan, M., and D. Rumelhart, 1992. Forward models: Supervised learning with a distal teacher, *Cognitive Science*, **16**(3):307-354.

[25] Kaebling, L.P., M.L. Littman, and A.W. Moore, Reinforcement Learning: A Survey, *Journal of Artificial Intelligence Research*, 1996, *4*:237-285.

[26] Kohonen, T., Self-organizing formation of topologically correct features maps, *Biological Cybernetics*, 1982, *43*(1):59-69.

[27] Karniel, A., R. Meir, and G. Inbar, Polyhedral mixture of linear experts for many-to-one mapping inversion, 1998.

[28] Karniel, A., Three creatures named 'forward model', *Neural Networks*, Elsevier Science Ltd., 2002, **15**:305-307.

[29] Kawato, M., Internal models for motor control and trajectory planning, *Current Opinion in Neurobiology*, Elsevier Science Ltd., 1999, **9**:718-727.

[30] Levelt, W.J.M., Roelofs, A., and Meyer, A.S., A theory of lexical access in speech production, *Behavioral and Brain Sciences*, 1999, **22**, 1-38.

[31] Michie, D., and R.A. Chambers, BOXES: An experiment in adaptive control, *Machine Intelligence 2*, Oliver and Boyd, 1968, pp. 137-152.

[32] Merkl D., and A. Rauber. Alternative ways for cluster visualization in self-organizing maps. *In Proceedings of the Workshop on SelfOrganizing Maps, Espoo, Finland, June 4-6*, 1997, 106-111.

[33] Minsky, M., Steps toward artificial intelligence, *Proceedings of the Institute of Radio Engineers*, 1963, **49**:8-30.

[34] Minsky, M., Seymour Papert, *Perceptrons, an introduction to computational geometry* The MIT Press, 1969.

[35] Mozer, M., Neural net architectures for temporal sequence processing, Predicting the future and understanding the past, A. Weigend and N. Gershenfeld, Redwood City, CA: Addison-Wesley Publishing, 1993.

[36] McClelland, J. L.,and J.L. Elman, Interactive processes in speech perception: The TRACE model, Parallel distributed processing: Explorations in the microstructure of cognition. *Volume II: Psychological and biological models*, MIT Press, 1986, pp.58-121.

[37] Narendra, K.S., and K. Parthasarthy, Identification and control of dynamical systems using neural networks, *IEEE Transactions on Neural Networks*, 1990, **1**:4-27.

[38] Nikovski, D., Sridhar Ramakrishna, Rajani Kanth Koneru, Srinivas Jamhed, and et al., Distal supervised learning for solving inverse kinematic problems,

[39] Pearlmutter, B.A., Gradient calculations for dynamic recurrent neural networks: A survey, *IEEE Transactions on Neural Networks 6* (1995), no. 5, 1212–1228.

[40] Pineda, F.J., 1989. *Recurrent backpropagation and the dynamical approach to adaptive neural computation*, Neural Computation, **1**:161-172

[41] Poggio, T., and F. Girosi, Networks for approximation and learning, *Proceedings of the IEEE*, **78**(9):1481-1497, 1990.

[42] Puskorius, G.V., and L.A. Feldkamp, Neurocontrol of nonlinear dynamical systems with Kalman filter-trained recurrent networks, *IEEE Transactions on Neural Networks*, 1994, **5**:279-297.

[43] Puskorius, G.V., L.A. Feldkamp, and L.I. Davis Jr., Dynamic neural network methods applied to on-vehicle idle speed control, *Proceedings of the IEEE*, 1996, **84**:1407-1420.

[44] Radio, M., J. Reggia, and R.S. Berndt, Learning word pronunciations using a recurrent neural network, University of Maryland Dept. of Computer Science Technical Report, Department of Computer Science, University of Maryland, College Park, MD 20742, 2001.

[45] Reggia, J., C. D'Autrechy, G. Sutton, and M. Weinrich, A competitive distribution theory of neo-cortical dynamics, *Neural Computation*, 1992, 4:287-317.

[46] Reggia, J., S. Gittens, S. Goodall, and Y. Shkuro, Lateralization and lesioning of a two hemisphere model of single-word reading, *Proc. Second Intl. Workshop on Neural Models of Brain and Cognitive Disorders* 1998.

[47] Reggia, J., S. Gittens, and J. Chhabra, Post-lesion lateralization shifts in a computational model of single-word reading, *Laterality*, 1999.

[48] Reggia, J., S. Goodall, and S. Levitan, Cortical map asymmetries in the context of transcallosal excitatory influences, *NeuroReport*, 2001, 13(8):1609-14.

[49] Riedmiller, M., and Heinrich Braun, A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm, Institut fur Logik, Komplexitat und Deduktionssyteme, University of Karlsruhe, W-7500 Karlsruhe, FRG, 1993.

[50] Roelofs, A., The WEAVER model of word-form encoding in speech productions, *Cognition*, 1997, **64**:249-284.

[51] Rumelhart, D.E., G.E. Hinton, and R.J. Williams, Learning internal representations by error propagation, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, Cambridge, MA, **1**:318–362, 1986.

[52] Rummery, G.A., and M. Niranjan, On-line Q-learning using connectionist systems, *Technical Report CUED/F-INFENG/TR 166*, Engineering Department, Cambridge University.

[53] Samuel, A. L., Some studies in machine learning using the game of checkers, *IBM Journal on Research and Development*, 1959, **3**:211-229.

[54] Shultz, R., Temporal Sequence Representation in One-Shot, Multi-Winner Self-Organizing Maps, University of Maryland, Dissertation Proposal, August 15, 2002.

[55] Singh, S., and J. Black, A study of twenty-six intervocalic consonants as spoken and recognized by four language groups, *Journal of the Acoustic Society of America*, **39**(2):372-387, 1966.

[56] Singh, S., *Distinctive Features: Theory and Validation*, University Park Press, 1976.

[57] Sutton, R.S., Temporal Credit Assignment in Reinforcement Learning, Ph.D. thesis, University of Amherst.

[58] Sutton, R.S., Learning to predict by the method of temporal differences, *Machine Learning*, 1988, *3*:9-44.

[59] Sutton, R.S., *Reinforcement learning: An Introduction*, MIT Press, 1998.

[60] Tani, J., Model-Based Learning for Mobile Robot Navigation from the Dynamical Systems Perspective, *IEEE Trans. System, Man and Cybernetics (Part B), Special Issue on Learning Autonomous Robots*, 1996, Vol. 26, No. 3:421-436.

[61] Tesauro, G., TD-Gammon, A self-teaching Backgammon program, achieves master-level play, *Neural Computation*, **6**:215-219.

[62] Toudeft, A., and Patrick Gallinari, Distal learning for inverse modelling of dynamical systems.

[63] Watkins, C.J.C.H., Learning from Delayed Rewards, Ph.D. thesis, Cambridge University.

[64] Werbos, P.J., Backpropagation through time: What it does and how to do it, *Proceedings of the IEEE*, 1990, **78**:1550-1560.

[65] Whitehead, S., Ballard, D.H., Learning to perceive and act by trial and error., *Machine Learning*, 1991, **7**:45-83.

[66] Wikipedia contributors. Formant. *Wikipedia, The Free Encyclopedia.* December 13, 2006, 07:02 UTC. Available at: http://en.wikipedia.org/w/index.php?title=Formant&oldid=94008153. Accessed January 29, 2007.

[67] Williams, R.J., and Z. Zipser, 1989. A learning algorithm for continually running fully recurrent neural networks, *Neural Computation*, **1**:270-280.

[68] Williams, R.J., and J. Peng, An efficient gradient–based algorithm for on–line training of recurrent network trajectories, *Neural Computation 2* (1990), no. 4, 490–501.

[69] Williams, R.J., and D. Zipser, Gradient–based learning algorithms for recurrent networks and their computational complexity, *Backpropagation: Theory, Architectures and Applications (Y. Chauvin and D. E. Rumelhart, eds.)*, Lawrence Erlbaum Publishers, Hillsdale, N.J., 1995, pp. 433–486.

[70] Witney, A.G., Philipp Vetter, and Daniel M. Wolpert, The influence of previous experience on predictive motor control, *NeuroReport: Motor Systems*, Lippincott Williams and Wilkins, March, 2001, **12**(4):649-653.

[71] Wolpert, D.M., Zoubin Ghahramani, Michael I. Jordan, An Internal Model for Sensorimotor Integration, *Science*, **269**:1880-1882.

[72] Wolpert, D.M., R. Chris Miall, and Mitsuo Kawato 1998. Internal models in the cerebellum, Elsevier Science Ltd., pp. 338-347.

[73] Ziemke, T., Remembering how to behave: Recurrent neural networks for adaptive robot behavior, *Recurrent Neural Networks: Design and Applications* CRC Press. 1999. pp. 341-375.

Appendix A

Algorithm used for the preliminary Single Phoneme Acquisition Model

procedure BABBLE(*max_epochs, error_threshold*)

% — Initialize variables — %

Broca ← Broca's area neural model *% distal controller*

FM ← forward model neural net

X ← list of phoneme intent vectors

Y^* ← list of target audio phonemes *% distal target values*

U^* ← list of motor phonemes needed to produce Y^* distally *% Broca's task is to come up with the motor phoneme list on its own*

distal_error ← ∞

epochs ← 0

% — Initial Babbling Phase to train forward connections (forward model) — %

[rand_motor_list, rand_audio_list] ← generate random motor/audio phoneme pairs *% for use in babbling stages*

train FM on training pairs [rand_motor_list, rand_audio_list]

% — Training the Distal Learner, Broca's area — %

do

U ← Broca(X) *% list of outputs of Broca's area when presented with X as list of inputs*

Y ← Env(U) *% list of actual outputs resulting from applying Broca's motor response to*

the environment

$dW_{kj} \leftarrow 0$

$dW_{ji} \leftarrow 0$

for each phoneme intent x **in** X **do**

actual_delta $\leftarrow Y_x^* - Y_x$

train FM on training pair $[U_x, Y_x]$

$[dW_{kj}^x, dW_{ji}^x] \leftarrow$ calculate update weight matrices to Broca based on delta values propagated back through the forward model

$dW_{kj} \leftarrow dW_{kj} + dW_{kj}^x$

$dW_{ji} \leftarrow dW_{ji} + dW_{ji}^x$

end

Broca.W$_{kj}$ $\leftarrow$ Broca.W$_{kj}$ + dW_{kj} *% update Broca weight matrices*

Broca.W$_{ji}$ $\leftarrow$ Broca.W$_{ji}$ + dW_{ji}

train FM on training pairs [rand_motor_list, rand_audio_list] *% continue random babbling to further train forward connections*

epochs $\leftarrow$ epochs + 1

distal_error $\leftarrow$ calculate error of Broca's output (RMSE(Broca(X),U^*))

until (epochs $<$ max_epochs) or (distal_error $>$ error_threshold)

end

Appendix B

Creating a Smooth Mapping from a Finite Mapping

Constructing a smooth environment mapping from the space containing the set of motor feature vectors to that containing the set of auditory feature vectors presented a particular challenge. A candidate environment function, f^*, sought to complete a task such as this would preferably have a particular set of specific properties. Let A and B be finite sets such that $|A| = |B|$, $A \subset \Re^m$, and $B \subset \Re^n$. Define some finite mapping f:A→B such that f(A)=B. The idea is to construct the new smooth mapping, f^*, that preserves the finite mapping f but is as smooth and differentiable as is feasibly possible. This way, where f(a), for $a \in \Re^m$ but $a \notin A$, would be undefined, $f^*(a)$ would be some reasonable approximation for a counterpart in $\Re^n$. Once it behaves in this fashion, the environment function can be approximated effectively by a multi-layered feedforward neural network. The latter can in turn be used to propagate back the error of the actual distal output, which is a distal consequence of the controller's local action, from the desired target distal output.

To illustrate this problem, the following table demonstrates a very simple environment function, f:$\Re \rightarrow \Re^+$. Do note the domain and range of this function over $\Re^m$ and $\Re^n$, respectively, is as defined previously with m=n=1.

A	f(A) = B
0.4	0.6
1.6	1.8
2.9	0.4

As one potential candidate for a smooth mapping alternative, f^*, for f, we can set f of each

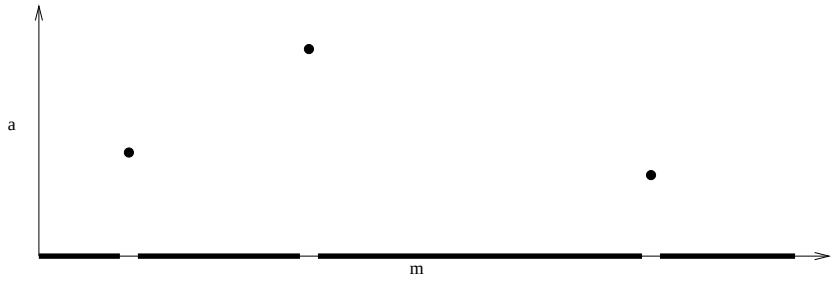


Figure B.1: Simple Mapping

member of A to the member of B to which it is associated (i.e. $f^*(\neg A) = 0$). For all other values m∈R, set f(m) = 0 (see Figure B.1). As such, this function satisfies the requirement that f(A) = B. However, no other information is encoded here, which is essential in training the controller effectively. Ideally, a function such as the one in Figure B.2 is sought. Using this function, any arbitrary m, even if it is not in A, has a defined f(m) whose value is dependent on the known values of B. A controller which offers some action m can then use the environment to judge how far off it was from achieving it's distal target and also modify itself to offer an action which is closer to the one required.

Unfortunately, arriving at a function such as the one in figure B.2 is not trivial. One way to approximate such a function is by using radial basis functions like that shown in Figure B.4. A radial basis function takes on the form $r(x) = exp(||x - c||^2/r^2)$, where the radius r determines the width of the resulting bell curve and c denotes the center. Here, $0 < r(x) \leq 1$, where r(x) = 1 if x = c and r(x) approaches 0 the further x is from c. Radial basis functions are used successfully in training radial basis neural nets [41] which have been shown to organize and learn from clustered input data better than standard neural networks.

Let y be the member of A such that $||x - c||$ is minimized (i.e. the closest A candidate in A to x). Initially, we will calculate f^*(x) as follows :

$A.1$ $\quad y = argmin_m||x - m||, m \in A$

$\quad f^*(x) = f(y) \times r_y(x)$

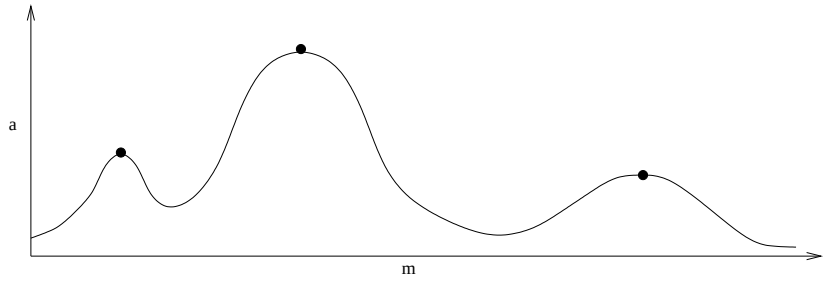


Figure B.2: Ideal Mapping

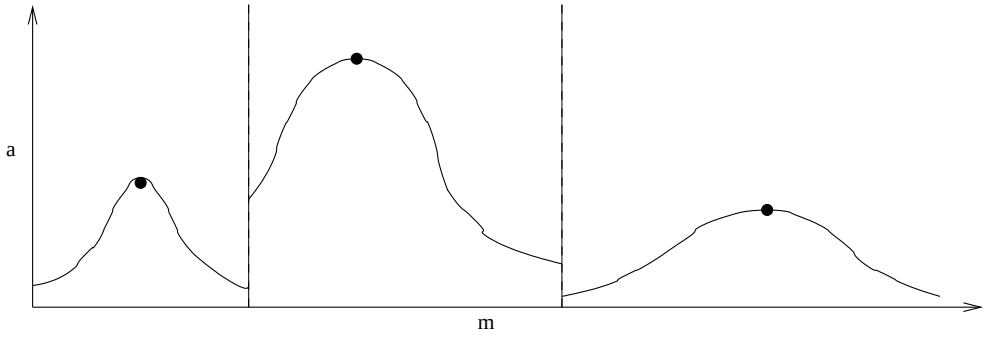


Figure B.3: Example figure of discontinuous mapping resulting from Equation A.1.

, where $r_y(x)$ is defined as the radial basis function centered at y. For x in A, y = x, $r_y(x) = 1$, and $f^*(m) = f(m) \times 1 = f(m)$. Otherwise, f^*(x) is assigned a multiple of the corresponding auditory phoneme to that closest motor feature vector, y, in A to x. The magnitude of this multiple will correspond inversely to the distance of x to the closest member of A.

The most significant problem with the function f^* is that it is highly discontinuous. The function landscape changes abruptly midway between neighboring members of A (Figure B.3). One way to offset such extreme discontinuities could incorporate adding a smoothing factor to Equation A.1 which takes into consideration the proximity of all candidate elements of the domain A in calculating f^*(x).

The new environment function, f^*(x), is now calculated as follows:

$A.2$ $\quad g(m,x) = 1/(||x-m||)^b; m \in A, b \geq 1, M = |A|$

$\quad h(m,x) = g(m,x)/\sum_y^M g(y,x); y \in A$

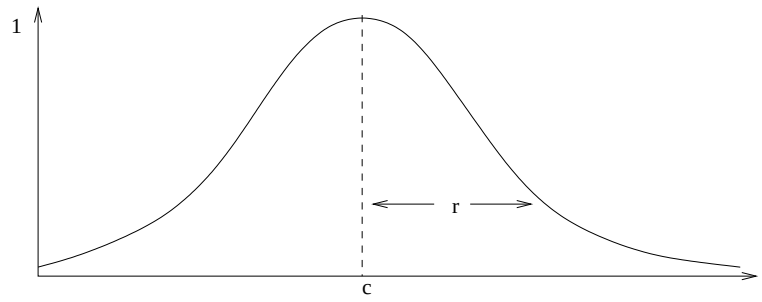


Figure B.4: Radial Basis Function

$$f^*(x) = \sum_z^M [h(z,x) \times f(z) \times r_z(x)]; z \in A$$

Here, g(m,x) is a measure for the proximity of the member m of set A to the input vector x. The smaller $||x - m||$, the larger g(m,x) becomes. The function h(m,x) is essentially a normalized version of g(m,x) such that $0 < h(m,x) < 1$. As a result, h(m,x) will approach 1 if x is very close to some member m∈A. A consequence of this is h(y,x) for all other y∈A will approach 0 since $\sum_z h(z,x) = 1$. Function f^*(x) will then take on most of the characteristics of f(m). Otherwise, should x be found to be midway between two or more members of A, f^*(x) should take on characteristics of all of their corresponding mappings of the target set B.

One drawback to constructing the environment mapping, f^*, in this manner is that it requires significantly more computation than that of Equation A.1. Even so, however, the resulting mapping is sufficiently smooth enough for the forward model to learn to approximate. Figure B.6 demonstrates two candidate function landscapes for transforming a finite mapping f to a smooth mapping f^*:[-1,1]^2 →[0,1] based on Equations A.1 and A.2.

One issue encountered in creating a function in this fashion is that those members m∈A which have large values for f(m)∈B can have radial basis mounds which disproportionately dominate values of $f^*(x)$ within some proximity of m despite the presence of other nearby radial basis mounds. This can have undesirable results where some large radial basis mounds envelope smaller ones or even create “false” mounds not centered around a member in A (figure B.5). As such, further improvement to f^*(x) can be obtained by ”slimming” the radial basis component assigned

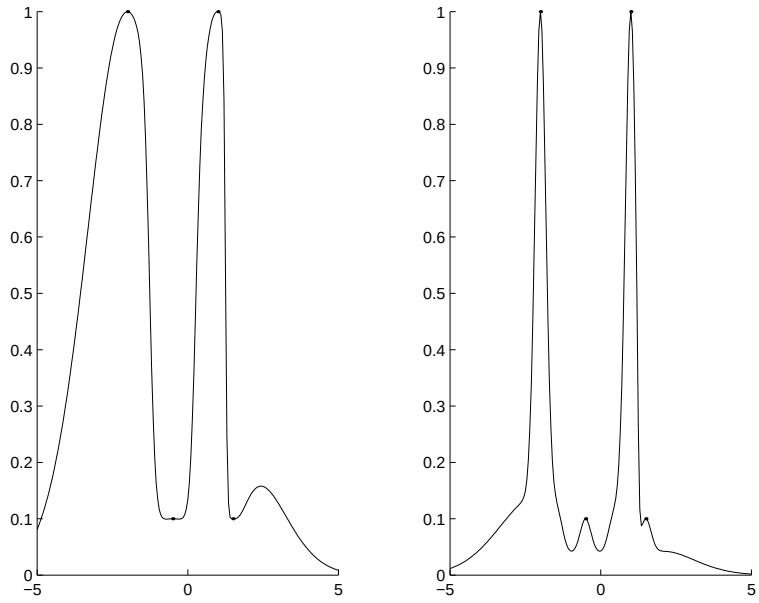


Figure B.5: (Left)The smooth mapping procedure shown without radius slimming. Notice that members of the domain with the smallest corresponding $f^*(x)$, (x=-.5 and x=1), have no radial basis mounds as they are being dominated by mounds of members with very large $f^*(x)$. Also notice the false mound created to the far left which corresponds to no member of the domain set, A. (Right) The same procedure using the radius slimming modification. By reducing the radii of the tallest mounds, the false mound disappears and the radial basis mounds for the members with small f^* are much more apparent.

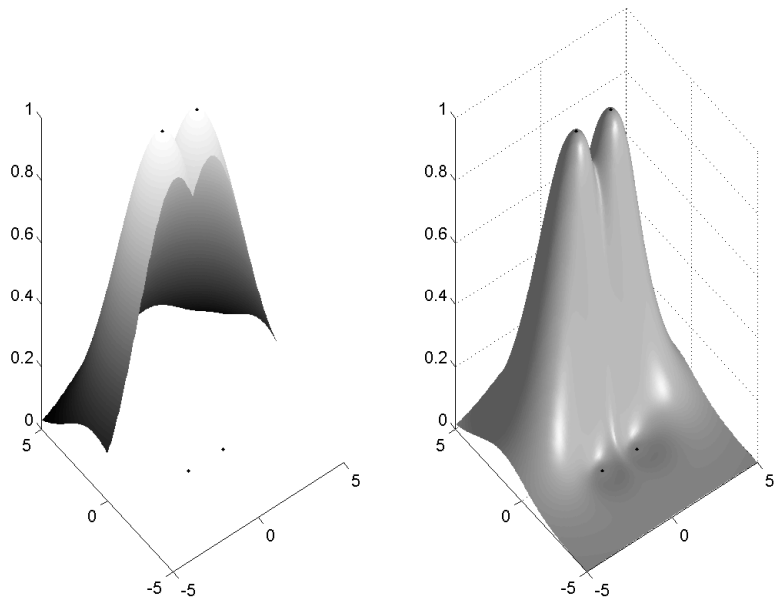


Figure B.6: Here two mapping methods are compared using Equation A.1 (left) and using Equation A.2 (right). The finite relation used to create these smooth mappings is as follows : (-1,-1) →0, (1,-1) →0, (1,-1) →1, (1,1) →1.

to a member m in A with maximum height f(m) in B. This can be achieved by reducing the radius term, r, in the radial basis portion of f^*(x) for larger values of f(y) and gradually increasing radii for mounds with smaller maximum heights.

Another obstacle in constructing the environment function in this manner stems from having to deal with zero output. When approximating a smooth function in this fashion, not only is it difficult to approximate zero valued outputs but since a radial basis mound of height zero is essentially non-existent it can contribute very little to the weighted averages introduced in the construction of this mapping. To alleviate the problem to a degree, some minimum value greater than zero can be assigned to replace all zeros in the feature vectors of A and B, thereby giving even null values radial basis information which can be utilized.

Appendix C

Motor / Auditory Feature Tables for English Language Phonemes for Use in Phoneme Sequence Production Task.

This section lists the essential English language phonemes used in the preliminary work of the single phoneme acquisition model (section 5.2) and intended for use in creating the proposed phoneme sequence acquisition computational brain model (section 5.3). Each column represents the vectors of known features which characterize a given phoneme. The tables are divided into motor phoneme and auditory phoneme tables and further divided into vowel and consonant tables. Here, motor phonemes denote commands which are produced through the primary motor cortex to produce a phonetic sound, while an auditory phoneme denotes the phonetic sound impressed on the primary auditory cortex upon hearing.

These tables were provided by Schultz [54] by combining feature systems from work done by Jakobsen, et al.[20] and Singh et al. ([55],[56]). Features known to be present in a phoneme are denoted by a '+' in the column while their absence is signaled by a '-'. Altogether, there are forty-one such phonemes but three are omitted as they are functionally equivalent to other phonemes already listed. In simulations for this study, each phoneme column can be regarded as vectors in the space $\{0,1\}^{21}$ for motor phonemes and $\{0,1\}^{34}$ for auditory phonemes by replacing '+'s and '-'s by 1's and 0's, respectively.

IPA	p	b	m	t	d	n	ʧ	ʤ	k	g	f	v	θ	ð	s	z	ʃ	ʒ	w	r	l	j	h	ŋ
Keyboard compatible	p	b	m	t	d	n	tch	dj	k	g	f	v	th–	th+	s	z	sh	zh	w	r	l	y	h	ng
Consonantal	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Vocalic	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Anterior	+	+	+	+	+	+	.	.	.	.	+	+	+	+	+	+	.	.	.	.	+	.	.	.
Coronal	.	.	.	+	+	+	+	+	.	.	.	.	+	+	+	+	+	+	.	+	+	.	.	.
+Voicing	.	+	+	.	+	+	.	+	.	+	.	+	.	+	.	+	.	+	+	+	+	+	.	+
–Voicing	+	.	.	+	.	.	+	.	+	.	+	.	+	.	+	.	+	.	.	.	.	.	+	.
Continuant	.	.	.	.	.	.	.	.	.	.	+	+	+	+	+	+	+	+	+	+	+	+	+	.
Stop	+	+	+	+	+	+	+	+	+	+	.	.	.	.	.	.	.	.	.	.	.	.	.	+
Nasal	.	.	+	.	.	+	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	+
Strident	.	.	.	.	.	.	+	+	.	.	+	+	.	.	+	+	+	+	.	.	.	.	.	.
Height: VH	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
H	.	.	.	.	.	.	+	+	+	+	.	.	.	.	.	.	+	+	+	.	.	+	.	+
M	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
L	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	+	.
VL	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Advancement: F	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
FC	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
C	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
BC	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
C	.	.	.	.	.	.	.	.	+	+	.	.	.	.	.	.	.	.	+	.	.	.	.	+

Table C.1: Distinct Feature System for Consonants (Motor)

IPA		o	a	e	u	ə	i	I	ɛ	æ	ʌ	ʊ	ɔ	ɚ	ai	əʊ
Keyboard compatible		o	ah	ay	oo	uh-	ee	ih	eh	ae	uh+	u	aw	er	ai	au
Consonantal		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Vocalic		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Anterior		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Coronal		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
+Voicing		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
–Voicing		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Continuant		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Stop		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Nasal		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Strident		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Height:	VH	.	.	.	+	.	+	.	.	.	.	.	.	.	+	+
	H	.	.	.	.	.	.	+	.	.	.	+	.	.	.	.
	M	+	.	+	.	+	.	.	.	.	.	.	.	+	.	+
	L	.	.	.	.	.	.	.	+	.	+	.	+	.	.	.
	VL	.	+	.	.	.	.	.	.	+	.	.	.	.	+	.
Advancement:	F	.	.	+	.	.	+	+	+	+	.	.	.	.	+	.
	FC	.	.	.	.	+	.	.	.	.	.	.	.	+	.	+
	C	.	.	.	.	.	.	.	.	.	.	.	.	+	.	.
	BC	.	.	.	.	.	.	.	.	.	+	.	.	.	.	.
	C	+	+	.	+	.	.	.	.	.	.	+	+	.	+	+

Table C.2: Distinct Feature System for Vowels (Motor)

IPA	p	b	m	t	d	n	ʧ	ʤ	k	g	f	v	θ	ð	s	z	ʃ	ʒ	w	r	l	j	h	ŋ
Keyboard compatible	p	b	m	t	d	n	tch	dj	k	g	f	v	th–	th+	s	z	sh	zh	w	r	l	y	h	ng
Consonantal	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Vocalic	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Compact	.	.	.	.	.	.	+	+	+	+	.	.	.	.	.	.	+	+	.	.	.	.	.	+
Diffuse	+	+	+	+	+	+	.	.	.	.	+	+	+	+	+	+	.	.	.	.	.	.	.	.
Grave	+	+	+	.	.	.	.	.	.	.	+	+	.	.	.	.	.	.	.	.	.	.	.	.
Acute	.	.	.	+	+	+	.	.	.	.	.	.	+	+	+	+	.	.	.	.	.	.	.	.
Nasal	.	.	+	.	.	+	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	+
Oral	+	+	.	+	+	.	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	.
Tense	+	.	.	+	.	.	+	.	+	.	+	.	+	.	+	.	+	.	.	.	.	.	+	.
Lax	.	+	.	.	+	.	.	+	.	+	.	+	.	+	.	+	.	+	.	.	.	.	.	.
Continuant	.	.	.	.	.	.	.	.	.	.	+	+	+	+	+	+	+	+	+	.	.	.	.	.
Interrupted	+	+	.	+	+	.	+	+	+	+	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Strident	.	.	.	.	.	.	+	+	.	.	.	.	.	.	+	+	.	.	.	.	.	.	.	.
Mellow	.	.	.	.	.	.	.	.	+	+	.	.	+	+	.	.	.	.	.	.	.	.	.	.
+Voicing	.	+	+	.	+	+	.	+	.	+	.	+	.	+	.	+	.	+	+	+	+	+	.	+
–Voicing	+	.	.	+	.	.	+	.	+	.	+	.	+	.	+	.	+	.	.	.	.	.	+	.
+Duration	.	.	.	.	.	.	.	.	.	.	.	.	.	.	+	+	+	+	.	.	.	.	.	.
–Duration	+	+	+	+	+	+	+	+	+	+	+	+	+	+	.	.	.	.	+	+	+	+	+	+
+(Af)Frication	.	.	.	.	.	.	+	+	.	.	+	+	+	+	+	+	+	+	.	.	.	.	+	.
–(Af)Frication	+	+	+	+	+	+	.	.	+	+	.	.	.	.	.	.	.	.	+	+	+	+	.	+
Liquid	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	+	+	.	.	.
Glide	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	+	.	.	+	.	.
Retroflex	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	+	.	.	.	.
$F_{2,VH}$	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
$F_{2,H}$	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
$F_{2,HM}$	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
$F_{2,LM}$	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
$F_{2,L}$	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
$F_{2,VL}/F_{1,VH}$	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
$F_{1,H}$	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
$F_{1,HM}$	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
$F_{1,LM}$	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
$F_{1,L}$	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
$F_{1,VL}$	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Table C.3: Distinct Feature System for Consonants (Auditory)

IPA	o	a	e	u	ə	i	I	ɛ	æ	ʌ	ʊ	ɔ	ɚ	ai	ɔʊ
Keyboard compatible	o	ah	ay	oo	uh-	ee	ih	eh	ae	uh+	u	aw	er	ai	au
Consonantal	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Vocalic	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Compact	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Diffuse	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Grave	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Acute	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Nasal	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Oral	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Tense	+	+	+	+	.	+	.	.	.	.	.	.	+	+	.
Lax	.	.	.	.	+	.	+	+	+	+	+	+	.	.	+
Continuant	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Interrupted	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Strident	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Mellow	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
+Voicing	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
–Voicing	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
+Duration	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
–Duration	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
+(Af)Frication	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
–(Af)Frication	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Liquid	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Glide	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
Retroflex	.	.	.	.	.	.	.	.	.	.	.	.	+	.	.
$F_{2,VH}$	.	.	+	.	.	+	+	.	.	.	.	.	.	.	.
$F_{2,H}$	.	.	.	.	.	.	.	+	+	.	.	.	.	+	.
$F_{2,HM}$	.	.	.	.	+	.	.	.	.	+	.	.	+	.	.
$F_{2,LM}$	.	+	.	.	.	.	.	.	.	.	.	.	.	.	+
$F_{2,L}$	.	.	.	.	.	.	.	.	.	.	+	+	.	.	.
$F_{2,VL}/F_{1,VH}$	+	.	.	+	.	.	.	.	.	+	.	.	.	.	.
$F_{1,H}$	.	+	.	.	+	.	.	.	+	.	.	.	.	.	.
$F_{1,HM}$	.	.	.	.	.	.	.	+	.	.	.	+	.	.	.
$F_{1,LM}$	+	.	+	.	.	.	.	.	.	.	.	.	+	+	+
$F_{1,L}$	.	.	.	+	.	.	+	.	.	.	.	.	.	.	.
$F_{1,VL}$	.	.	.	.	.	+	.	.	.	.	+	.	.	.	.

Table C.4: Distinct Feature System for Vowels (Auditory)