2005-07-01

Time Invariance and Liquid State Machines

Eric Goodman
ya_krutoi@hotmail.com

Dan A. Ventura
ventura@cs.byu.edu

Original Publication Citation

Eric Goodman and Dan Ventura, "Time Invariance and Liquid State Machines", Proceedings of the Joint Conference on Information Sciences, pp. 42-423, July 25.

Time Invariance and Liquid State Machines

Eric Goodman and Dan Ventura

Computer Science Department, Brigham Young University, Provo, UT 84602
eric_goodman@byu.edu, ventura@cs.byu.edu

Abstract

Time invariant recognition of spatiotemporal patterns is a common task of signal processing. Liquid state machines (LSMs) are a paradigm which robustly handle this type of classification. Using an artificial dataset with target pattern lengths ranging from 0.1 to 1.0 seconds, we train an LSM to find the start of the pattern with a mean absolute error of 0.18 seconds. Also, LSMs can be trained to identify spoken digits, 1-9, with an accuracy of 97.6%, even with scaling by factors ranging from 0.5 to 1.5.

Keywords: Time-invariance, Liquid State Machines, Pattern Recognition, Spiking Neurons.

1 Introduction

Signals, which are any time-varying phenomenon, pervade and permeate the world we live in. The series of images that flashes across the retina of the human eye is a signal, as is the succession of sounds heard by the ear. Signals often contain spatiotemporal patterns that occur frequently and have a particular meaning. These patterns can exhibit a large degree of variation, but despite these differences, identification is still necessary. This paper focuses on time-invariant classification of spatiotemporal patterns.

Formally, a *signal* is simply a function of time, $x : \mathcal{T} \rightarrow \mathcal{R}$. A *spatiotemporal pattern*, α, is a tuple $(\alpha_s, \alpha_\tau, \{x(t)\}|_{\alpha_s}^{\alpha_s+\alpha_\tau})$ where

- $\alpha_s \in \mathcal{T}$ is start time of the pattern
- $\alpha_\tau \in \mathcal{R}$ is the temporal length of the pattern
- and $\{x(t)\}|_{\alpha_s}^{\alpha_s+\alpha_\tau}$ is the signal x between times α_s and $\alpha_s + \alpha_\tau$.

Signal x is said to *contain* pattern α.

It is also convenient to talk about *classes of patterns*, which are defined as any arbitrary set of spatiotemporal patterns. A *classifier*, $r : \mathcal{X} \rightarrow \{\mathcal{T} \times \mathcal{R} \times \mathcal{C}\}$, is a function that takes as input signals $x \in \mathcal{X}$ and returns sets of tuples of the form (s, τ, c), where $s \in \mathcal{T}$ is the start time of a spatiotemporal pattern, $\tau \in \mathcal{R}$ is the duration, and $c \in \mathcal{C}$ is the class of the pattern. A classifier r is said to *recognize a pattern* α belonging to class c and contained in signal x if $(\alpha_s, \alpha_\tau, c) \in r(x)$. Also, a classifier r is said to *recognize a class* c if for all $x \in \mathcal{X}$ and for all $\alpha \in c$, if x contains α, $(\alpha_s, \alpha_\tau, c) \in r(x)$.

In this paper we are interested in time-invariant classes of patterns, i.e. patterns that are the same except for location in time and scaling along the time dimension. For clarity, these two ideas shall be referred to as time-shift and time-scale invariance. More precisely, a class c is said to be *time-shift invariant*, if given any patterns $\alpha, \beta \in c$ contained in signals x and y, respectively, $\alpha_\tau = \beta_\tau$ and for all $\epsilon \in [0, \alpha_\tau]$, $x(\alpha_s + \epsilon) = y(\beta_s + \epsilon)$. A class c is *time-scale invariant* if $x(\alpha_s + \epsilon) = y(\beta_s + k\epsilon)$ for all $\epsilon \in [0, \alpha_{tau}]$ and for any $k > 0$. Of course for practical purposes, k will most likely be restricted to some range, but the objective is to have as a large a range as possible.

This paper demonstrates that a paradigm known as liquid state machines (LSMs) [2] [5] [6] can achieve classification for both types of time-invariance. Section 2 gives a brief introduction to LSMs. Sections 3 and 4 present the results LSMs achieve on two datasets which exhibit time variance. Section 5 then wraps up with some conclusions and ideas for future work.

2 Liquid State Machines

To understand the basic idea behind LSMs, imagine a pool of water into which various objects are dropped [6]. As the objects plunge into the liquid, they perturb the surface of the liquid, resulting in complex patterns. These patterns provide a history and describe both temporally and spatially how the objects entered the liquid. Stated another way, we have a signal x, which is transformed into another signal with a function d that encapsulates the dynamics of the liquid. Then a readout function r can then be trained from the transformed signal $d(x)$ to classify the inputs.

Now, instead of a pool of water, consider for a moment the human brain as a liquid. Inputs enter the

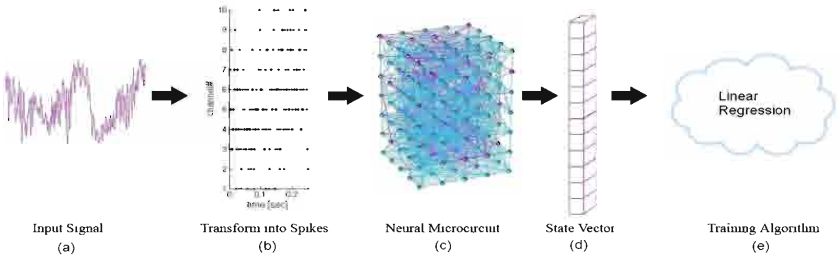

Figure 1: **Training a Liquid State Machine** - First an input signal (a) is transformed into spikes trains via some encoding process (b) (black dots represent times when a neuron spiked). The spikes then stimulate the liquid (c), which in this case is a neural microcircuit. At regular intervals, the state of the liquid is transformed into a multi-dimensional state vector (d). From the sequence of state vectors (d), a training algorithm can be employed to classify the input data, in this case linear regression.

brain through a variety of sources - through eyes and ears and any of the other senses. These inputs are encoded via spike trains, or in other words, series of electrical impulses which form the basis of communication between neurons. These input spikes in turn cause a cascade of spikes within the brain, producing complex interactions, analogous to the ripples and interference patterns produced in the pool of water.

The liquid we use in this paper attempts to model the complex behavior of the brain with a recurrently-connected spiking neural network, often called a neural microcircuit (the modeling software is from [7]). A *spiking neural network* consists of a finite set V of *spiking neurons*, a set $E \subseteq V \times V$ of *synapses*, a *weight* $w_{u,v} \geq 0$ and a *response function* $\epsilon_{u,v} : \mathcal{R}^+ \rightarrow \mathcal{R}$ for each synapse $\langle u, v \rangle \in E$, and a *threshold function* $\Theta_v : \mathcal{R}^+ \rightarrow \mathcal{R}^+$ for each neuron $v \in V$. For more information on spiking networks, see [1] and [4].

Unlike many artificial neuron models in use today, (e.g. perceptrons and sigmoidal units), the neurons in a neural microcircuit actually model the spiking behavior of real biological neurons. Aside from providing many more degrees of freedom, spiking neurons can naturally represent time-varying functions, since by definition they are a temporal phenomenon. Also, neural microcircuits have the potential to retain information from inputs far in the past, allowing inputs to be integrated together over time.

As stated before, a readout function r is trained on the output of the liquid, $d(x)$. However, since the liquid we use is a neural microcircuit, often x must first be encoded as spike trains with some function e in order to interact with neurons of the circuit. Also, to enable the use of a wide variety of training algorithms which can not directly use spikes, samples of the state of the liquid are taken and form a series of vectors, called *state vectors*, which can then be used to train a readout function. This sampling process will be denoted by a function s. All together, the application of an LSM to a signal x can be described by the expression $r(s(d(e(x))))$. Figure 1 displays graphically how an LSM works.

For the most part, the default parameters of the modeling software are used, except as explicitly stated in each individual section. However, all experiments have the following parameters settings in common. Linear least-squares regression, a fairly simple algorithm limited in its representational power, is chosen for the training of the readout function so that results can be attributed more to the circuit and its ability to simplify the task of classification rather than to the power of the learning algorithm. Also, both experiments use a network size of 135 neurons, with the topology of the circuit being a square column of dimension $3 \times 3 \times 15$ neurons. The state vectors are composed of 135 elements, one element for each neuron in the circuit. Each element is a rough approximation of firing rate of the neuron at the time of the sample.

3 Boundary Detection

This first problem tests both the time-shift and time-scale invariant properties of the LSM on an artificial dataset that mimics real world problems. The basic problem can be described as follows. There is a set of time-shift and time-scale invariant pattern classes, $C = \{c_0, c_1, ..., c_n\}$, and a set of signals, $X = \{x_0, x_1, ..., x_m\}$. The task is to find the α_s and α_τ of any pattern $\alpha \in C$ contained in some $x_i \in X$. The problem we solve here is somewhat simplified in that we only consider one pattern class, $C = \{c_0\}$, and signals which have the general form $x_i = [\theta_i, \alpha_i, \phi_i]$, i.e. the pattern $\alpha_i \in c_0$ is sandwiched between two portions of the signal which are completely random, θ_i and ϕ_i.

The artificial dataset is created in the following manner. A template of 100 points is created, drawn from a uniform distribution ranging from -1 to 1.

This template, considered to be of length 0.55 seconds, is then scaled along the time dimension to create 100 instances or target pattern α_i's between 0.1 seconds to 1 second in length, again using a uniform distribution. Each of the 100 α_i's is also sandwiched between two random signals, i.e. θ_i and ϕ_i. Every θ_i and ϕ_i is created uniquely, though in a similar manner to how the template that formed the basis for each of the α_i is created. Like the template, each random piece is created from 100 points, drawn from the same uniform distribution. Also like the template, they are considered to be of length 0.55 seconds and then scaled to the range between 0.1 seconds and 1 second in length. The only difference being that each random piece is created uniquely and never repeated, not within the same signal nor in any other x_i. Finally, each instance is then sampled at a rate of 100 samples per second, using linear interpolation to find the values between points.

Each x_i is translated into spikes in the following manner. A number of input neurons, n_{in} is chosen that will represent the signal over time, for these experiments $n_{in} = 10$. Now, each of the input neurons is assigned to cover a portion of the range of the signal. The assigned range for each neuron, r_j, is calculated with following formula:

$$r_j = \begin{cases} [\frac{(j-1)(\Omega-\omega)}{n_{in}} + \omega, \frac{j(\Omega-\omega)}{n_{in}} + \omega) & 1 \leq j < n_{in} \\ [\frac{(j-1)(\Omega-\omega)}{n_{in}} + \omega, \Omega] & j = n_{in} \end{cases}$$

where Ω is the maximum value in the range and ω is the minimum value. Then for each time t corresponding to a discrete point in the signal having value $x_i(t)$, the neuron j that has $x_i(t) \in r_j$ would fire at time t.

The dataset is divided into ten different randomly generated subdivisions of 80% for training and 20% for validation. From each of the training subdivisions, a linear regressive model is trained on the state vectors, which is then tested on the validation set. An example application of a model to an instance is shown in Figure 2.

As stated earlier, samples of the circuit, state vectors, are taken at periodic intervals and a determination is made by the readout function as to whether that sample point belongs to the target pattern class. A moving average is used to approximate α_{i_s} and α_{i_τ} for each signal x_i. More specifically, given a sequence $v_0, v_1, ..., v_n$, of state vectors, a readout function r, a window size δ, and a threshold η, the following formulae describe how α_{i_s} and α_{i_τ} are approximated:

$$t_s = \underset{0 \leq j \leq n}{\arg\text{first}} \left(\sum_{k=j}^{j+\delta} \frac{r(v_k)}{\delta} > \eta \right)$$

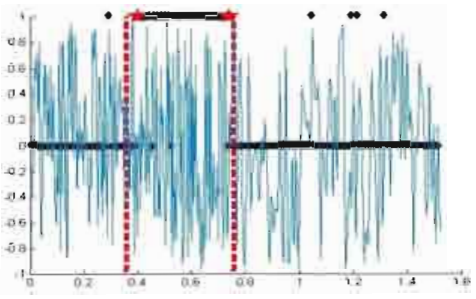

Figure 2: **Orignal signal and output of a trained model applied to it** - The dark colored diamonds indicate the output of the model. A 1 denotes the model believes the pattern is present; a zero denotes otherwise. The vertical dashed lines represent when the pattern is actually occurring, while the stars indicate the estimated start and stop times of the target pattern

$$t_\tau = \underset{t_s < j \leq n}{\arg\text{first}} \left(\sum_{k=j}^{j+\delta} \frac{r(v_k)}{\delta} < \eta \right)$$

$$\hat{\alpha}_{i_s} = \xi(t_s)$$

$$\hat{\alpha}_{i_\tau} = \xi(t_\tau) - \xi(t_s)$$

where the function $\xi : \mathcal{Z} \rightarrow \mathcal{T}$ gives the time at which the j^{th} state vector is sampled.

After the start and end times of the pattern are estimated on the training data, the median error is used to offset the estimation. For example, the adjustment for the start times is accomplished by

$$est(i) = est(i) - median(\epsilon)$$

where $est(i)$ signifies the estimated start time for signal x_i and ϵ represents the set of start errors from the initial estimation. Similarly, the end times are adjusted. As a general rule, the LSM has a tendency to be late in its prediction, so subtracting the median error helps to correct this fault.

Experiment	1	2	3
Mean Syn. Delay(s)	default	0.01	0.55
Train Error Start (s)	0.21 (0.02)	0.23 (0.03)	0.16 (0.02)
Train Error Stop (s)	0.19 (0.03)	0.19 (0.02)	0.16 (0.02)
Val. Error Start (s)	0.30 (0.06)	0.22 (0.04)	0.18 (0.03)
Val. Error Stop (s)	0.25 (0.06)	0.19 (0.03)	0.15 (0.04)

Table 1: Average error from three experiments on the boundary detection problem. Numbers in parenthesis denote standard deviations.

Table 1 shows the results on the dataset for three different tests. In all three cases, the circuit is sampled every 0.01 seconds. The only parameter that is varied in each of the three tests is the mean delay

time of the synapses. The first test uses the default mean delay time, which varies from 8e-4 to 1.5e-3 seconds, depending on the type of synapse. The second test sets the mean delay time for all synapses to 0.01 seconds, and the final test has it at 0.55 seconds.

4 Spoken Digit Recognition

This next problem tests the time-scale invariant properties of the LSM using a sample of spoken digits. The task is to be able to identify the spoken digits, 1-9, despite time-scaling variances. Two examples of each digit are selected from the TIDIGITS corpus [3] and any silence is removed so that just the speech signal remains. To test the time-scale invariance, numerous additional instances are created by shortening and lengthening the original signals by a factor $c \in \{0.5, 0.55, 0.60, ..., 1.40, 1.45, 1.5\}$. The resultant dataset is of size 378 instances.

The translation process into spikes differs from what is used in the previous example. Each instance is first translated into series of 13 Mel frequency cepstral coefficients (mfccs). The frame size, n_s, which is the number of points used to compute the Fourier transform, is set to be the largest number of points which is both a power of two and less than 30 ms. The frame rate, or how often mfccs are calculated, is then set to be $\frac{n_s}{2}$. Each of the 13 mfccs is assigned its own neuron to represent it. The maximum and minimum values of each the mfccs across all instances are found. From these maximum and minimum values, the firing rate of of each of the 13 neurons is determined by the following equation:

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

where Ω_i is the largest i^{th} mfcc, ω_i is the smallest i^{th} mfcc, and where the maximum rate is set to 200 Hz.

Experiment	1	2	3
Mean Syn. Delay(s)	default	0.5	1
Train Acc.(%)	90.46(3.41)	97.79(0.59)	98.26(1.05)
Val. Acc.(%)	80.00(5.61)	96.46(2.05)	97.59(2.11)

Table 2: Average accuracy for three experiments on the speech data. Numbers in parenthesis denote standard deviations.

Table 2 displays the results obtained on the speech data. Three experiments are conducted, each with varying mean delay times. The sample rate is held constant, sampling every 0.1 seconds. As with the previous experiment, ten iterations are run on ten different subdivisions of the dataset, with 80% used for training and 20% used for validation.

5 Conclusions and Future Work

LSMs provide a robust way of handling signals that require time-invariant classification of spatiotemporal patterns. The first artificial dataset provides evidence that LSMs can give good accuracy even when there exists a large amount of time-shift and time-scale variance in the data. The more realistic problem of identifying digits further shows that LSMs have the potential to solve real-world problems that demonstrate time-scale variance.

Better methods for determining α_s and α_τ need to be explored; the naive approach of using a moving average worked well for a first attempt, but could be much improved, perhaps with statistical methods. Also, further study is required to understand how delay times affect the accuracy of the LSM. Results from both datasets hint that long mean delay times somehow increase accuracy, but how and why remains unexplained.

Acknowledgements

We thank Sandia National Laboratories for partially funding this research.

References

[1] W. Gerstner and W.M. Kister. *Spiking Neuron Models: Single Neurons, Populations, Plasticity.* Cambridge University Press, 2002.

[2] S. Häusler, H. Markram, and W. Maass. Perspectives of the high dimensional dynamics of neural microcircuits from the point of view of low dimensional readouts. *Complexity (Special Issue on Complex Adaptive Systems)*, 8(4):39–50, 2003.

[3] R.G. Leonard and G. Doddington. Tidigits speech corpus. http://morph.1ds.upenn.edu/Catalog/LDC93S10.html, 1993. Texas Instruments, Inc.

[4] W. Maass. On the complexity of networks of spiking neurons. In *Advances in Neural Information Processing*, volume 7, pages 183–190, Cambridge, 1995. MIT Press.

[5] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[6] 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*, 8(1):39–43, 2002.

[7] Neural microcircuits. http://www.lsm.tugraz.at/index.html.