2008-03-18

Improving Liquid State Machines Through Iterative Refinement of the Reservoir

R David Norton
Brigham Young University - Provo

IMPROVING LIQUID STATE MACHINES THROUGH ITERATIVE REFINEMENT OF THE RESERVOIR

by

R. David Norton

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

April 2008

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

R. David Norton

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Dan Ventura, Chair
Date	Michael A. Goodrich
Date	Charles Knutson

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of R. David Norton in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Dan Ventura Chair, Graduate Committee
Accepted for the Department	Parris K. Egbert Graduate Coordinator
Accepted for the College	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

IMPROVING LIQUID STATE MACHINES THROUGH
ITERATIVE REFINEMENT OF THE RESERVOIR

R. David Norton
Department of Computer Science
Master of Science

Liquid State Machines (LSMs) exploit the power of recurrent spiking neural networks (SNNs) without training the SNN. Instead, a reservoir, or liquid, is randomly created which acts as a filter for a readout function. We develop three methods for iteratively refining a randomly generated liquid to create a more effective one. First, we apply Hebbian learning to LSMs by building the liquid with spike-time dependant plasticity (STDP) synapses. Second, we create an eligibility based reinforcement learning algorithm for synaptic development. Third, we apply principles of Hebbian learning and reinforcement learning to create a new algorithm called separation driven synaptic modification (SDSM). These three methods are compared across four artificial pattern recognition problems, generating only fifty liquids for each problem. Each of these algorithms shows overall improvements to LSMs with SDSM demonstrating the greatest improvement. SDSM is also shown to generalize well and outperforms traditional LSMs when presented with speech data obtained from the TIMIT dataset.

Contents

List of Figures

Chapter 1

Introduction

1.1 Neural Networks

Artificial neural networks (ANNs) are a biologically inspired model for computation derived from the architecture of the living nervous system. However, as with all models, traditional ANNs are an abstraction of the actual system they model. Despite this high level of abstraction, ANNs have shown significant computational potential and have undergone many refinements in an attempt to mimic the computational power that is noticeably present in biological systems. Spiking neural networks, or SNNs, are one refinement of the model that more closely models the biological counterpart.

The spiking neurons that make up SNNs emulate biological neurons by transmitting signals in a sequence of spikes with each spike having a constant amplitude. Information is encoded in the varying frequencies of spikes produced by the neurons rather than in the single rate value commonly used in non-spiking networks [8]. SNNs can convey temporal information more accurately by maintaining non-uniform spiking frequencies. This allows SNNs to have greater computational power for problems in which timing is important [1] [15], including such problems as speech recognition, biometrics, and robot control. Even in problems where timing isn't an explicit factor, SNNs achieve competitive results with fewer neurons than traditional ANNs [2] [3].

Aside from SNNs, another refinement of neural networks that more closely models biological systems is the incorporation of recurrence, defined as allowing cycles to occur within the network. This property theoretically improves ANNs by allowing neural activity at one point in time to effect neural activity at a later time. In other words, context is incorporated directly into the network, endowing the network with the capability for solving timing-critical problems. SNNs are often created with recurrence as a default attribute.

Despite the obvious advantages of these two types of neural networks, both suffer from a distinct lack of satisfactory training algorithms. Concerning the recurrent networks alone, only a handful of established algorithms exist, all of which have very high computational costs, limiting them to very small networks [12]. All of these methods also require very specific and sensitive parameter settings for each application in which they are used. Training SNNs is also a developing area of research. Most SNN training algorithms currently proposed only allow for a single output spike from each neuron [2] [3] [17]. This is unrealistic and computationally limiting.

1.2 Liquid State Machines

One approach for harnessing the power of recurrent SNNs without actually training them is called the liquid state machine, or LSM [18] [19]. LSMs are a type of reservoir computing comparable to echo state networks (ESN) and Backpropagation Decorrelation (BPDC) [22], neither of which employs spiking neurons. LSMs are composed of two parts: a *reservoir* featuring a highly recurrent SNN, and a *readout function* characterized by a simple learning function. Input is fed into the reservoir which acts as a filter. Then the state of the reservoir, or *state vector*, is used as input for the readout function. In essence, the readout function trains on the output of the reservoir. No training occurs within the reservoir itself. This process has been analogized with dropping objects into a container of liquid and subsequently reading

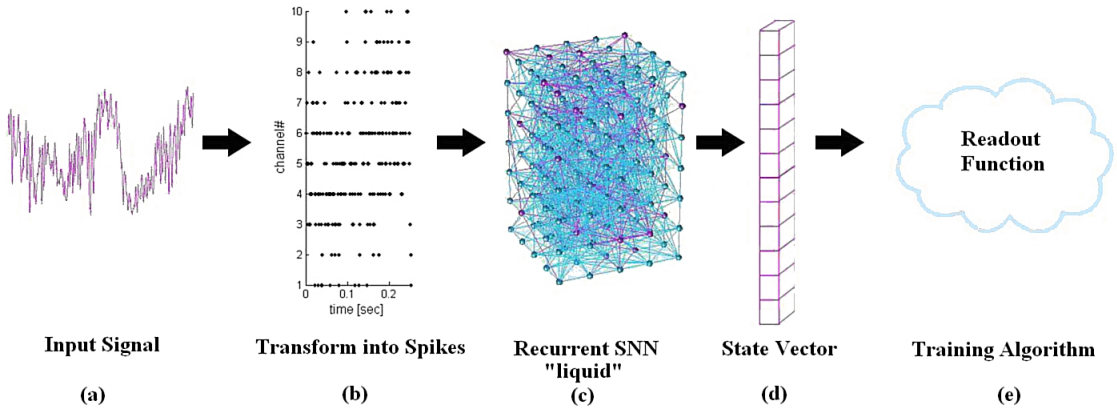


Figure 1.1: Diagram of a liquid state machine. (a, b) The input signal is transformed into a series of spikes via some function. (c) The spike train is introduced into the recurrent SNN, or "liquid". (d) Snapshots of the state of the "liquid" are recorded in state vectors. (e) The state vectors are used as input to train a learning algorithm, the readout function.

the ripples created to classify the objects—hence the name liquid state machine. See Figure 1.1.

Because no training occurs in the reservoir, the quality of the LSM is dependent upon the ability of the liquid to effectively separate classes of input. Here the term "effectively separate" is defined as the ability of a liquid to yield a state vector with which a readout function can attain acceptable accuracy. Typically, the liquid is created randomly according to some carefully selected parameters specific to the problem at hand. This parameter selection has been the topic of much research [9] [22] although the research has not yet led to a consistent and general method of generating liquids for all problems [23]. Even when adequate parameters for a given problem have been implemented, the creation of a useful liquid is not guaranteed. Typically hundreds or even thousands of liquids will be created to find a suitable filter. Fortunately, once such a liquid is discovered, the results of the LSM are comparable with the state-of-the-art [11] [22] [24]. Such results, coupled with the lack of

a sufficient training algorithm to train these latent networks, fuel the exploration of LSMs.

1.3 Thesis Description

It is clear that a general method for creating effective liquids in an LSM without having to rely on the generation of many random liquids would be useful. In this thesis we approach this problem by randomly creating a liquid in the traditional fashion and then adjusting the liquid's architecture until it can "effectively separate" as defined above. We present the liquid with sample data, observe the resulting behavior, and then use these observations to make the necessary changes to the liquid. This observation of behavior may focus on individual neurons or be more general depending on the specific algorithm we are using. Although this approach essentially involves using training data to modify the liquid, it is not a standard training algorithm in that the goal of our method is different. The goal is to create a liquid that will effectively separate classes of input into different patterns of state vectors. Afterwards, the readout function will learn to extract information from the state vectors via training.

1.4 Thesis Overview

We begin in Chapter 2 by defining a separation metric to evaluate how well a liquid can "effectively separate" classes of input. This separation metric will be used throughout the rest of the thesis. In the next three chapters we introduce three algorithms for iteratively refining the reservoir of a liquid state machine. In Chapter 3 we introduce Hebbian learning to LSMs, an unsupervised approach that applies changeable synapses to the liquid. We also show the results of using both random noise and speech data as input into such a liquid. The contents of this

chapter were published in the 2006 proceedings of IJCNN. Chapter 4 describes a new type of reinforcement algorithm that can be used in conjunction with LSMs and shows the results of preliminary experiments involving simple problems. In Chapter 5 we define a new algorithm called separation driven synaptic modification (SDSM) that is inspired by Hebbian and reinforcement learning. Chapter 6 shows the results of applying SDSM to several artificial problems. This chapter also explores the ability of liquids created with SDSM to generalize to a variety of problems. In Chapter 7 we compare all three of our algorithms across the artificial problems outlined in Chapter 6 and find that all perform better than traditional LSMs. In Chapter 8 we explore SDSM further by testing it with real world speech data from the TIMIT dataset. Finally, we summarize our conclusions and recommend future work in Chapter 9. All of the LSMs used in this thesis are created using CSIM, "a tool for simulating heterogeneous networks composed of different model neurons and synapses" [18].

Chapter 2

Separation

Separation is a metric used to determine the effectiveness of a liquid. It essentially measures how well the liquid separates different classes of input into different reservoir states, or state vectors, and is analogous to supervised clustering. A metric for separation was first devised by Goodman [11] and is inspired by the description of the properties of a liquid presented by Maass [16]. Goodman's separation metric is shown in Equation 2.1.

$$Sep_{\Psi}(O) = \sum_{m=1}^{N} \sum_{n=1}^{N} \frac{\|\mu(O_m) - \mu(O_n)\|_2}{N^2} \tag{2.1}$$

Here Ψ is the liquid and O is the set of all state vectors induced by Ψ divided into subsets, O_m, for each class. N is the total number of classes, so O contains N subsets. $\mu(O_m)$ is the center of mass for all of the state vectors of class m and is calculated by Equation 2.2. Goodman's definition of separation essentially finds the mean distance between the center of mass for every pair of classes. In the following equation, o_n is an individual state vector in the subset O_m.

$$\mu(O_m) = \frac{\sum_{o_n \in O_m} o_n}{|O_m|} \tag{2.2}$$

Goodman's definition of separation is used for all of the experiments with Hebbian learning in Chapter 3.

2.1 An Improved Separation Metric

In order to more accurately perform the desired measure of separation, we have revised Goodman's definition to take into consideration the variance in state vectors. With Goodman's metric, a liquid could be attributed with high separation while having great overlap across the different clusters of state vectors, as long as the centers of mass for these clusters were divergent. This situation is problematic for the readout function since it becomes difficult to delineate different classes. Our definition of separation presented in a similar form to Goodman's (for comparative purposes) is shown in Equation 2.3.

$$Sep_{\Psi}(O) = \sum_{m=1}^{N} \sum_{n=1}^{N} \frac{\|\mu(O_m) - \mu(O_n)\|_2}{N^2 + N \sum_{m=1}^{N} \rho(O_m)} \tag{2.3}$$

Here all variables and functions have the same meaning as those in the previous definition. The additional function $\rho(O_m)$ is the amount of variance within a class of state vectors and is calculated by Equation 2.4. Our separation metric essentially finds the mean distance between the center of mass for every pair of classes, and then divides it by a function of the class variances. This decreases the separation value for liquids that yield overlapping clusters of state vectors, thus presenting a more accurate representation of separability in terms of the readout function.

$$\rho(O_m) = \frac{\sum_{o_n \in O_m} \|\mu(O_m) - o_n\|_2}{|O_m|} \tag{2.4}$$

2.2 A Time-dependent Version of Separation

The representation in Equation 2.3 allows for a direct comparison to Goodman's original separation metric. However, due to the separation-dependent nature of the SDSM algorithm that will be defined and explored in Chapters 5 and 6, we need to include time in the above separation metric. This section shows the derivation of

a more readable version of our separation metric that includes the variable of time and new terms that will be necessary in understanding SDSM. For clarification, time refers to the iteration of a given synapse modifying algorithm.

Separation is calculated with the set of state vectors, O, as described previously with the added variable of time making it $O(t)$. $O(t)$ is divided into N subsets, $O_m(t)$, one for each class, where N is the total number of classes. The center of mass for each class, m, can be calculated with Equation 2.5. Equation 2.6 is the average amount of variance for each state vector within class m from the center of mass for that class.

$$\mu(O_m(t)) = \frac{\sum_{o_n \in O_m(t)} o_n}{|O_m(t)|} \tag{2.5}$$

$$\rho(O_m(t)) = \frac{\sum_{o_n \in O_m(t)} \|\mu(O_m(t)) - o_n\|_2}{|O_m(t)|} \tag{2.6}$$

Separation is divided into two parts, the inter-class distance, $C_d(t)$, and the intra-class variance, $C_v(t)$. $C_d(t)$ is defined by Equation 2.7 and is the mean distance between every combination of $\mu(O_m(t))$. $C_v(t)$ is defined by Equation 2.8 and is the mean variance of every cluster of state vectors. Separation can now be defined by Equation 2.9. $C_v(t)$ is incremented by one to ensure that separation never approaches ∞.

$$C_d(t) = \sum_{m=1}^{N} \sum_{n=1}^{N} \frac{\|\mu(O_m(t)) - \mu(O_n(t))\|_2}{N^2} \tag{2.7}$$

$$C_v(t) = \frac{1}{N} \sum_{m=1}^{N} \rho(O_m(t)) \tag{2.8}$$

$$Sep_{\Psi}(O(t)) = \frac{C_d(t)}{C_v(t) + 1} \tag{2.9}$$

Careful examination of Equation 2.9 will reveal that it is identical to Equation 2.3 for a given iteration t. With the exception of Chapter 3, all references to separation throughout this thesis refer to Equation 2.9 and not Goodman's original metric.

2.3 Validation of Separation Metric

In Figure 2.1 we show that separation as defined in Equation 2.9, does correlate with the effectiveness of a liquid. Here, effectiveness is measured as the accuracy of the LSM at classifying inputs in an artificial problem. One thousand liquids were generated with varying parameter settings to create a large variety of separation values. The artificial problem consisted of five input classes expressed as spiking patterns for four input neurons. Separation was calculated with only three examples from each class. Since we were not applying a synapse modifying algorithm to the liquids, only one iteration, t, was observed. The correlation coefficient between accuracy and separation is a convincing 0.6876.

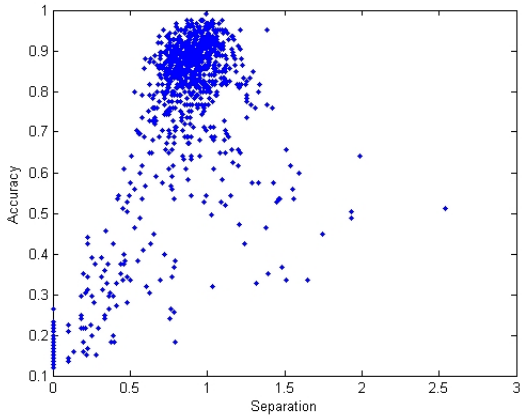


Figure 2.1: Correlation between accuracy and separation in 1000 different liquids run on an artificial problem. The correlation coefficient is 0.6876.

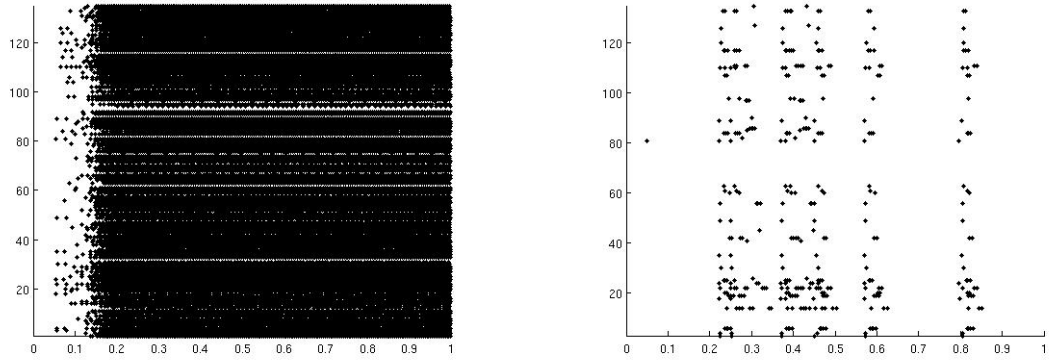
Chapter 3

Hebbian Learning

Hebbian learning is often implemented in recurrent SNNs with STDP synapses (spike-time-dependant plasticity synapses). As with other spiking synapses, there is a weight and time delay associated with the synapse. In addition, the STDP synapse has several other parameters that are related to how its weight changes as its pre- and post-synaptic neurons fire [8]. The synapse's weight changes in proportion to the temporal correlation between the pre- and post-synaptic neurons. If the pre-synaptic neuron fires first, then the weight is increased; if the post-synaptic neuron fires first then the weight is decreased. In this way, synapses that participate in a neuron's breach of threshold (resulting in a spike) are strengthened while those that don't are weakened. We refer to LSMs that use STDP synapses as a Hebbian Liquid State Machines (HLSMs). This chapter looks at two experiments with HLSMs to try and understand how Hebbian learning affects LSMs. The contents of this chapter were published in the 2006 proceedings of IJCNN [20].

3.1 Pathological Synchrony and Over-Stratification

One observation we will make in the first experiment of this chapter is the effect of Hebbian learning on liquids exhibiting two negative behaviors that are common to randomly generated liquids. These behaviors can significantly decrease the separation of a liquid and are termed *pathological synchrony* and *over-stratification*. Pathological



(a) Pathological Synchrony (b) Over-stratification

Figure 3.1: Behavior of liquids exhibiting pathologcial synchrony and over-stratification. The x-axis shows the passage of time while the y-axis shows the 135 neurons making up these liquids. Dots indicate occurrence of a spike for a given neuron.

synchrony occurs when most of the neurons in the liquid get caught in infinite positive feedback loops with respect to their firing. These infinite loops continuously influence the state of the liquid overriding the flow of information from the input. In extreme cases the entire liquid can begin firing continuously in synchrony as in Figure 3.1(a). Such liquids have low separation because of the loss of pattern associated with such crowded spiking. Over-stratification occurs when groups of neurons do not propagate a series of spikes induced by an input spike long enough. In these cases, input spikes do not influence each other within the liquid, thus resulting in a loss of temporal coherency that can be represented by the liquid as in Figure 3.1(b).

3.2 Effects of Random Input on Separation

The first experiment in this chapter explores the effect of a single channel of random input on HLSM separation in order to better understand the dynamics of the HLSM architecture. Two initial liquid states were investigated, a state in pathological synchrony and an over-stratification state. These initial states were selected in order

to observe how Hebbian Learning could potentially recover from them. For both initial states, actual Hebbian learning and random weight updates were compared resulting in a total of four sub-experiments.

3.2.1 Methods

Each of the four sub-experiments employed 100 iterations of training on the liquid. This training was either Hebbian learning or random weight alterations. For each iteration of training the separation of the liquid was determined with a set of state vectors, O, of size 100. Each state vector in O was created by introducing a randomly generated train of 25 spikes over a 1.0 second time interval, d, as input into the liquid. The state vector was measured at time d with $e = 1.0\ ms$. Since the input was random, each state vector belonged to a unique output class. Thus, for this experiment $N = O$, where N is the total number of classes in a given problem.

Each liquid was prepared with 135 neurons to be comparable to previous research [10]. The input was encoded as a spike train from a single input neuron. The remainder of the settings were chosen based on a series of preliminary experiments and reflect the best results obtained in those trials. The connection probability from the input neuron to the other inter-neurons was 0.1 while the probability of connection between the inter-neurons was 0.05. The mean delay for the synapses of the liquid was 10 ms with a standard deviation of 1 ms. To induce the negative behaviors in initial liquids, the mean weights of the synapses were adjusted accordingly. For the liquids initiated in a pathological synchrony state, the mean weight value for synapses was set at 1×10^{-7} while the mean weight value in liquids initiated in an over-stratification state was set at 8×10^{-8}. The standard deviation of weight values for both initial states was 1×10^{-8}.

For the Hebbian learning, all STDP Synapse settings were selected based on preliminary experiments. The maximum weight value allowed for all synapses in

both Hebbian learning and random weight update experiments was 1×10^{-5}. Each training of the liquid involved introducing a randomly generated train of 25 spikes over a 1.0 second time interval. During this input interval and in the case of Hebbian learning, the weights were allowed to change in accordance with the STDP synapse rules outlined earlier. For random weight updates, each synapse's weight was updated at the end of the input interval by a value drawn from a normal distribution with a mean of -2.8742×10^{-8} and a standard deviation of 1.5726×10^{-7}. This mean and standard deviation were obtained by calculating and averaging the mean and standard deviation of weight changes in ten preliminary runs through unique liquids using Hebbian Learning. Thus, the values, though random, represent reasonable changes in weight for the liquids in this study.

3.2.2 Results

The results of the above experiment are seen in Figure 3.2. For each of the four experiments, the average separation of ten unique liquids is displayed. The Hebbian learning trials don't show a significant change in separation while the random weight update trials drop steadily in separation after only ten iterations.

Figure 3.3 demonstrates how the physical structure of the liquid changes with training. The images show how the synapses (lines) connect to each of the neurons (dots). The brighter the synapse, the stronger the magnitude of the weight. We dont differentiate between positive and negative weights. Black synapses have effectively zero weight. When stimulated with random input, Hebbian learning eliminates many of the synapses while strengthening those that remain. Random weight updates, on the other hand, results in overall significantly strengthened synapses after random stimulation.

Figure 3.4 and Figure 3.5 demonstrate how the spiking patterns of each experiment change with training. For each graph, the x-axis represents time in seconds and

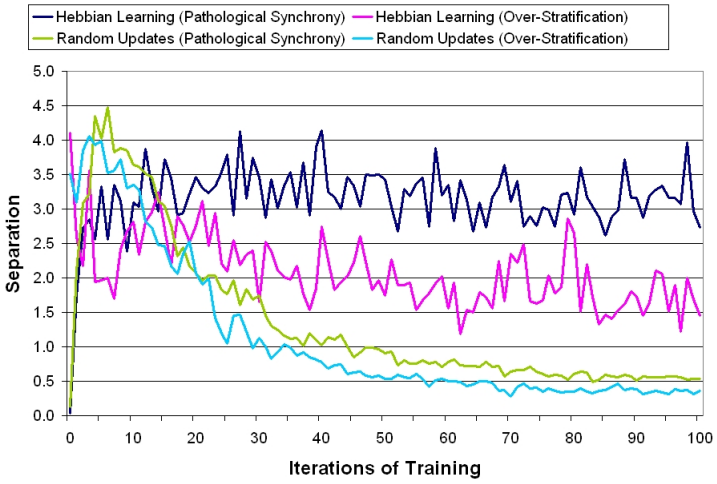


Figure 3.2: Separation values for four experiments given completely random input. Separation values are the mean reported by ten trials each with a different initial liquid. The Hebbian learning trials don't show a significant change in separation while the random weight update trials show a steady drop in separation after only ten iterations.

the y-axis the neuron ID number (there are 135 total). Hebbian learning relieves the state of pathological synchrony as seen in the reduction of firing (Figure 3.4). It also overcomes over-stratification by generating denser firing patterns. Random weight updates results in over-stratification regardless of the initial state (Figure 3.5). This seems unusual since the synapses are much stronger according to the results in Figure 3.3. This occurs because most of the synapses become strongly inhibitory due to the mean negative weight update.

3.2.3 Discussion

This experiment showed that given ideal initial liquids, Hebbian learning cannot improve separation with random input. However, the experiment also showed that given poor initial conditions Hebbian learning can improve performance. Finally, the experiment showed that even under the fabricated random input scenario, Hebbian learning does more than simply randomly update weights.

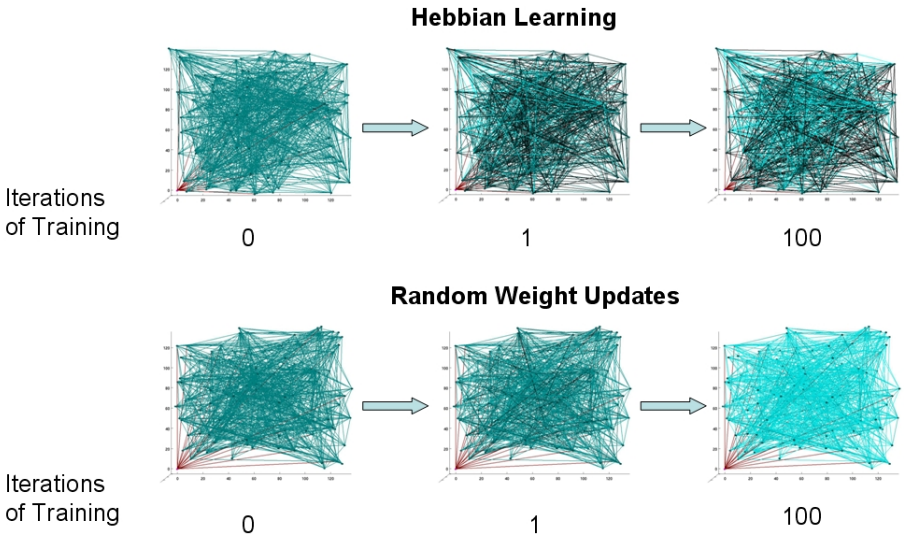


Figure 3.3: The physical characteristics of the liquid can be seen before and after training. Bright colors indicate strong weights (either negative or positive), dark colors indicate weak weights. Top: Hebbian learning eliminates many of the synapses while strengthening those that remain. Bottom: Random weight updates results in overall significantly strengthened synapses after random stimulation.

In the experiments initiated in a pathological synchronous state, both random update and Hebbian learning improved the separation of the liquids dramatically after only a single iteration of training. In fact, in the initial pathological state, the separation was zero in all trials. This dramatic improvement can best be explained by the assumption that arbitrary pruning of synapses reduces the number of infinite loops in the liquid. This also concurs with previous findings that investigated the reduction of neuron inter-connections to reduce synchronous firing [11].

Other than the initial improvement in pathological states, Hebbian learning doesn't improve the separation of the liquid over successive iterations of training. Also, the amount of separation at each level of training fluctuates greatly. The overall lack of improvement is likely due to the fact that the input for the training is entirely

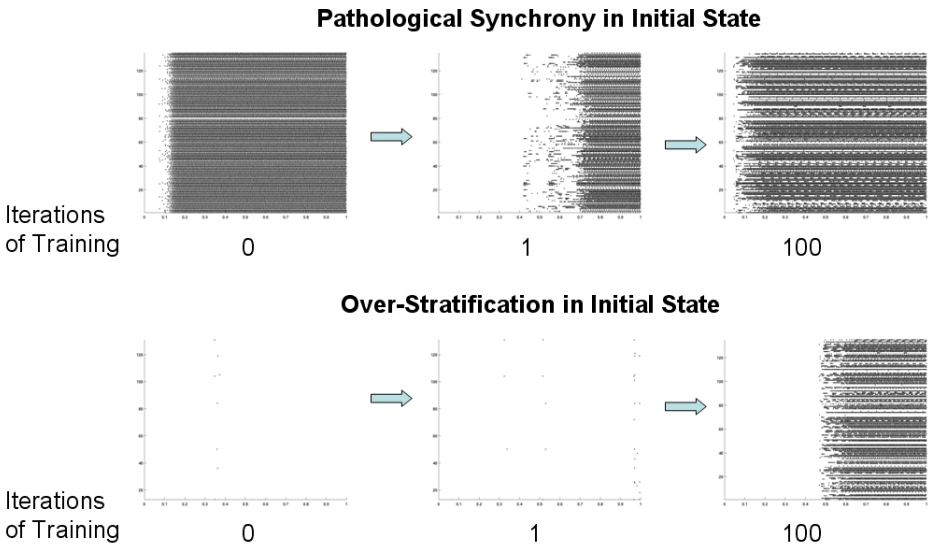


Figure 3.4: Spiking patterns in liquids trained with Hebbian learning. Top: The pathological synchrony state of the liquid is somewhat relieved by Hebbian learningthere are fewer neurons firing continuously and less dense patterns of firing. Bottom: Over-stratification is clearly relieved by iteration 100 through the Hebbian process.

random—the input is effectively noise. While it is clear that the effectiveness of the liquid is not lessened by this noise, there is no useful structure in the data.

In the experiments using random weight update training, after the initial increase in separation, we see a steady decline in separation, until it levels off close to zero. The change in spiking patterns indicates that the patterns become over-stratified (Figure 3.5) explaining the poor results. The primary benefit of these random weight update experiments is that through comparison, we can see that Hebbian Learning performs a role beyond random weight changing, even when confronted with nothing but noise.

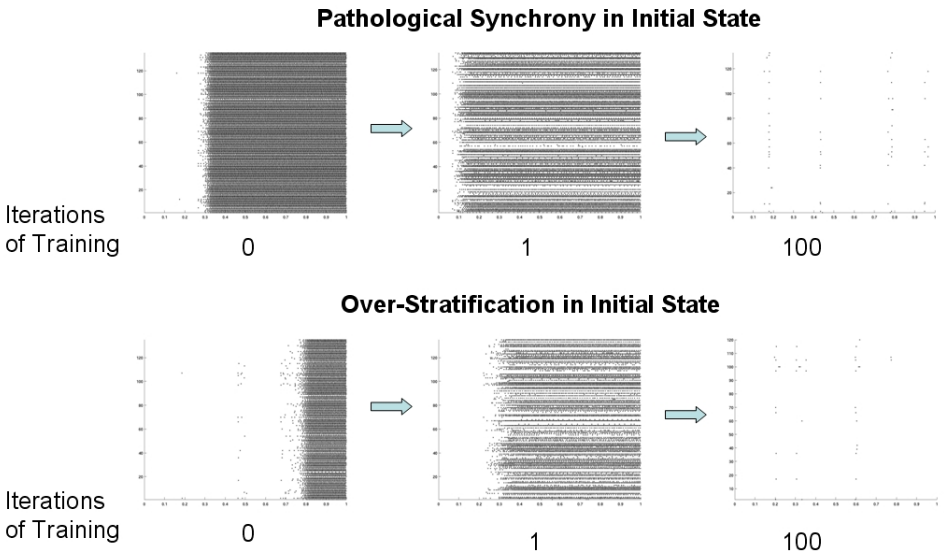


Figure 3.5: Spiking patterns in liquids trained with random weight updates. Random updates to synapse weights results in over-stratification over time regardless of the initial state of the liquid.

3.3 Effects of Spoken Digit Input on Separation

The second experiment in this chapter explores the effect of Hebbian learning on the separation of the liquid when exposed to real world data. The input for this experiment was a selection of 3519 training files from the TIDIGIT dataset [14]. These files consist of different people speaking single digits: one through nine, and zero and oh (both for 0).

3.3.1 Methods

To convert the sound files into spike trains, all silence was removed from the sound files, and they were converted into thirteen Mel frequency cepstral coefficients (mfcc) as is common in speech applications [5]. The frame size for the Fourier transform was 256 bytes with a frame step of 128 bytes. Thirteen input neurons were

used, one for each of the thirteen mfcc's. The firing rate of each of these neurons was determined with Equation 3.1 taken from [11].

$$Rate_i(t) = \frac{mfcc_i(t)}{(\Omega_i - \omega_i)} \cdot MaxRate \tag{3.1}$$

Here Ω represents the largest frequency for a given mfcc, ω represents the smallest frequency, and t is the time interval of a given firing rate, determined by the frame step.

The separation of the liquid was calculated before and after 1000 iterations of training on the TIDIGIT training dataset. The training dataset contained 3519 files, 1000 of which were randomly selected to train the liquid. To calculate separation, a set of state vectors, O, of size 100 was used as in the previous experiment. In this case each state vector of O was created by introducing one of 3519 randomly selected test files from the TIDIGIT testing dataset as input into the liquid. This test data was different from the training data but was prepared for the HLSM in the same fashion. Each file had a unique time interval, d. The state vector was measured at time d for each file with $\epsilon = 1.0ms$. In order to allow for an exhaustive permutation correlation test, the 100 test samples chosen before and after the training were identical.

3.3.2 Results

This experiment was run ten times on different liquids with the results indicated in Figure 3.6. The average improvement in separation for all ten trials was 0.064 and is statistically significant with a p-value of < 0.001. This p-value was calculated by finding the average difference in separation for every permutation of differences for the 10 trials. A single permutation consisted of swapping the order of the difference calculation (pre-training - post-training rather than post-training - pre-training) for a single trial. The number of permutations with averages greater than 0.064 was tabulated and divided by the total number of permutations, 1024, to

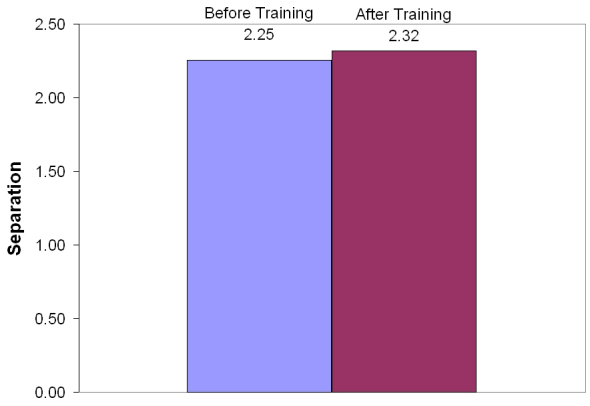
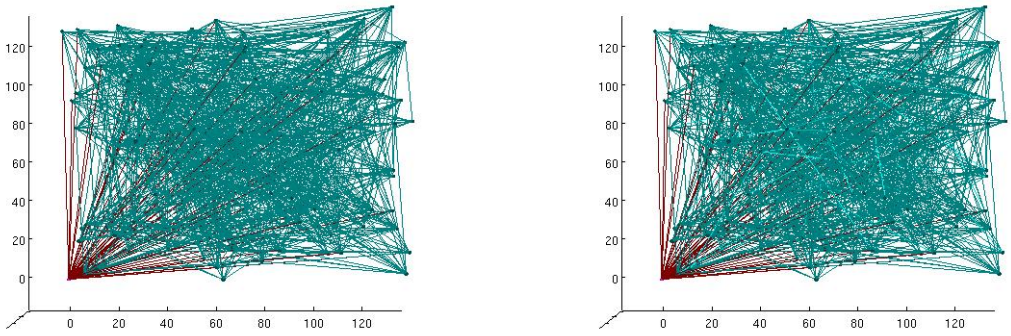


Figure 3.6: Average separation in liquid before and after training on the TIDIGIT dataset.

yield the given p-value. Unsupervised Hebbian learning can improve the separation of the liquid indicating a strong likelihood that the organization of the liquid has become a more effective component for learning.

3.3.3 Discussion

This experiment showed that Hebbian learning can improve the liquid given nonrandom input. Figure 3.7 shows how the physical structure of the liquid changes with non-random input from a speech recognition task. Notably, very little does change in comparison to the experiments with random input. The noticeable change is that a few connections are greatly strengthened. These figures were representative of all ten trials. More interesting was the improvement in separation noted after Hebbian learning took place, demonstrating that unsupervised learning can improve separation in complex neural microcircuits. The new question raised is whether or not the improvement comes at a lower cost than simply creating an effective liquid to begin with. The effectiveness of the original liquid is a product of all of the parameters used to create it, including mean weight, probability of inter-neuron connections, mean delay values, etc. Separation values for the pre-training liquids ranged from 2.09 to 2.70, a much greater difference than the average difference between pre- and



(a) Initial Liquid

(b) Liquid After Hebbian Learning

Figure 3.7: (a) shows the network connections of a liquid prior to training with Hebbian learning. The brighter the color of the connection, the stronger the weight of the synapse. (b) shows the network connections of a liquid after 1000 iterations of training on the TIDIGIT dataset using Hebbian learning. Note that there are several bright connections that were not present prior to training corresponding to synapses strengthened by Hebbian learning.

post-training separation. Also, the effectiveness of the Hebbian learning is sensitive to initial parameter settings (with different settings for the parameters used in the Hebbian learning, the post-training liquids actually resulted in lower separation). It is unclear whether discovering the ideal settings for Hebbian learning is less difficult then the effort required discovering the ideal settings for the initial liquid. It is also uncertain whether Hebbian learning or any other post-parameter setting adjustments provide a gain in separation that is not available in the parameter-setting stage.

Chapter 4

Reinforcement Learning

This chapter explores the use of reinforcement learning to change the architecture of the liquid. We refer to LSMs created with reinforcement learning as RLSMs. The reinforcement algorithm we use is derived from the OLPOMDP reinforcement learning algorithm [6]. However, since we are not interested in training the liquid in real-time, we have modified the algorithm with the goal of improving the liquid's separation.

4.1 Algorithm

Because we want the liquid to be able to take on any form, we initialize each neuron with synaptic connections to every other neuron, including the input neurons. The permanent delay and initial weight for each synapse is set by sampling from a normal distribution, and the weight can be positive (excitatory) or negative (inhibitory) with equal probability.

The premise of the reinforcement learning algorithm is to update every synapse's weight according to a reward function and an eligibility value attributed to each synapse:

$$w_{ij}(t + \Delta t) = w_{ij}(t) + \lambda r(t + \Delta t) z_{ij}(t + \Delta t) \tag{4.1}$$

Here λ is the learning rate, w and z are respectively the weight and eligibility of the synapse connecting neuron j to neuron i, and $r(t+\Delta t)$ is the reward at the new time

step. The reward is calculated as a sigmoid function of separation:

$$r(t+\Delta t) = \frac{1}{1+e^{\gamma(\frac{Sep_{\Psi}(O(t+\Delta t))}{Sep^{*}_{\Psi}})}} \tag{4.2}$$

Here γ is the gain of the sigmoid and Sep^{*}_{Ψ} is the optimal separation value for the liquid. We approximate Sep^{*}_{Ψ} by calculating the separation, using Equation 2.9, for 1000 artificially created sets of state vectors where each set consists of one state vector per class, and then selecting the highest separation value. Each of these artificial sets of state vectors is created by generating one state vector at a time that is as different as possible from all previously generated state vectors. This process is partially random since multiple possibilities exist for each new state vector.

The eligibility of each synapse is initialized to zero. The eligibility is changed based on a function of the firing behavior and firing likelihood of the post-synaptic neuron (Equation 4.4). An increment in eligibility occurs when the post-synaptic neuron fires at time step t and is proportional to the likelihood that the post-synaptic neuron will not fire. A decrement in eligibility occurs when the post-synaptic neuron does not fire at time step t and is proportional to the likelihood that the post-synaptic neuron will fire. The new eligibility of a synapse is calculated with the following equation:

$$z_{ij}(t+\Delta t) = \beta z_{ij}(t) + \sum_{o_k \in O} \frac{\zeta_{ij}(t, o_k)}{|O|} \tag{4.3}$$

where β is a discount factor. As time progresses, the eligibility drops according to β. $\zeta_{ij}(t, o_k)$ is the change in eligibility at time t as a function of each state vector, o_k. $|O|$ is the cardinality of the set of all state vectors collected at t. The summation in Equation 4.3 averages the change in eligibility for all o_k generated to calculate

$Sep_\Psi(O(t))$. A series of equations for calculating ζ_{ij} follows:

$$\zeta_{ij}(t, o_k) = \begin{cases} 1 - \pi_i(t), & \text{if } f_i(t, o_k) = 1 \\ -\pi_i(t), & \text{if } f_i(t, o_k) = 0 \end{cases} \tag{4.4}$$

$$\pi_i(t) = \frac{\rho_i(t)}{\sum_{k \in Q} \rho_k(t)} \tag{4.5}$$

$$\rho_i(t) = \sum_{j \in Z_i} w_{ij} - \min_{j \in Q} \sum_{k \in Z_i} w_{jk} \tag{4.6}$$

Here $\pi_i(t)$ is the likelihood that neuron i will fire at t and $f_i(t, o_k)$ is an indicator function that returns 1 if neuron i fires at t and 0 otherwise. ρ is a function used to simplify the likelihood equation. The likelihood of firing is not a probabilistic calculation but rather an estimate based on the combined weight of incoming synapses. This estimate is used in the interest of speeding up an already very time consuming algorithm. Also, the stochastic nature of liquids does not require a precise calculation of all parameters. In Equations 4.5 and 4.6, Q is the set of all neurons in the liquid. Finally, Z_i is the set of neurons directly upstream of neuron i.

4.2 Results

The first experiments using the reinforcement algorithm applied artificial input to small liquids. Also, as a simplification reinforcement learning was only applied to the interconnecting synapses of the liquid and not the synapses connecting input neurons to the liquid. Our artificial input was a series of spikes occurring at a fixed frequency and then jittered by uniform random noise. Different classes of input were defined by different signature frequencies obtained from Equation 4.7, where f is the frequency in spikes per second, c is the class number, and the value 3.125 is chosen empirically.

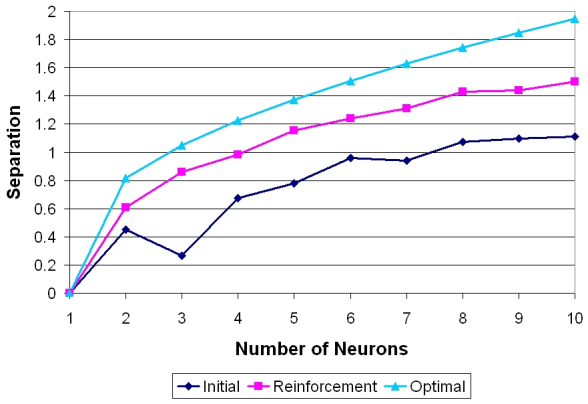


Figure 4.1: Average separation in liquids of varying size before and after reinforcement learning with an artificial problem. Data points are an average of ninety trials spanning problems with two through ten classes (ten trials of each class). Optimal separation included for comparison.

$$f = 3.125 \times 2^c \tag{4.7}$$

Since each class' signature frequency is at least a power of two different from any other class, the classes are highly differentiated (up to a certain number of classes when the frequency becomes so high that the liquid can't make distinctions). With this simple artificial problem on small networks, we are able to see how the reinforcement algorithm works under controlled circumstances. Figure 4.1 shows the effects of reinforcement learning on small liquids of varying numbers of neurons. Each point represents the mean separation (over ten trials) of problems with varying numbers of classes. The values labeled as "initial" are the separation of the liquid before reinforcement learning has occurred, while the values labeled as "reinforcement" are the maximum separation obtained out of 500 iterations of reinforcement learning. Finally, the values labeled as "optimal" are the average (over number of classes) maximum separation that a liquid of a given number of neurons can have.

Figure 4.2 looks at how the number of neurons affects the accuracy of the readout for binary classification. As in Figure 4.1, it shows the accuracy of an LSM

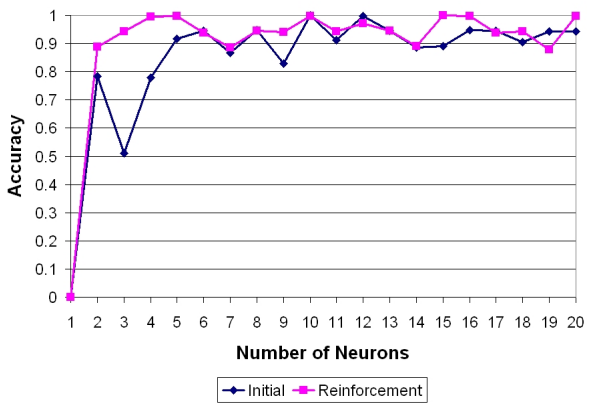


Figure 4.2: Average accuracy in liquids of varying size before and after reinforcement learning with an artificial problem of only two classes. Data points are an average of ten trials.

before and after the liquid has been modified with reinforcement learning. The "reinforcement" label refers to the liquid with the highest separation obtained from the training period. The accuracy indicates the average accuracy of a readout function consisting of two perceptrons (one for each class) trained with state vectors obtained from the liquid and is an average of ten experimental trials. The x-axis indicates the number of neurons in the liquid.

Figure 4.3 also shows the effect of reinforcement learning on accuracy but explores the number of classes in the problems presented to the algorithm rather than the number of neurons. Here the accuracy indicates the average accuracy of a readout function composed of n perceptrons (where n is the number of classes) trained with state vectors obtained from the liquid and is an average of ten experimental trials. The x-axis indicates the number of classes used for each trial. A liquid of ten neurons was used for each trial.

4.3 Discussion

It is clear from Figure 4.1 that reinforcement learning improves the separation property of the liquid in the given problem. The fact that separation is positively

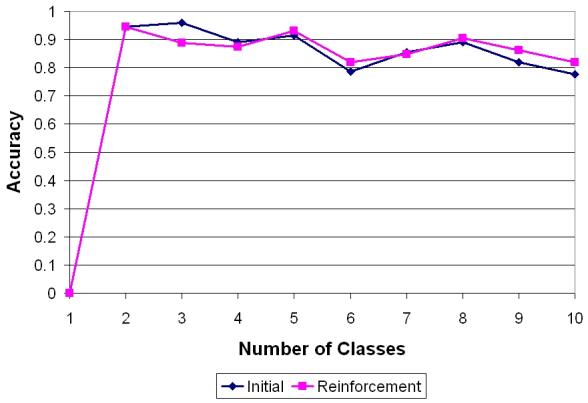


Figure 4.3: Average accuracy of readout in liquids for problems with varying numbers of classes before and after reinforcement learning with an artificial problem. Data points are an average of ten trials each with a liquid of ten neurons

correlated with the number of neurons in the liquid is not surprising—as the number of neurons increases the maximum difference between two or more state vectors also increases as indicated by the optimal separation line. However, this improvement in separation does not carry over to an appreciable improvement in accuracy as demonstrated by Figures 4.2 and 4.3. While this causes us to question the effectiveness of our separation metric, it may be that the problem is too simple to demonstrate an accurate correlation between separation and accuracy. The results of Chapter 2 clearly illustrate that a correlation between accuracy and separation does exist. Despite the marginal increase in accuracy afforded by RLSMs, the fact that there is an overall increase at all reveals potential for RLSMs. Unfortunately, the creation of RLSMs is much slower than even HLSMs. This has discouraged further investigation of the algorithm. In chapter 5 we will introduce an algorithm that borrows principals of reinforcement learning and shows significant improvement to separation and accuracy while strengthening their correlation.

Chapter 5

Separation Driven Synaptic Modification

Separation Driven Synaptic Modification or SDSM is an approach used to modify the synapses of the liquid by using the separation metric defined in Chapter 2. For convenience we will reprint the definition of separation here:

$$Sep_{\Psi}(O(t)) = \frac{C_d(t)}{C_v(t) + 1} \tag{5.1}$$

$$C_d(t) = \sum_{m=1}^{N} \sum_{n=1}^{N} \frac{\|\mu(O_m(t)) - \mu(O_n(t))\|_2}{N^2} \tag{5.2}$$

$$C_v(t) = \frac{1}{N} \sum_{m=1}^{N} \rho(O_m(t)) \tag{5.3}$$

Recall that $C_d(t)$ is the mean distance between the center of mass for every pair of classes and is referred to as the inter-class distance. $C_v(t)$ is the mean variance of each class and is referred to as the intra-class variance.

Now we will look at the actual synaptic modification equation for SDSM in Equation 5.4.

$$w_{ij}(t + \Delta t) = sgn(w_{ij}(t))(|w_{ij}(t)| + E(t)\lambda F(t)) \tag{5.4}$$

Here $w_{ij}(t)$ is the weight of the synapse from neuron j to neuron i at time t, λ is the learning rate, $sgn(w_{ij}(t))$ is the sign of $w_{ij}(t)$, $E(t)$ is a function of the effect of

separation on the weight at time t, and $F(t)$ is a function of the firing behavior of all neurons in the liquid at time t.

First we will look at the function $E(t)$. To explain this function and its derivation it is first important to understand what we mean by relative synaptic strength, R_s, defined by Equation 5.5.

$$R_s = \frac{|w_{ij}(t)| - \mu_w}{M_w} \tag{5.5}$$

Here μ_w estimates the expected value of the magnitude of synaptic weights in the initial liquid. M_w estimates the maximum value of random variables drawn from the same distribution used to generate synaptic weights in the initial liquid. (These approximations were obtained via simulation with 10,000 samples). M_w essentially normalizes the synaptic strength while μ_w is used to differentiate weak synapses and strong synapses. A negative R_s is considered weak while a positive R_s is considered strong.

Too little distance between centers of mass, $C_d(t)$ (Equation 5.2), or too much variance within classes, $C_v(t)$ (Equation 5.3), can decrease separation and thus the overall effectiveness of the liquid. Generally speaking, if there is too little distance between centers of mass, it is because strong synapses are driving the liquid to behave a particular way regardless of input class. To rectify this, we want to strengthen weak synapses and weaken strong synapses. This will drive the liquid towards a more chaotic structure that will yield results more dependent on the input. On the other hand, if there is too much variance within classes, it is because the liquid is too chaotic to drive inputs of the same class to behave similarly. To relieve this problem, it is necessary to strengthen strong synapses and weaken weak synapses even more. This will polarize the liquid, requiring greater differences in input to cause a change in the liquid's behavior (in other words, the liquid will be less chaotic).

The motivation behind the function $E(t)$ is balancing these two solutions at the level of an individual synapse. The first solution, solving the problem of differentiating classes of input, d_i, is implemented with Equation 5.6.

$$d_i = \alpha_i \left(1 - \frac{C_d}{Sep^*_\Psi}\right) \tag{5.6}$$

$$\alpha_i = \frac{\sum_{k=1}^{N} \mu_i(O_k(t))}{N} \tag{5.7}$$

Here α_i is the activity of a specific neuron i (the post-synaptic neuron of synapse w_{ij}) and is defined by Equation 5.7. α_i contains $\mu(O_k(t))$ which is the mean of the state vectors in class k. Specifically, $\mu_i(O_k(t))$ is the value of the i^{th} element of the mean state vector. This is also the fraction of state vectors belonging to class k in which neuron i fires. In Equation 5.6, the normalized value of $C_d(t)$ is subtracted from one so that d_i will provide greater correction for smaller values of $C_d(t)$. Essentially what Equation 5.6 does is to multiply the activity of a particular neuron by the amount of correction necessary for too little distance between class centers of mass. We assume that neuron activity is to blame for this problem. This may or may not be the case; however, consistently assuming correlation between $C_d(t)$ and neuron activity should eventually impose this correlation on the liquid and ultimately yield the desired results.

The solution to the second problem (too much variance within classes), is implemented with Equation 5.8.

$$v_i = \frac{\sum_{k=1}^{N} \mu_i(O_k(t))\rho(O_k(t))}{N} \tag{5.8}$$

v_i is calculated similarly to α_i except that each instance of $\mu_i(O_k(t))$ is multiplied by the mean variance for class k, because mean variance is determined class by class. The end result is that Equation 5.8 provides greater correction for larger values of

$C_v(t)$ which is desirable since we are trying to reduce intra-class variance. Like the equation for d_i, the equation for v_i assumes a correlation between the neuron's activity and $C_v(t)$.

The function $E(t)$ is derived from the Equations 5.5-5.8 as follows:

$$E(t) = R_s \left(v_i - d_i\right) \tag{5.9}$$

Here d_i is subtracted from v_i because, as mentioned previously, we want the distance correction, d_i, to strengthen weak synapses and weaken strong synapses while we want the variance correction, v_i to strengthen strong synapses and weaken weak synapses. In other words, we want d_i to increase the chaotic nature of the liquid and v_i to decrease the chaotic nature of the liquid. Ultimately the goal of Equation 5.9 is to find a balance between a liquid that is too chaotic and one that is too stable [4]. Equation 5.10 shows Equation 5.9 fully expanded by substituting it with Equations 5.2, 5.3, and 5.5-5.8 to show the full process of evaluating $E(t)$.

$$E(t)=R_s\left(\frac{\sum_{k=1}^{N}\mu_i(O_k(t))\rho(O_k(t))}{N}-\frac{\left(\sum_{k=1}^{N}\mu_i(O_k(t))\right)\left(1-\frac{\sum_{m=1}^{N}\sum_{n=1}^{N}\|\mu(O_m(t))-\mu(O_n(t))\|_2}{N^2 Sep_{\Psi}^{*}}\right)}{N}\right) \tag{5.10}$$

We now turn our attention to $F(t)$, the function of the firing behavior of all neurons in the liquid at time t. The function is expressed in three parts as follows:

$$F(t) = \begin{cases} \frac{1}{\phi(t)}, & \text{if } w_{ij}(t)E(t) \geq 0 \\ \phi(t), & \text{if } w_{ij}(t)E(t) < 0 \end{cases} \tag{5.11}$$

$$\phi(t) = 2^{kA(t)-b} \tag{5.12}$$

$$A(t) = \frac{\sum_{o \in O(t)} \frac{\sum_{\eta \in o} \eta}{|o|}}{|O|} \tag{5.13}$$

Here $A(t)$ is the activity of the entire liquid at time t and is calculated by finding the average fraction of neurons, η, that fire in each state vector in $O(t)$. $\phi(t)$ is a transformation of $A(t)$ that reduces it to a function that will allow $F(t)$ to work as a simple multiplication of $E(t)$ in Equation 5.4. $\phi(t)$ contains two variables, k and b, that represent, respectively, the scale and offset of the transformation. For our experiments, $k = 6$ and $b = 3$ were found, through preliminary experiments, to yield the highest separation values. $F(t)$ uses the state of the synapse and the results of $E(t)$ to determine how the global activity of the liquid at time t will effect the change in weight. The effect of $F(t)$ is to promote the overall strengthening of excitatory synapses while promoting the overall weakening of inhibitory synapses if less than half of the neurons in the liquid fire. If more than half of the neurons fire, the effect of $F(t)$ is reversed. The goal of $F(t)$ is to direct the liquid to a "useful" amount of activity. This assumes that half of the neurons firing for all state vectors is the desired fraction of activity to achieve the maximum separation possible.

Chapter 6

Applying SDSM

Two artificial problems were developed to test the effect of Separation Driven Synaptic Modification (SDSM), defined in Chapter 5, on the separation of a liquid and ultimately the accuracy of a LSM. This chapter explores the results of these experiments and shows the potential of this algorithm to select ideal liquids for LSMs. Not only does SDSM select functional liquids for a given problem, but we show that these liquids generalize to other problems.

6.1 Definition of Artificial Problems

The first problem is the simpler of the two, and we call it the *frequency recognition* problem. This problem has four input neurons and five classes. Each input neuron fires at a *slow* or *fast* frequency. The five classes are defined by specific combinations of fast and slow input neurons as shown Table 6.1, where 1 represents a fast input neuron and 0 a slow one. These particular patterns were chosen to challenge the liquid with a variety of combinations as well as the task of ignoring one channel (input neuron 4).

Individual samples from each class are generated by following the above template and then jittering the frequencies. Since each class is distinctly defined by a particular pattern of behavior on a neuron-by-neuron basis, this is a fairly simple problem. It does however test the liquid with multiple input neurons (channels of

	Input 1	Input 2	Input 3	Input 4
Class 1	1	0	0	0
Class 2	0	1	0	0
Class 3	1	1	0	0
Class 4	0	0	1	0
Class 5	1	0	1	0

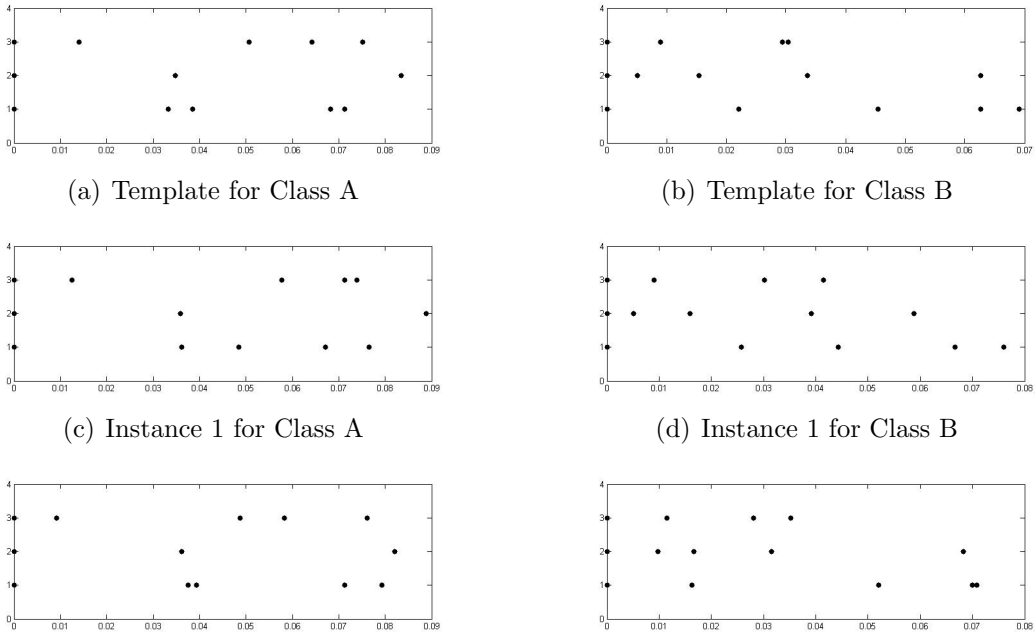
Table 6.1: Frequency patterns for each class in the frequency recognition problem. Each input represents one of four input neurons. A 1 indicates a fast spiking frequency while a 0 represents a slower spiking frequency.

input), something that artificial problems mentioned earlier (Chapters 3 and 4) did not do.

The second problem is more general and complex. We call it the *pattern recognition* problem. This problem has eight input neurons and a variable number of classes. Each class is based on a template spike pattern randomly created for each input neuron. The random pattern is generated by plotting individual spikes with a random distance between one another. This distance is drawn from the absolute value of a normal distribution with a mean of $10ms$ and a standard deviation of $20ms$. Once the template pattern for each input neuron in a class is created, individual instances from the class are created by jittering each spike in the template. The spikes are jittered by an amount drawn from a zero-mean normal distribution with a standard deviation of $5ms$ making the problem particularly difficult. All of these values were determined empirically to create a solvable but difficult problem. A simplified example with only two classes and three input neurons is shown in Figure 6.1.

6.2 Parameter Settings

Extensive preliminary experiments dictated the choices for each of the many parameters of the liquid used in both the frequency and pattern recognition problem. The different parameters we looked at were the number of neurons, connection



(a) Template for Class A

(b) Template for Class B

(c) Instance 1 for Class A

(d) Instance 1 for Class B

(e) Instance 2 for Class A

(f) Instance 2 for Class B

Figure 6.1: The templates for two classes, A and B, are shown in (a) and (b) respectively. Each class has three input neurons designated by the y-axis. The x-axis is time spanning 100ms. (c) and (e) show examples of instances from class A created from jittered versions of the template. (d) and (f) shown examples of instances from class B.

probability, the mean synaptic weight and delay, the standard deviation of the synaptic weight and delay, the number of samples per class used to determine separation at each instance, the number of iterations to run, the learning rate, the decay time constant, and the amount of noise present in each neuron. Table 6.2 shows the parameters we settled on for all of the results presented in this chapter as well as Chapters 7 and 8. As the results of this chapter and the next show, these parameters generalize very well as long as the temporal scale of the input is on the order of one second.

Some of the parameters presented in Table 6.2 require further explanation. The connection probability is the probability that any given neuron (including input neurons) is connected to any other liquid neuron (liquid neurons cannot connect back to input neurons). The value for the connection probability indicated in Table 6.2

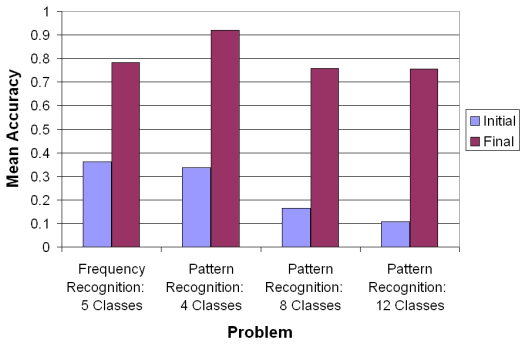
Neurons	64
Connection Probability	0.3
Synaptic Weight Mean	$2 \cdot 10^{-8}$
Synaptic Weight SD	$4 \cdot 10^{-8}$
Samples per Class	3
Training Iterations	500
λ	$5 \cdot 10^{-10}$
Inoise	$5 \cdot 10^{-8}$
τ	0.003
Synaptic Delay Mean	0.01
Synaptic Delay SD	0.1

Table 6.2: Parameters used by SDSM for Artificial and Phonetic problems.

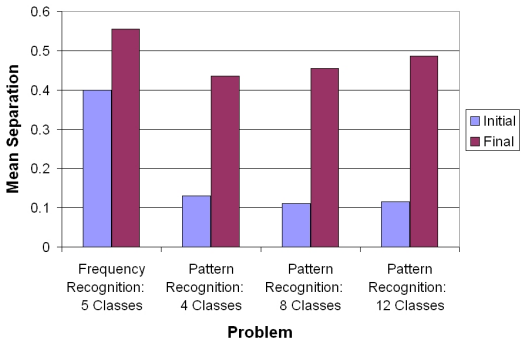
means that each neuron is connected to roughly one third of the other neurons. The "samples per class" parameter refers to the number of training samples used from each class in the calculation of separation. This in turn is what drives the SDSM algorithm. The more samples used, the more accurate the separation calculation will be, but at the cost of speed. The number of iterations is simply how long to run the SDSM algorithm. By 500 iterations, most liquids had reached a plateau in separation improvement. λ is the learning rate first shown in Chapter 5. τ is the decay time constant which refers to the rate at which the membrane potential of each synapse decays. Inoise is the amount of noise produced by each neuron and is necessary for an efficient liquid [13].

6.3 Empirical Results

Using the established parameters, we created LSMs with SDSM for both the pattern and the frequency recognition problems (explained above). For the pattern recognition problem we explored 4-, 8-, and 12-class problems. We only explored the specifically defined 5-class scenario for the frequency recognition problem. For each problem we ran fifty experiments each with a unique randomly generated initial liquid. State vectors obtained from both the initial liquid and the liquid after



(a) Mean Accuracy



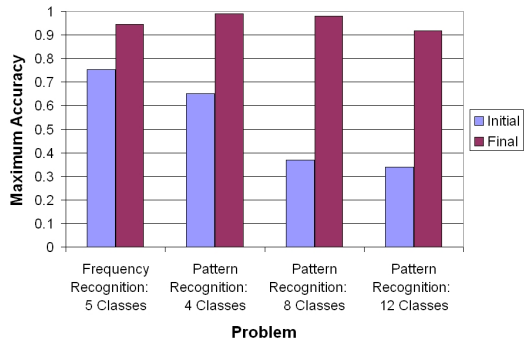
(b) Mean Separation

Figure 6.2: A comparison of traditional liquids (initial) and those shaped by SDSM (final) across four problems. Results are the mean accuracy (a) and separation (b) of fifty LSMs.

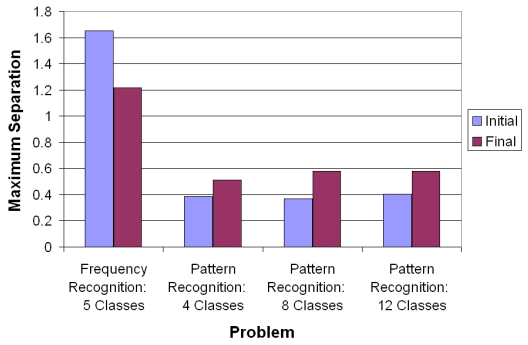
five hundred iterations of SDSM were used as input to multiple perceptrons. Each perceptron was trained to classify members of one particular class, so there were N binary classifiers. The output of each perceptron was then compared, assigning the class of the perceptron with the greatest confidence to the state vector in question. This readout function was used because it is very simple, thus allowing us to more carefully scrutinize the quality of the liquid. For all of our experiments, the test size was one hundred samples per class.

Figure 6.2(a) shows the mean accuracy (over all fifty experiments) of the LSM for each problem. Additionally Figure 6.3(a) shows the best accuracy obtained out of the fifty experiments for each problem. In practice, the liquid with the maximum accuracy is the one that we would select for future use. However, since we are interested in the potential of SDSM to enable the creation of a single liquid, we are also interested in the mean accuracy. Keep in mind that the initial liquid is the typical LSM scenario: a strictly randomly generated liquid. So both of these figures are a comparison of standard LSMs to LSMs shaped with SDSM.

In addition to the accuracy of the LSMs, Figures 6.2 and 6.3 show the separation of the liquids. It should be noted that the liquid with the maximum separation



(a) Maxiumum Accuracy



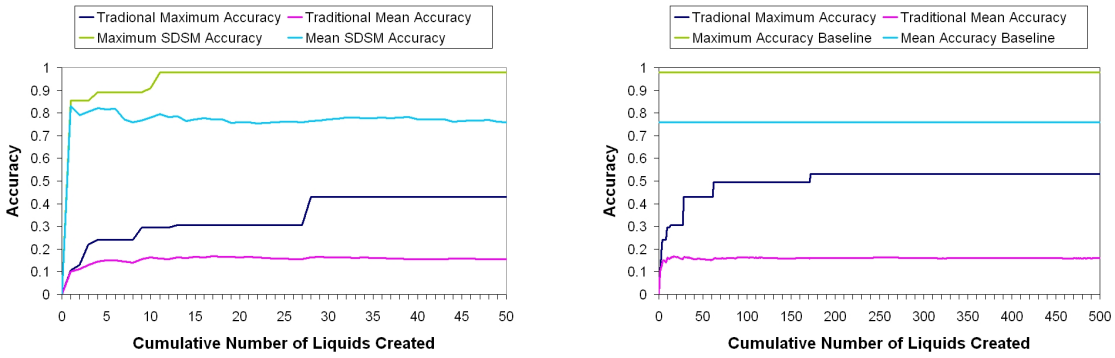
(b) Maximum Separation

Figure 6.3: A comparison of traditional liquids (initial) and those shaped by SDSM (final) across four problems. Results are the best accuracy (a) and separation (b) obtained out of fifty LSMs.

is not necessarily the same liquid that accounts for the maximum accuracy. Overall these results support the correlation between separation and accuracy, though there is an anomaly in Figure 6.3(b) where a higher separation occurred in an initial random liquid than in any SDSM shaped liquid. Such anomalies are bound to occur occasionally due to the random nature of LSMs. The mean results in Figure 6.2(b) confirm the abnormal nature of this event.

These results show a substantial increase in the performance of liquids using SDSM, particularly in the problems with more classes. Traditional liquids may see an improvement with different parameter settings, however these were the best parameter settings for initial liquids that we found. The improvement with SDSM is so substantial that it is unlikely that different liquid parameter settings would prove a sufficient alternative.

Figure 6.4 compares the trends in liquid exploration for both traditional and SDSM generated liquids. The results in this figure were obtained from the pattern recognition problem with eight classes. Figure 6.4(a) shows the accuracy of the best liquid obtained so far, given an increasing number of total liquids created (the mean accuracy is also shown). The figure demonstrates how far fewer liquids are required



(a) Accuracy Over 50 Liquid Creations

(b) Accuracy Over 500 Liquid Creations

Figure 6.4: A comparison of accuracy trends in traditional liquids and those generated with SDSM. These results show how the accuracy of the best liquid obtained so far improves as more liquids are created. The results also show how the mean accuracy stabilizes as more liquids are created. (a) shows the trend up to fifty liquid creations. (b) shows the trend up to 500 liquid creations for traditional liquids only (the accuracy for SDSM generated liquids is shown as a baseline only).

to obtained a satisfactory liquid using SDSM when compared with the traditional method. Figure 6.4(b) extends the graph further for traditional liquid creation, and compares the results to the best results obtained from SDSM after only fifty liquid creations. We see from this figure that even after 500 liquid creations, a liquid has not been created by the traditional means that can compete with SDSM after only eleven liquid creations. This figure demonstrates that not only can SDSM find a suitable liquid quicker than traditional methods, but it can also potentially create a liquid that obtains higher accuracy than what is statistically possible with conventional LSMs.

Figure 6.5 shows how separation changes with each iteration over the history of a typical SDSM trial. Figure 6.6 shows the mean value of separation over fifty trials of SDSM. These results show that separation is clearly improving over time and that the SDSM algorithm is doing what we want it to do. This in fact further strengthens the idea that separation correlates with accuracy since the final result is a significant improvement in accuracy. It is important to keep in mind that separation in both of these figures was calculated using only three samples per class for each iteration

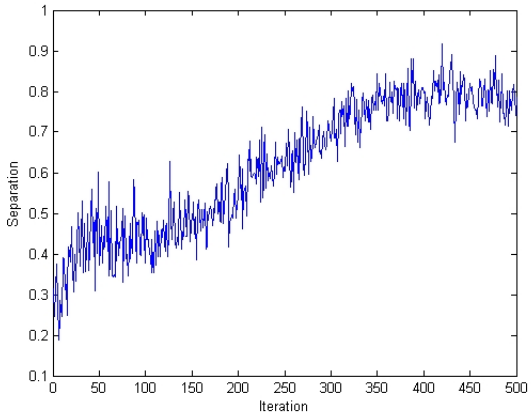
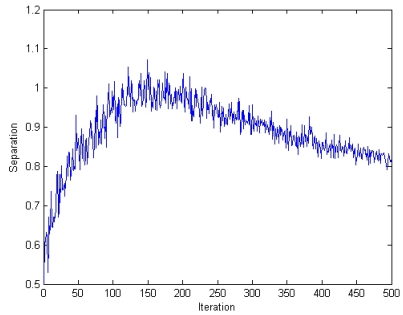


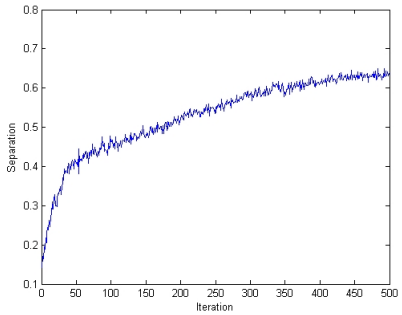
Figure 6.5: The separation of a liquid at each iteration during training of SDSM. This is a single representative example out of two hundred different liquids created in this set of experiments. This particular liquid was created using the pattern recognition problem with eight classes.

and is thus a very rough estimate. The initial and final separation values indicated in Figures 6.2(b) and 6.3(b) are much more accurate approximations of separation using one hundred samples per class.

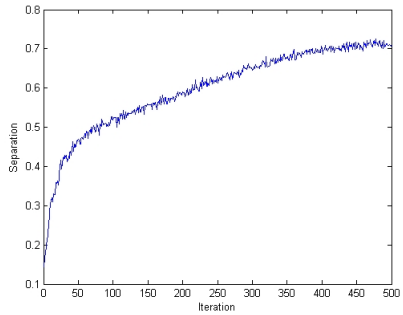
Though the correlation between accuracy and separation is not perfect, it is satisfactory as a metric for both evaluating the quality of a liquid and revealing beneficial changes that can be made to a liquid. As the algorithm in Chapter 5 reveals, the relationship between separation and synaptic modification within the SDSM algorithm is complicated and can't necessarily be predicted. Some flaws of the separation metric become apparent when studying Figure 6.3(b). While the accuracy for the frequency recognition problem and the pattern recognition problems are similar, the best liquid separation is distinctly higher in liquids created with the frequency recognition problem. This demonstrates that different problems will yield different separation values due to their unique properties. In this particular case the disparity is probably at least partly a result of the differing number of input neurons present in the different problems. The pattern recognition problems all have twice as many input neurons as the frequency recognition problem. This characteristic of the



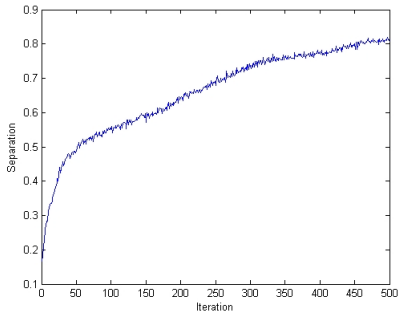
(a) Frequency Recognition: 5 classes



(b) Pattern Recognition: 4 classes



(c) Pattern Recognition: 8 classes



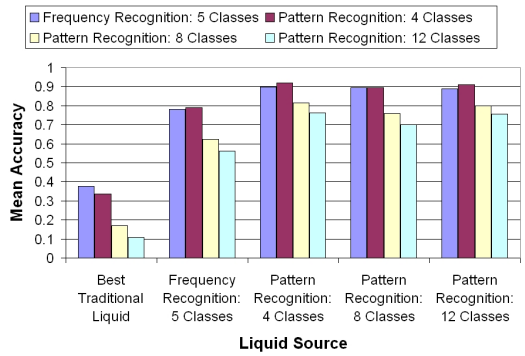
(d) Pattern Recognition: 12 classes

Figure 6.6: The mean separation history using SDSM for the four problems explored in this chapter. Each history is an average if fifty trials.

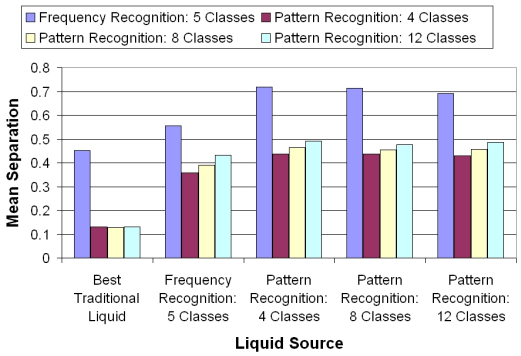
separation metric does not necessarily call for a change in the metric. It does mean however, that separation values between different problems cannot be adequately compared to one another.

6.4 SDSM Generalization on Artificial Problems

One of the potential strengths of LSMs is the ability of the liquid to generalize across a variety problems. This is an important property since useful liquids can be difficult to find. Because SDSM essentially uses training data to create the liquid, there is a concern that the liquid's ability to generalize may be compromised. In order to test the generalizing capability of the SDSM generated liquids, we ran each of the problems addressed in Figures 6.2 and 6.3 on every liquid used to generate



(a) Mean Accuracy



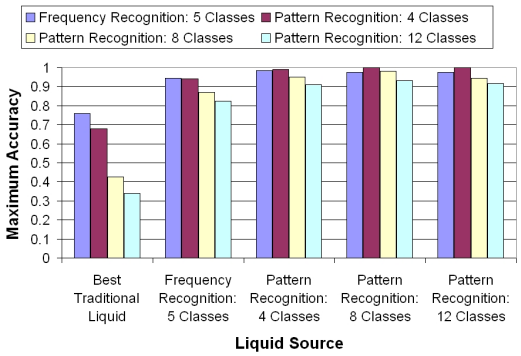
(b) Mean Separation

Figure 6.7: The performance of liquids created using SDSM with four different liquid sources, across four different problems. The best traditional liquid is the best mean result obtained from fifty liquids randomly generated for one of the four problems. The figure shows the mean result of fifty unique liquids per data point.

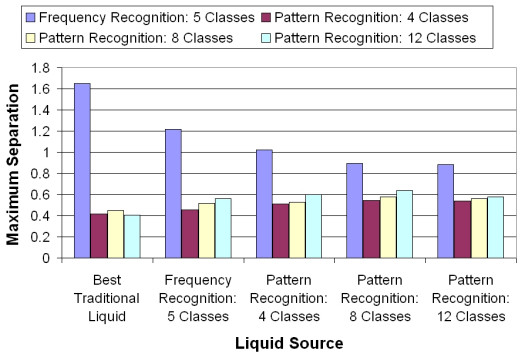
those figures. The results are shown in Figures 6.7 and 6.8. When discussing the problem used to create a specific liquid, we refer to the problem as the *liquid source*.

Input neurons are considered part of the liquid, thus their synapses are modified as part of SDSM. When a liquid is created from a source, it has I input neurons, where I is the number of spike trains present in the source's input. Because different problems or sources have varying numbers of spike trains, discrepancies between the number of spike trains and number of input neurons must be resolved.

When running a problem on a liquid that differs from the source, we use the following approach. If the new problem's input is encoded in fewer spike trains than the source problem, then the spike trains are mapped arbitrarily to a subset of the input neurons. The excess input neurons receive a null signal as input. If the new problem's input is encoded in more spike trains than the source problem, then multiple spike trains get mapped arbitrarily to individual input neurons. The spiking patterns are combined, increasing the total number of spikes firing in each input neuron. For each experiment, the same arbitrary mapping is used for every input to maintain consistency.



(a) Max Accuracy

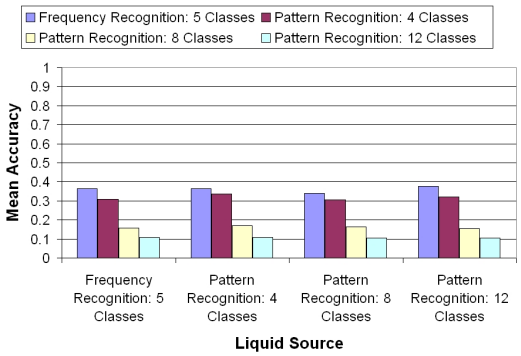


(b) Max Separation

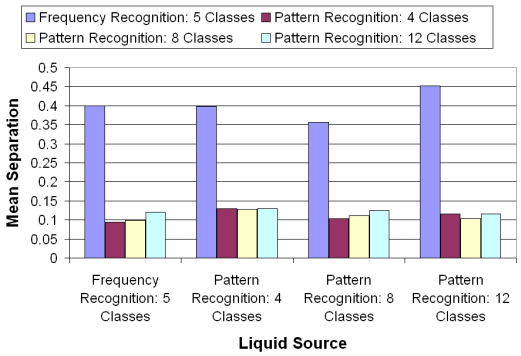
Figure 6.8: The performance of liquids created using SDSM with four different liquid sources, across four different problems. The best traditional liquid is the best maximum result out of fifty liquids randomly generated for one of the four problems. The figure shows the maximum result out of fifty unique liquids per data point.

The results shown in Figures 6.7 and 6.8 were obtained using two types of problems. All of the pattern recognition problems use eight spike trains for each instance while the frequency recognition problem only uses four spike trains (see Section 6.1). When a pattern recognition problem is used as the source for the frequency recognition problem, four of the input neurons have no signal. When the frequency recognition problem is used as the source for a pattern recognition problem, each input neuron combines the signals of two spike trains. Figures 6.9 and 6.10 show the results of each problem on the initial liquid randomly generated for the indicated problem. Since all of the pattern recognition problems have the same number of spike trains, and since the initial liquids are untrained, the three data points showing the results for these initial liquids are actually redundant. However, they are included for completeness.

In addition to showing the results of each problem on the four liquid sources, Figures 6.7 and Figures 6.8 also show the results of each problem on the best traditional liquid. The best traditional liquid is the best result obtained from Figures 6.9



(a) Mean Accuracy



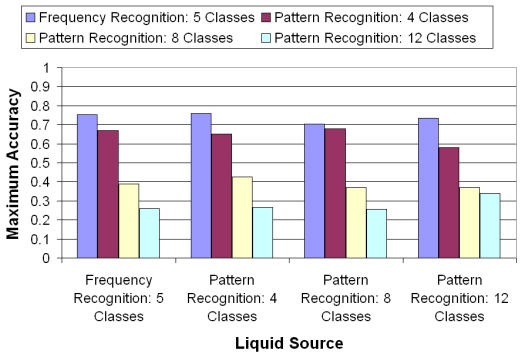
(b) Mean Separation

Figure 6.9: The performance of randomly generated liquids created for the four different problems (referred to as the liquid source for convenience). The figure shows the mean result of fifty unique liquids per data point.

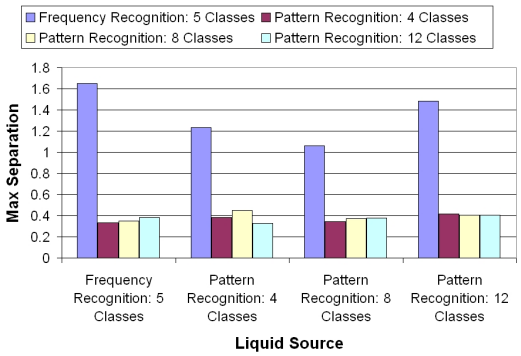
and 6.10 respectively. The best results from the initial random liquids were used as comparison to exemplify the generalizing ability of SDSM.

These results demonstrate the ability of liquids to generalize to different problems. Figures 6.9 and 6.10 emphasize this by their distinct lack of variation in behavior between liquids generated for frequency recognition and pattern recognition problems. One would expect a difference in behavior between these two different liquid states since liquids created for frequency recognition only have four input neurons while liquids created for pattern recognition problems have eight. The fact that the results show no significant difference indicates that the liquid is indeed acting as a universal temporal filter.

SDSM essentially uses training data to create new liquids from those randomly generated for Figures 6.7 and 6.8. Since the new liquids that are created depend upon the problem used to train them, one would expect that the ability of the liquid to generalize will be compromised. Interestingly, the results shown in Figures 6.7 and Figures 6.8 clearly demonstrate that this is not the case. In fact, liquids not created with the frequency recognition problem performed better on the problem than liquids actually created with the frequency recognition problem. However, liquids cre-



(a) Max Accuracy



(b) Max Separation

Figure 6.10: The performance of randomly generated liquids created for the four different problems (referred to as the liquid source for convenience). The figure shows the maximum result out of fifty unique liquids per data point.

ated with pattern recognition problems did perform better on those problems than liquids generated with the frequency recognition problem. In both cases, SDSM still performed significantly better than traditional LSMs. The fact that liquids created with pattern recognition performed better on both problems indicates that the problem used to create the liquid can make a difference. Looking at Figure 6.8(a) we see that all liquids created with pattern recognition problems found liquids that performed with over 90% accuracy on all of the problems. Pattern recognition is clearly the more complicated of the two problems; and, by using SDSM with the more complicated problem, the liquid may be primed for overall better performance. It should be noted that both problem types are very similar in that classes are determined by specific overall spiking patterns. Very distinct types of problems may not share this performance correlation. Future research should explore diverse problem types to more rigorously evaluate the ability of SDSM to generalize.

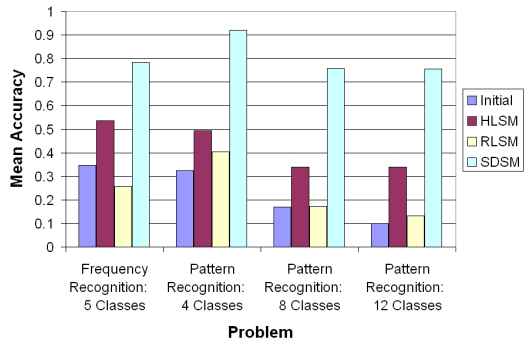
Chapter 7

A Comparison of SDSM, RLSMs, and HLSMs

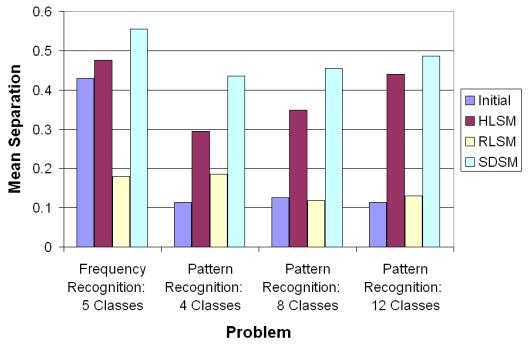
Three new methods for creating liquids have been described in previous chapters. In Chapter 3 we introduced Hebbian learning and referred to LSMs created with Hebbian learning as HLSMs. We performed preliminary experiments with random and speech data; however, we only looked at the effects of Hebbian learning on separation. In Chapter 4 we introduced a special type of reinforcement learning and referred to LSMs created with this learning method as RLSMs. We performed experiments with only artificial data on very small liquids, but explored the effects of reinforcement learning on both separation and accuracy. In Chapter 5 we introduced Separation Driven Synaptic Modification (SDSM), which was inspired by RLSMs. In Chapter 6 we explored the effects of SDSM on both separation and accuracy using a variety of difficult artificial problems. In this chapter, using the same artificial problems outlined in Chapter 6, we will compare all three of these algorithms in terms of separation and accuracy.

7.1 Parameter Settings

Up to this point, neither Hebbian nor reinforcement learning have been applied to large scale problems to look at accuracy. Considering that both of these learning methods are significantly slower than SDSM, after minimal preliminary experiments, we maintained most of the parameter settings found in Table 6.2. The only difference



(a) Mean Accuracy



(b) Mean Separation

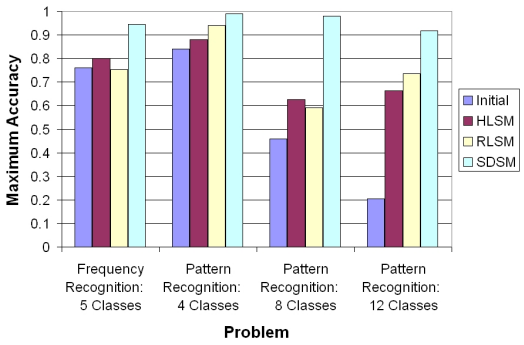
Figure 7.1: A comparison of traditional liquids (initial), HLSMs, RLSMs, and LSMs created by SDSM. The comparison is across four problems. Results are the mean accuracy (a) and separation (b) of fifty LSMs.

made to these settings was to change the learning rate, λ, from $5e^{-10}$ to $1e^{-9}$. The additional parameter settings unique to the reinforcement learning were as follows: the gain, γ, was set to 5; the discount factor for eligibility, β, was set to 0.99. The parameter settings for the STDP (spike time dependant plasticity) synapses used in Hebbian learning were the same as those used in Chapter 3.

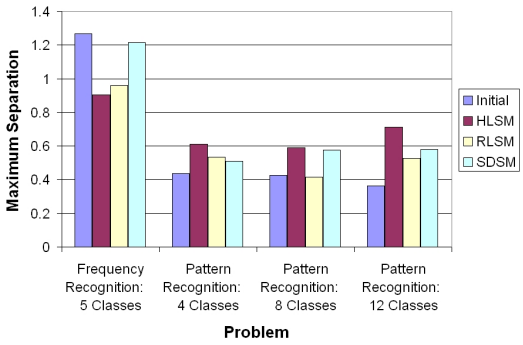
7.2 Results

As in Chapter 6, fifty experiments were run for each algorithm for each of the four problems. The mean results are shown in Figure 7.1, and the best results obtained are shown in Figure 7.2. The results include initial liquids taken for comparison against the traditional LSM. These initial liquids were a different set than those used in Chapter 6 but were created under exactly the same conditions.

To be able to more fully compare how the different algorithms work, the mean separation history for HLSMs (Figure 7.4) and RLSMs (Figure 7.6) have been included. These can be compared to the separation history for SDSM in Figure 6.6. Likewise, a specific representative example from each of the algorithms has been in-



(a) Max Accuracy



(b) Max Separation

Figure 7.2: A comparison of traditional liquids (initial), HLSMs, RLSMs, and LSMs created by SDSM. The comparison is across four problems. Results are the best accuracy (a) and separation (b) out of fifty LSMs.

	Traditional	HLSM	RLSM	SDSM
Frequency Recognition: 5 Classes	0.455	0.669	0.343	0.829
Pattern Recognition: 4 Classes	0.386	0.560	0.431	0.929
Pattern Recognition: 8 Classes	0.367	0.544	0.290	0.774
Pattern Recognition: 12 Classes	0.493	0.512	0.180	0.825

Table 7.1: The ratio of mean accuracy to maximum accuracy across four problems for traditional LSMs, HLSMs, RLSMs, and LSMs using SDSM.

cluded: Figure 7.3 shows an example from a HLSM and Figure 7.5 shows an example from a RLSM.

7.3 Discussion

From these results it is clear that SDSM is the best algorithm of those presented for the problems explored in this study. Not only does it yield the overall best liquid out of all of those created for every problem (Figure 7.2), but it also results in the most consistently effective liquids as indicated by the mean accuracy ratings (Figure 7.1). This last statistic is most telling since SDSM scores a mean accuracy rating of nearly double any other method for every problem we explored. Another way of analyzing the constancy of the algorithms is to look at the ratio of mean

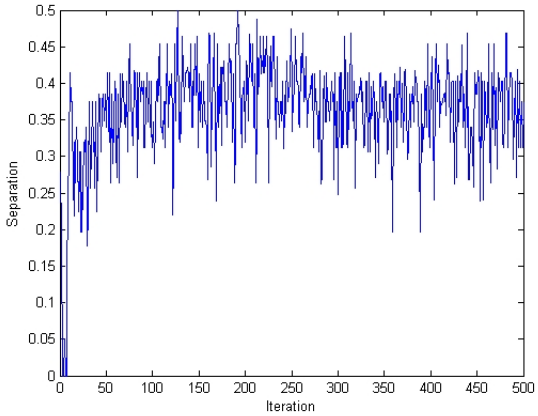
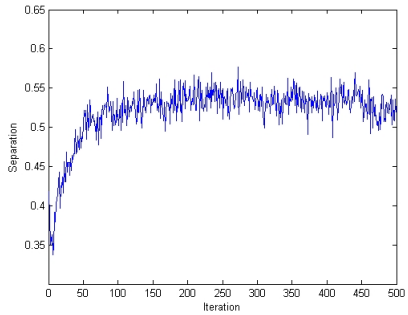


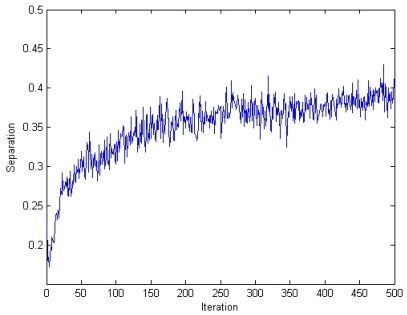
Figure 7.3: The separation of a liquid at each iteration during training of a Hebbian LSM. This is a single representative example out of two hundred different liquids created in this set of experiments. This particular liquid was created using the pattern recognition problem with eight classes.

accuracy to maximum accuracy shown in Table 7.1. We see that SDSM shows the highest mean to maximum accuracy ratio across all four problems, demonstrating that most of the liquids produced behave close to the best liquid created. The reason for SDSM's success can be most likely tied to the fact that it focuses on making specific changes to the liquid that will increase the liquid's separation. These changes are a function of individual components within the separation metric. Reinforcement learning does use separation in it's reward metric; however, the regulation provided by the separation value in this case is only indirect. As for Hebbian learning, it is unsupervised and doesn't even use the separation metric. Once again, the validity of separation as a means to quantify the effectiveness of a liquid is supported.

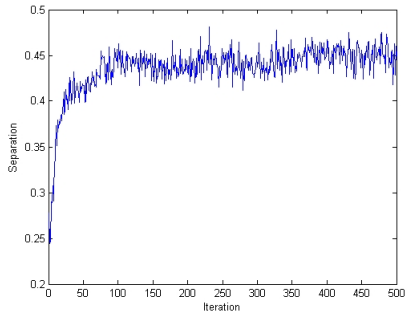
It is important to recall that for both the Hebbian and reinforcement learning algorithms, parameter settings were only superficially explored for these experiments. The primary reason for this was the extensive amount of time it takes to run these algorithms coupled with the unsatisfactory results obtained in earlier experiments (see Chapters 3 and 4). Finding the optimal settings for these algorithms given a particular problem may very well yield better results. The fact that they both



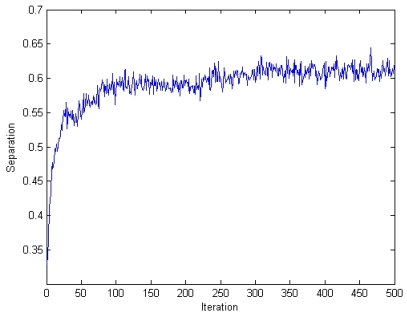
(a) Frequency Recognition: 5 classes



(b) Pattern Recognition: 4 classes



(c) Pattern Recognition: 8 classes



(d) Pattern Recognition: 12 classes

Figure 7.4: The mean separation history of HLSMs for four problems. Each history is an average if fifty trials.

perform at least marginally better than traditional LSMs indicates potential for these algorithms. Looking at Figures 7.4 and 7.6 also demonstrates that the algorithms are generally performing in the direction that they should—an asymptotic increase in separation. Of particular interest is the effect of Hebbian learning on the liquid. HLSMs perform better than both RLSMs and traditional LSMs even though they are based on unsupervised learning. This strengthens the validity of LSMs as a biological model since Hebbian learning has been implicated as a learning mechanism in the human nervous system [21]. Since there is potential for HLSMs and RLSMs to accurately classify data, future work should explore the parameter space of these algorithms for problems like pattern and frequency recognition.

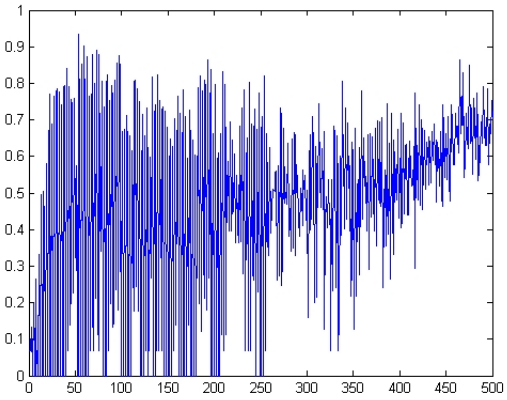
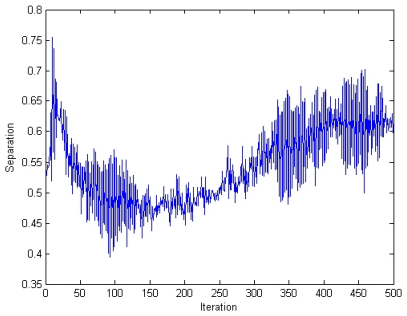
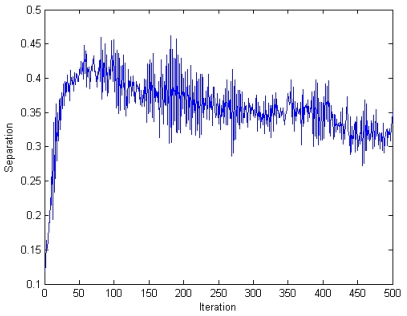


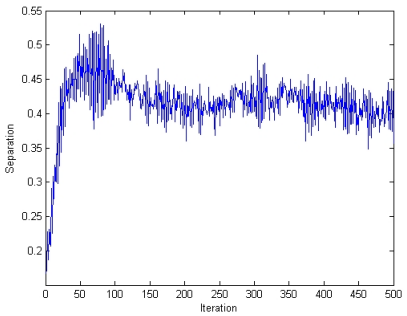
Figure 7.5: The separation of a liquid at each iteration during a training of a RLSM. This is a single representative example out of two hundred different liquids created in this set of experiments. This particular liquid was created using the pattern recognition problem with eight classes.



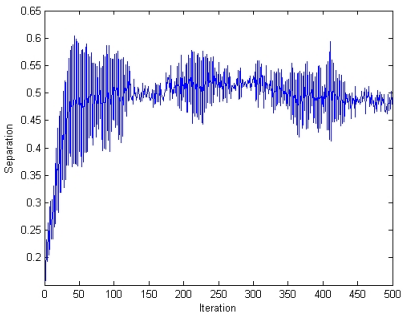
(a) Frequency Recognition: 5 classes



(b) Pattern Recognition: 4 classes



(c) Pattern Recognition: 8 classes



(d) Pattern Recognition: 12 classes

Figure 7.6: The mean separation history of RLSMs for four problems. Each history is an average if fifty trials.

Chapter 8

TIMIT Classification with SDSM

Liquids created with Separation Driven Synaptic Modification (SDSM) have been shown to be significantly better than traditionally created liquids. Although other methods of creating liquids have also shown improvements, the relative speed of SDSM makes it a more desirable choice for further research. In this chapter we will explore the use of SDSM in classifying phonemes found within the TIMIT dataset [7].

8.1 TIMIT

TIMIT consists of 6300 spoken sentences sampled at 16 kHz read by 630 people employing various English dialects. Since we are interested in identifying context independent phonemes, each sentence was broken down into its individual phonemes using phonetic indices included with the sound files. This resulted in 177389 training instances. Each of these phoneme WAV files was then converted into its 13 Mel frequency cepstral coefficients (mfccs) [5] sampled at 200 MHz. Finally, the mfccs were converted into thirteen spike trains (one for each mfcc) with a firing rate calculated using Equation 8.1. This equation is comparable to the one in chapter 3 that was taken from [11].

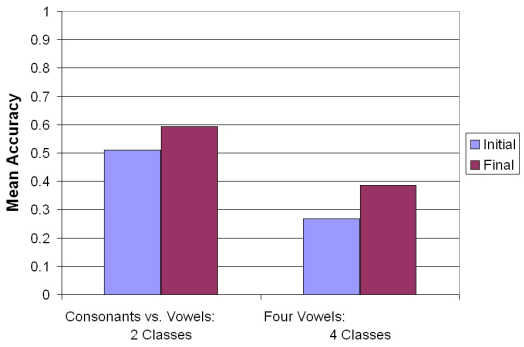
$$Rate_i(t) = \frac{\text{mfcc}_i(t) - \omega_i}{(\Omega_i - \omega_i)} \cdot MaxRate \tag{8.1}$$

Here $\text{mfcc}_i(t)$ is the i^{th} mfcc value at time t which corresponds with the firing rate for input neuron i; ω_i is the minimum value of the i^{th} mfcc, and Ω_i is its maximum value. $MaxRate$ is the maximum rate of firing possible for a given input neuron. $MaxRate$ is a constant that, based on empirical results, we defined as 200 spikes per second for all experiments in this chapter. Additionally, preliminary experiments showed that in order for the liquid to achieve any appreciable level of separation, the time span of the input needs to be on the order of one second due to the operating timescale of the liquid. Since single phonemes have a length on the order of $50ms$, each of the spike trains was temporally stretched using Equation 8.2.

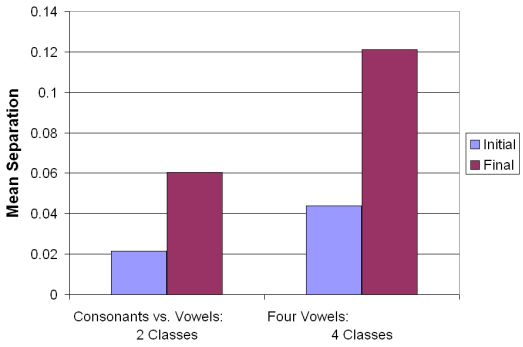
$$T_{new} = \frac{1}{1 + e^{-k \cdot T_{old}}} \tag{8.2}$$

Here T_{old} is the original length of a spike train while T_{new} is the new stretched length. The variable k is a sigmoid gain that we set to five, based on empirical results, for all of the experiments shown in this chapter.

The TIMIT dataset contains roughly fifty two phonemes. Out of context, correctly identifying all of these is a daunting task. Using six natural classes of phonemes, we have reduced this problem to two simpler problems. The first is a general problem that involves identifying phonemes as either consonants or vowels. For this problem "stops", "affricates", and "fricatives" are considered consonants. "Nasals" and "semivowels" are removed to avoid ambiguous sounds. The training data consisted of 1000 instances of each class and the test data contained one hundred instances of each class. The second problem is more specific and involves identifying one of four distinct "vowel" phonemes. The phonemes used in this problem are ē as in b*ee*t; ĕ as in b*e*t; ŭ as in b*u*t; and the *er* sound in butt*er*. For this problem, the training data consisted of one hundred and fifty instances of each class while the test data contained fifty instances.



(a) Mean Accuracy



(b) Mean Separation

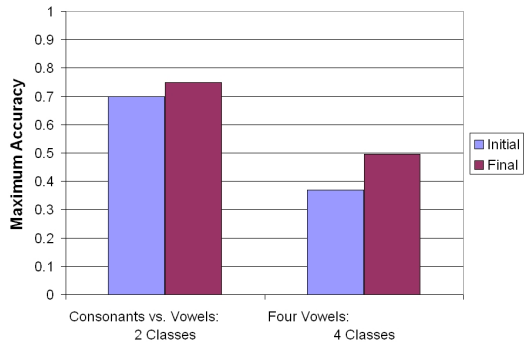
Figure 8.1: A comparison of SDSM and traditional LSMs across two problems derived from the TIMIT dataset. Results are the mean accuracy (a) and separation (b) of fifty LSMs.

8.2 Results

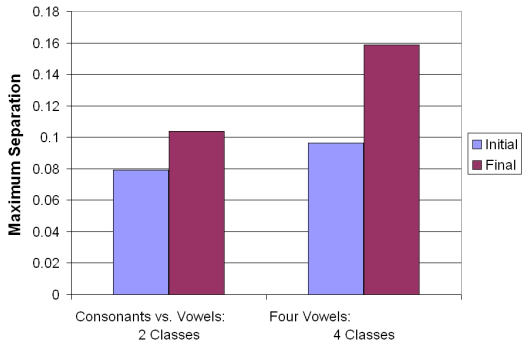
The two problems outlined above were run on LSMs using SDSM, and traditional LSMs. The mean results can be found in Figure 8.1 and the best results are in Figure 8.2. These results were obtained by running the problems on fifty liquids either generated with SDSM or created randomly, with both the accuracy of the LSMs as well as the separation of the liquids displayed. Keep in mind that the liquids showing the best separation do not necessarily correspond to the LSMs with the highest accuracy. The parameter settings used in these algorithms were the same as those outlined in Chapter 6 with the exception of the number of training iterations. In experiments with TIMIT, SDSM was only run for two hundred iterations since liquids tended to reach a plateau in separation improvement by this point.

8.3 Discussion

Although the results of SDSM on TIMIT data are not as distinguished as the results obtained with artificial template matching (Chapter 6), SDSM still shows a significant improvement over traditional liquids. Based on Figure 8.1(a), on average



(a) Max Accuracy



(b) Max Separation

Figure 8.2: A comparison of SDSM and traditional LSMs across two problems derived from the TIMIT dataset. Results are the best accuracy (a) and separation (b) out of fifty LSMs.

traditional liquids do no better than guessing with either of the phoneme recognition problems while LSMs using SDSM improve over this baseline. While neither of these phoneme recognition problems are performed at an immediately applicable level, even when only looking at the best liquids created with SDSM (Figure 8.2(a)), it should be reiterated that these results are obtained by classifying individual phonemes completely out of context. Most of these sounds last one twentieth of a second and span a diverse range of accents. Often speech recognition tasks involve classifying phonemes as they are heard in a stream of data, thus providing context for each sound (i.e. preceding phonemes and pauses). We chose to perform only out of context experiments in order to keep these problems parallel to the artificial ones in chapter 6 and to focus on the separation properties of the liquid.

The improvement of SDSM LSMs over traditional LSMs is clear. Also, the fact that the results from both of these real-data problems were obtained using essentially the same parameters as those obtained from all five of the artificial-data problems in Chapter 6, emphasizes another strength of the SDSM algorithm—a robustness of algorithm parameters. Extensive parameter exploration of the liquids for these problems did not show a marked improvement over the settings already obtained in

Chapter 6. Parameter exploration on a problem by problem basis is a ubiquitous and time consuming component of machine learning. While these results do not exclude the possibility of the necessity for parameter exploration on other problems, they show that there is a level of robustness here not common in machine learning.

For the consonants-versus-vowel problem, Figure 8.1(b) shows a large improvement in mean separation from traditional liquids to those created with SDSM. This demonstrates that SDSM has indeed performed as it is presumed to by improving separation. The fact that accuracy also improves supports the correlation between liquid separation and LSM accuracy. The fact that the magnitude of the improvement in separation does not correlate with the magnitude of accuracy improvement reveals an imperfection in the separation metric. This has also been noted in previous chapters and illuminates one path for future research. Since SDSM is dependant on the current separation metric, a new algorithm would need to be implemented after any appreciable change to the metric. For this reason, we leave this area of research to future studies.

Chapter 9

Conclusions and Future Work

The computational potential of recurrent spiking neural networks is evidenced by living systems and has been established theoretically. Unfortunately, a satisfactory method for exploiting that potential has yet to be discovered. For now, the liquid state machine is a viable alternative. Somewhat haphazardly taking advantage of this potential, it has shown a level of functionality on par with other contemporary learning algorithms. The primary weakness with LSMs is the fact that success is largely random and only achievable after numerous "guesses". In this thesis we have shown several methods that take some of this randomness out of the LSM creation process. More importantly, these methods have shed light on potentially applicable approaches for training recurrent SNNs and one of these methods has even exhibited learning transfer, a difficult problem frequently investigated in current machine learning research. Finally, the success of these algorithms have strengthened the case for LSMs in general.

Throughout this thesis we report on experiments comparing our new algorithms with traditional LSMs. In all of these experiments we created relatively few liquids, on the order of fifty liquids for each data point rather than the typical hundreds of liquids. We did this in order to judge our new algorithms against a strict criteria—can the LSMs perform well with limited attempts at liquid creation? If they can, then we have successfully reduced randomness in the liquid creation process, one of the goals of our research.

Chapter 2 introduced a new metric for assessing the quality of a liquid without needing to look at the accuracy of the resulting LSM. This metric is a variation of the separation metric proposed by Goodman and has been empirically shown throughout this thesis to correlate well with accuracy. Having this reliable estimate of liquid quality is essential for two of the algorithms we have presented. Although our separation metric shows a strong correlation with accuracy, we have exposed several weaknesses in the metric. Future research could look at applying alternative metrics to these algorithms, such as statistical complexity [4]. Additionally, it would be interesting to try incorporating clustering concepts into separation since there appears to be a natural fit.

In Chapter 3 we introduced Hebbian learning to liquid state machines and showed that Hebbian learning improves the separation property of liquids. While the experiments of this chapter were preliminary in nature, they show that the idea of modifying liquids for a given problem is promising. These experiments provided the impetus for the rest of the research presented in this thesis and so were critical in that respect. Later, in Chapter 7 we showed that HLSMs do indeed outperform traditional LSMs when drawing from a pool of only fifty liquids. They surpass traditional LSMs in terms of both accuracy and the separation property of liquids.

In Chapter 4 we described a reinforcement learning algorithm that could be used in LSMs. The results in Chapter 7 show that while reinforcement learning doesn't perform on the same level as Hebbian learning or SDSM, RLSMs still yield higher accuracy than traditional LSMs in all pattern recognition problems. Since RLSMs are dependent on separation for the reward function, they could be improved with a new separation metric. Future work that explores different separation metrics could include implementing RLSMs with these metrics.

Separation driven synaptic modification (SDSM) was introduced and explored in Chapters 5 and 6. SDSM shows encouraging results in improving the accuracy

of LSMs as demonstrated by achieving mean accuracies of more than double those of traditional LSMs for several different problems. Selecting the best liquid out of a pool of only fifty liquids resulted in accuracies of over 90% whereas traditional liquids performed as low as 34%. The fact that the mean accuracy of these SDSM augmented LSMs was also high, indicates that SDSM is much more consistent than traditional LSMs. Equally important, Chapter 6 shows that the generalization properties of LSMs are maintained and possibly even improved by SDSM. This is particularly interesting since what is happening here is effectively learning transfer. Future work could confirm the extent of this learning transfer by exploring a greater variety of problems.

Chapter 8 showed how SDSM fared when faced with a difficult speech recognition problem. Even though the results were not satisfactory from an application point of view, SDSM still performed significantly better than traditional LSMs. The phoneme recognition problem presented in this chapter was particularly difficult due to the lack of temporal context. It would be interesting to see how the algorithm performs with context incorporated into the problem–for example, if the liquid was presented with a stream of speech data rather than one phoneme at a time. The challenge we faced with phoneme recognition may also lie with our process of converting mfccs into spike trains. Different methods of encoding input into spike trains would be another useful area for future investigation.

The algorithms presented in this thesis are essentially training algorithms for highly recurrent SNNs. Furthermore, some of these algorithms, SDSM in particular, have been shown to be effective under non-trivial circumstances—a first in recurrent SNN related machine learning. As most of the results in this thesis are empirical, analytical verification (proof of convergence, etc.) would strengthen the results and provide deeper insights into the LSM model. The function of LSMs in general can be compared to SVMs (support vector machines) with the liquid effectively playing the

role of the kernel. Finding a SVM that corresponds to SDSM (or HLSMs, or RLSMs) would make it possible to mathematically explain the behavior of the algorithms presented in this thesis, and thus understand analytically their underlying mechanics. This in turn would supply a means of better understanding recurrent SNNs and provide valuable insights into machine learning in general.

Appendix A

Comparison of Liquid Creation Methods

These are the results used to obtain the Figures in Chapters 6.3 and 7. The results are all either the mean result of fifty liquids or the maximum result obtained out of fifty liquids.

	Traditional	HLSM	RLSM	SDSM
Frequency Recognition: 5 Classes	0.346	0.5355	0.2576	0.7822
Pattern Recognition: 4 Classes	0.3242	0.4932	0.405	0.9197
Pattern Recognition: 8 Classes	0.1687	0.3402	0.1712	0.759
Pattern Recognition: 12 Classes	0.1003	0.3393	0.1324	0.7562

Table A.1: Mean Accuracy

	Traditional	HLSM	RLSM	SDSM
Frequency Recognition: 5 Classes	0.4288	0.4764	0.1796	0.5557
Pattern Recognition: 4 Classes	0.1144	0.295	0.1851	0.4361
Pattern Recognition: 8 Classes	0.126	0.3493	0.118	0.4548
Pattern Recognition: 12 Classes	0.1137	0.4403	0.1303	0.4867

Table A.2: Mean Separation

	Traditional	HLSM	RLSM	SDSM
Frequency Recognition: 5 Classes	0.76	0.8	0.752	0.944
Pattern Recognition: 4 Classes	0.84	0.88	0.94	0.99
Pattern Recognition: 8 Classes	0.46	0.625	0.59	0.98
Pattern Recognition: 12 Classes	0.2033	0.6633	0.7367	0.9167

Table A.3: Max Accuracy

	Traditional	HLSM	RLSM	SDSM
Frequency Recognition: 5 Classes	1.267	0.9048	0.9611	1.215
Pattern Recognition: 4 Classes	0.4352	0.6127	0.533	0.5098
Pattern Recognition: 8 Classes	0.4261	0.5907	0.4143	0.5771
Pattern Recognition: 12 Classes	0.3625	0.7135	0.5272	0.579

Table A.4: Max Separation

Appendix B

Generalization of Liquids

These are the results used to obtain the Figures in Chapter 6.4. The "SDSM" results refer to the results using SDSM while "traditional" results refer to traditional LSMs. The labels for the columns are as follows: "FR5" is the Frequency Recognition problem with 5 Classes, "PR4" is the Pattern Recognition problem with 4 Classes, "PR8" is the Pattern Recognition problem with 8 Classes, and "PR12" is the Pattern Recognition problem with 12 Classes.

	FR5	PR4	PR8	PR12
Frequency Recognition: 5 Classes	0.7822	0.7908	0.6237	0.5615
Pattern Recognition: 4 Classes	0.8994	0.9197	0.8145	0.7627
Pattern Recognition: 8 Classes	0.8945	0.8959	0.759	0.7007
Pattern Recognition: 12 Classes	0.8886	0.91	0.7984	0.7562

Table B.1: Mean SDSM Accuracy

	FR5	PR4	PR8	PR12
Frequency Recognition: 5 Classes	0.5557	0.3571	0.391	0.4329
Pattern Recognition: 4 Classes	0.7175	0.4361	0.4634	0.4905
Pattern Recognition: 8 Classes	0.7127	0.4362	0.4548	0.4773
Pattern Recognition: 12 Classes	0.6916	0.4284	0.4566	0.4867

Table B.2: Mean SDSM Separation

	FR5	PR4	PR8	PR12
Frequency Recognition: 5 Classes	0.944	0.94	0.87	0.8233
Pattern Recognition: 4 Classes	0.984	0.99	0.95	0.91
Pattern Recognition: 8 Classes	0.976	1	0.98	0.9333
Pattern Recognition: 12 Classes	0.976	1	0.945	0.9167

Table B.3: Max SDSM Accuracy

	FR5	PR4	PR8	PR12
Frequency Recognition: 5 Classes	1.215	0.453	0.5148	0.5631
Pattern Recognition: 4 Classes	1.024	0.5098	0.5275	0.5981
Pattern Recognition: 8 Classes	0.8971	0.5472	0.5771	0.6364
Pattern Recognition: 12 Classes	0.8825	0.5369	0.5603	0.579

Table B.4: Max SDSM Separation

	FR5	PR4	PR8	PR12
Frequency Recognition: 5 Classes	0.3628	0.3071	0.1578	0.1085
Pattern Recognition: 4 Classes	0.3629	0.3369	0.1688	0.1084
Pattern Recognition: 8 Classes	0.3388	0.3067	0.1635	0.1051
Pattern Recognition: 12 Classes	0.3778	0.3207	0.1547	0.1061

Table B.5: Mean traditional Accuracy

	FR5	PR4	PR8	PR12
Frequency Recognition: 5 Classes	0.3998	0.09354	0.09876	0.1197
Pattern Recognition: 4 Classes	0.3975	0.1303	0.1279	0.1301
Pattern Recognition: 8 Classes	0.357	0.1033	0.1109	0.1248
Pattern Recognition: 12 Classes	0.4523	0.1153	0.1039	0.1152

Table B.6: Mean traditional Separation

	FR5	PR4	PR8	PR12
Frequency Recognition: 5 Classes	0.752	0.67	0.39	0.26
Pattern Recognition: 4 Classes	0.76	0.65	0.425	0.2667
Pattern Recognition: 8 Classes	0.704	0.68	0.37	0.2567
Pattern Recognition: 12 Classes	0.736	0.58	0.37	0.34

Table B.7: Max traditional Accuracy

	FR5	PR4	PR8	PR12
Frequency Recognition: 5 Classes	1.65	0.3326	0.3474	0.3852
Pattern Recognition: 4 Classes	1.236	0.3855	0.4504	0.3293
Pattern Recognition: 8 Classes	1.062	0.3428	0.3702	0.3801
Pattern Recognition: 12 Classes	1.483	0.4154	0.4049	0.4032

Table B.8: Max traditional Separation

Appendix C

TIMIT Results with SDSM

These are the results used to obtain the Figures in Chapter 8. The "SDSM" results refer to the results using SDSM while "traditional" results refer to traditional LSMs.

	traditional	SDSM
Consonants vs. Vowels: 2 Classes	0.5097	0.5941
Four Vowels: 4 Classes	0.2676	0.3869

Table C.1: Mean Accuracy

	traditional	SDSM
Consonants vs. Vowels: 2 Classes	0.02137	0.0605
Four Vowels: 4 Classes	0.0439	0.1212

Table C.2: Mean Separation

	traditional	SDSM
Consonants vs. Vowels: 2 Classes	0.7	0.75
Four Vowels: 4 Classes	0.37	0.495

Table C.3: Max Accuracy

	traditional	SDSM
Consonants vs. Vowels: 2 Classes	0.07904	0.1039
Four Vowels: 4 Classes	0.09629	0.1586

Table C.4: Max Separation

Bibliography

[1] S. M. Bohte, "Spiking neural networks," Ph.D. dissertation, Centre for Mathematics and Computer Science, 2003.

[2] S. M. Bohte, J. N. Kok, and H. L. Poutré, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, pp. 17–37, 2001.

[3] O. Booij and H. T. Nguyen, "A gradient descent rule for spiking neurons emitting multiple spikes," *Information Processing Letters*, vol. 95, pp. 552–558, 2005.

[4] N. Brodu, "Quantifying the effect of learning on recurrent spiking neurons," *Proceedings of the International Joint Conference on Neural Networks*, pp. 512–517, 2007.

[5] ETSI ES 202 212STQ: DSR, "Extended advanced front-end feature extraction algorithm; compression algorithms; back-end speech reconstruction algorithm," European Telecommunications Standards Institute (ETSI), Tech. Rep., 2003.

[6] R. V. Florian, "A reinforcement learning algorithm for spiking neural networks," *Proceedings of the Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 299–306, 2005.

[7] J. S. Garofolo et al., "Timit acoustic-phonetic continuous speech corpus," http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S1, 1993, linguistic Data Consortium, Philadelphia.

[8] W. Gerstner and W. Kistler, Eds., *Spiking Neuron Models*. New York: Cambridge University Press, 2002.

[9] E. Goodman and D. Ventura, "Effectively using recurrently connected spiking neural networks," *Proceedings of the International Joint Conference on Neural Networks*, vol. 3, pp. 1542–1547, 2005.

[10] ——, "Time invariance and liquid state machines," *Proceedings of the Joint Conference on Information Sciences*, pp. 420–423, 2005.

[11] ——, "Spatiotemporal pattern recognition via liquid state machines," *Proceedings of the International Joint Conference on Neural Networks*, pp. 3848–3853, 2006.

[12] H. Jaeger, "A tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the "echo state network" approach," International University Bremen, Tech. Rep., 2005.

[13] K. Jim, C. L. Giles, and B. G. Horne, "An analysis of noise in recurrent neural networks: Convergence and generalization," *IEEE Transactions on Neural Networks*, vol. 7, pp. 1424–1438, 1996.

[14] R. G. Leonard and G. Doddington, "Tidigits speech corpus," http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S10, 1993, linguistic Data Consortium, Philadelphia.

[15] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[16] ——, "Real-time computing without stable states: A new framework for neural computations based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[17] W. Mass, "Networks of spiking neurons: the third generation of neural network models," *Transactions of the Society for Computer Simulation International*, vol. 14, pp. 1659–1671, 1997.

[18] T. Natschläger, "Neural micro circuits," http:/www.lsm.turgraz.at/index.html, 2005.

[19] T. Natschläger, W. Maass, and H. Markram, "The "liquid" computer: A novel strategy for real-time computing on time series," *Special Issue on Foundations of Information Processing of TELEMATIK*, vol. 8, no. 1, pp. 39–43, 2002.

[20] D. Norton and D. Ventura, "Preparing more effective liquid state machines using Hebbian learning," *Proceedings of the International Joint Conference on Neural Networks*, pp. 4243–4248, 2006.

[21] O. Paulsen and T. J. Sejnowski, "Natural patterns of activity and long-term synaptic plasticity," *Current Opinion in Neurobiology*, vol. 10, pp. 172–179, 2000.

[22] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt, "An experimental unification of reservoir computing methods," *Neural Networks*, vol. 20, pp. 391–403, 2007.

[23] D. Verstraeten, B. Schrauwen, and D. Stroobandt, "Adapting reservoirs to get Gaussian distributions," *European Symposium on Artificial Neural Networks*, pp. 495–500, 2007.

[24] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. V. Campenhout, "Isolated word recognition with liquid state machine: a case study," *Information Processing Letters*, vol. 95, pp. 521–528, 2005.