Quasi-Orthogonal Maps for Dynamic Language Recognition

Alan D. Blair[1] **and Jordan B. Pollack**[2]

[1]Dept. of Computer Science, University of Queensland, 4072, Australia, `blair@cs.uq.edu.au`
[2]Dept. of Computer Science, Brandeis University, Waltham, MA 02254, `pollack@cs.brandeis.edu`

Abstract

Although recurrent neural networks can be trained to recognize formal languages from positive and negative examples, it is difficult for them to learn large and complex data sets due to the geometry of their state space, which tends to push activations into the corners and dilute the effect of differentials backpropagated through multiple levels of recurrence.
We propose a new framework for dynamical recognizers in which the state space is the surface of an n-sphere and the maps are either orthogonal or *quasi-orthogonal.*

1 Introduction

A **dynamical recognizer** consists of a *state space* $\mathrm{X} \subseteq \mathbf{R}^n$ together with a partition $\mathrm{X} = \{\mathrm{X}_{\mathrm{yes}}, \mathrm{X}_{\mathrm{no}}\}$ of X into *accept* and *reject* states, an *initial state* $x_0 \in \mathrm{X}$, and a family of maps $f_\sigma : \mathrm{X} \to \mathrm{X}$ indexed by the letters σ in an alphabet A. Its purpose is to recognize a formal language $\mathcal{L} \subseteq A^*$ in the following manner: for any string $\Sigma = \sigma_1 \ldots \sigma_T \in A^*$, let $x_{_T} = f_{\sigma_T} \circ \ldots \circ f_{\sigma_1}(x_0)$. Then Σ is accepted if $x_{_T} \in \mathrm{X}_{\mathrm{yes}}$, rejected if $x_{_T} \in \mathrm{X}_{\mathrm{no}}$.

This framework, introduced in (Pollack, 1991) has been extended in a number of directions both experimental (Giles et al., 1992, Watrous & Kuhn, 1992) and theoretical (Moore, 1997, Casey, 1996). A variety of methods have been developed for extracting a deterministic finite automaton (DFA) from the dynamical recognizer once it is trained (Omlin & Giles, 1996, Das & Mozer, 1994, Manolios & Fanelli, 1994, Zeng et al., 1994) although the recognizer will sometimes induce a non-regular language which cannot be modeled exactly by any DFA (Blair & Pollack, 1997, Kolen, 1993).

Many of these approaches have used a neural network framework in which the state space X is a hypercube and each map f_σ is a composition of a linear transform and a sigmoidal function. Such architectures have been successfully applied to many tasks, but have generally had difficulty learning longer strings or more complicated languages without considerable modifications to the learning algorithm. Among the reasons for these difficulties have been:
(i) the geometry of the neural network state space is such that activations often get stuck in the corners, which can impede further learning,
(ii) the derivative of the sigmoid function tends to dilute the differentials, making it difficult to effectively backpropagate more than 5 or 6 time steps.[1]

The purpose of this paper is to propose a new framework of *quasi-orthogonal maps*, with the aim of overcoming some of these difficulties.

2 Details of Algorithm

In our proposed framework the state space X is the surface of a sphere in n dimensions and, for each $\sigma \in A$, the map $f_\sigma : \mathrm{X} \to \mathrm{X}$ is given by

$$f_\sigma(x) = \frac{\mathrm{A}_\sigma(x)}{||\mathrm{A}_\sigma(x)||}, \quad \text{where} \quad \mathrm{A}_\sigma = \mathrm{U}_\sigma(\mathrm{I} + \mathrm{H}_\sigma)$$

A_σ is a linear transformation composed of one map $(\mathrm{I} + \mathrm{H}_\sigma)$ which is *self-transpose* or *symmetric* ($\mathrm{H}'_\sigma = \mathrm{H}_\sigma$) and another U_σ which is *orthogonal* ($\mathrm{U}'_\sigma \mathrm{U}_\sigma = \mathrm{I}$). We assume that $||\mathrm{H}_\sigma||$ is small enough to keep $||\mathrm{A}_\sigma(x)||$ bounded away from zero when $||x|| = 1$. Note that A_σ would be orthogonal if H_σ were equal to zero. We will call it **quasi-orthogonal** in the general case (where H_σ is small but non-zero). This setup has the following desirable features:
(i) the spherical state space provides a featureless landscape with no corners for the activations to get stuck in,
(ii) keeping the maps nearly orthogonal helps to prevent dilution of the differentials, thus allowing backpropagation to proceed effectively through many levels of recurrence.

We choose for our initial state $x_0 \in \mathrm{X} \subseteq \mathbf{R}^n$ the vector $[010\ldots0]'$ which has second coordinate 1.0 and all other coordinates 0. The partition into accept and reject states is based on the first coordinate $x^0_{_T}$ of $x_{_T}$ — specifically, $\mathrm{X}_{\mathrm{yes}} = \{x_{_T} \,|\, x^0_{_T} >= 0\}$, $\mathrm{X}_{\mathrm{no}} = \{x_{_T} \,|\, x^0_{_T} < 0\}$. So our aim is to make $x^0_{_T}$ close to ε, where ε is $+1$ for accept strings and -1 for reject strings.

To achieve this, we apply a gradient descent algorithm based on backpropagation through time (Williams & Zipser, 1989) using the error function

$$E = \frac{1}{2}(x^0_{_T} - \varepsilon)^2$$

The pull-back of the differential $-d\mathrm{E}$ to the cotangent space of X at $x_{_T}$ is

$$dx_{_T} = (\varepsilon - x^0_{_T}) \begin{bmatrix} (1 - x^0_{_T})^2 \\ -x^0_{_T}\hat{x}_{_T} \end{bmatrix}, \quad \text{where} \quad x_{_T} = \begin{bmatrix} x^0_{_T} \\ \hat{x}_{_T} \end{bmatrix}$$

[1]Almost all approaches have involved some kind of recurrence, though purely feed-forward time-delay networks have been able to learn languages generated from DFA's with a special kind of internal structure (Clouse et al., 1997).

Recall that, for $1 \leq t \leq T$,

$$x_t = \frac{y_t}{||y_t||}, \qquad \text{where} \quad y_t = \mathrm{A}_\sigma(x_{t-1})$$

Subsequent differentials can be computed by iterating the following formulas:

$$dy_t = \frac{(\mathrm{I} - x_t x_t')}{||y_t||} \, dx_t, \quad dx_{t-1} = \mathrm{A}_{\sigma_t}' dy_t$$

The idea is to move A_{σ_t} in the direction of $d\mathrm{A}_{(t)} = dy_t x_{t-1}'$ while preserving its symmetric-orthogonal decomposition. This is achieved by decomposing $\mathrm{U}_{\sigma_t}' d\mathrm{A}_{(t)}$ into

$$\mathrm{U}_{\sigma_t}' d\mathrm{A}_{(t)} = \mathrm{dU}_{(t)} + \mathrm{dH}_{(t)}$$

where $\mathrm{dH}_{(t)}$ is symmetric and $\mathrm{dU}_{(t)}$ is skew-symmetric. The following updates are then performed:

$$\mathrm{dU}_\sigma \leftarrow \mu \, \mathrm{dU}_\sigma + \sum_{\sigma_t = \sigma} \mathrm{dU}_{(t)}$$

$$\mathrm{dH}_\sigma \leftarrow \mu \, \mathrm{dH}_\sigma + \sum_{\sigma_t = \sigma} \mathrm{dH}_{(t)}$$

$$\mathrm{U}_\sigma \leftarrow \mathrm{U}_\sigma e^{\eta \mathrm{dU}_\sigma}, \quad \mathrm{H}_\sigma \leftarrow \delta \, \mathrm{H}_\sigma + \eta \, \mathrm{dH}_\sigma$$

Here η is the learning rate, μ is the momentum and δ is the *weight decay factor* (Hinton, 1986) which we rely on to keep $||\mathrm{H}_\sigma||$ small.

It should be noted that this procedure is computationally intensive and highly non-local. We are certainly not proposing it as a biologically plausible model of learning in the brain, nor as an efficient alternative to symbolic language induction algorithms (Lang, 1992, Trakhtenbrot & Barzdin', 1973). Our purpose is rather to explore the capabilities of new dynamical recognizer architectures and look for more elegant ways to expand their application to larger and more challenging data sets.

Rather than fixing the number of dimensions in advance, we follow a strategy of starting with only 3 dimensions and gradually adding dimensions as the learning proceeds. Note that U_σ lies in the group $\mathbf{O}^n$ of $n \times n$ orthogonal matrices, H_σ in the space $\mathbf{H}^n$ of symmetric $n \times n$ matrices. $\mathbf{O}^n$ embeds naturally into $\mathbf{O}^{n+1}$ (similarly $\mathbf{H}^n$ into $\mathbf{H}^{n+1}$). Whenever learning at a particular dimension appears to have stagnated[2] we increment the dimension and use these embeddings to locate U_σ and H_σ within $\mathbf{O}^{n+1}$ and $\mathbf{H}^{n+1}$, respectively,[3] so the extra dimensions can provide more leeway for the (real valued) activation vectors to gradually spread apart as the training proceeds.

[2]The heuristic we used to detect stagnation was to keep track of the lowest value the error function had reached so far, and check every 50 epochs to see whether this had been reduced by a non-trivial amount from its previous value.

[3]We also add a little noise to U_σ in the directions orthogonal to $\mathbf{O}^n$ inside $\mathbf{O}^{n+1}$.

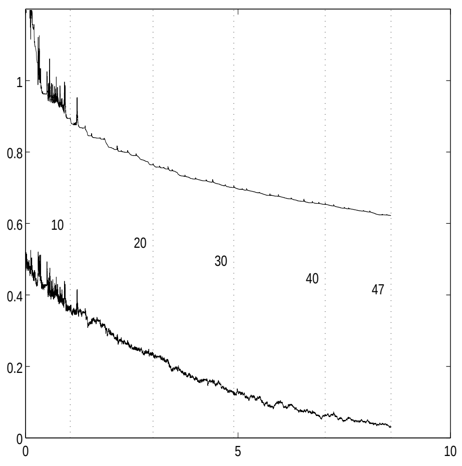

Figure 1: Error function per string (upper) and misclassification rate (lower) for orthogonal recognizer. Horizontal axis measures time in 1000's of epochs; vertical lines track the growth of the state space dimension.

3 Experiments

We tested our architecture on Training Set 7 of the *Abbadingo DFA Competition*,[4] which consists of 1521 strings selected randomly from a uniform distribution of strings of length ≤ 15, and classified by a randomly generated DFA with 65 states.

We believe this kind of data would present a serious challenge for earlier architectures, which had generally relied on datasets in which shorter strings were over-represented (Tomita, 1982) classified by DFA's with many fewer states (Watrous & Kuhn, 1992) or with a special kind of internal structure (Clouse et al., 1997).

The training was done in batch mode, using a fully parallel algorithm on a MasPar MP-2, with two processors devoted to each string.

Experiment 1: The Orthogonal Case

In our first experiment, we set $\delta = 0$ which implies that $\mathrm{A}_\sigma = \mathrm{U}_\sigma$ is always orthogonal. The other parameters were $\eta = 0.0003$, $\mu = 0.9$. The results are shown in Figure 1, where the upper plot shows the average value of E per string, while the lower plot shows the fraction of training strings classified incorrectly. The vertical dotted lines indicate the time at which the dimension reached 10, 20, 30, 40, etc. This orthogonal recognizer ultimately achieved a misclassification rate of less than 5%, but required 47 dimensions to do so.

Experiment 2: Quasi-Orthogonal Case

In our second experiment we set set $\delta = 0.9$ so that A_σ is no longer orthogonal. We kept $\mu = 0.9$ and, after some preliminary trials, reduced η to 0.00002. As Figure 2 shows, this recognizer learned well initially, but started to be unstable as the dimension increased from 20 to 30 (truncated plot). We therefore adopted an annealing schedule, reducing η gradually from 0.00002 to

[4]http://abbadingo.cs.unm.edu/

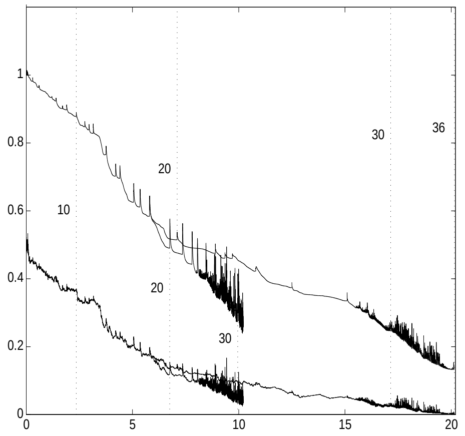

Figure 2: Quasi-orthogonal recognizer without annealing (truncated) and with annealing (extended).

0.000015 in steps of 0.000001 between $d = 19$ and $d = 24$ (extended plot). Compared to the original algorithm, the annealed version showed slower but more reliable improvement, achieving approximately the same value of E and error rate at the same dimension. After 20,000 epochs this quasi-orthogonal recognizer, operating within a 36-dimensional state space, achieved a misclassification rate of less than 0.4% on the training data.

4 Conclusions and Further Work

These experiments seem to indicate a tradeoff between rigidity and flexibility. The flexibility of neural networks to bend and stretch their state space in various directions makes them versatile enough to be good universal function approximators, but restricts the number of levels through which differentials can be successfully backpropagated. On the other hand, the rigidity of the orthogonal maps in Experiment 1 allow successful backpropagation through many levels by ensuring that the differentials retain their size no matter how many times they are backpropagated. However, the fact that they always preserve the distance between activation vectors – never allowing them to be moved closer together or farther apart – makes them an unnatural model for fitting data generated by a typical dynamical system or symbolic DFA.

The quasi-orthogonal maps employed in Experiment 2 seem to find a good compromise between these extremes, allowing the state space to be slightly deformed each time a map is applied, but keeping it rigid enough to allow successful backpropagation through many levels.

We have shown that large, complex data sets can be successfully learned with this paradigm. In further work we hope to address the issue of language induction – examining how quasi-orthogonal maps might generalize to classify new, unseen strings – and apply clustering techniques, analysis of attractor dynamics and other methods to gain a better understanding of the properties and capabilities of this class of dynamical recognizers.

Acknowledgments

This work was funded by ONR grant N00014-96-1-0418, NSF grant IRI-95-29298 and by a University of Queensland Postdoctoral Fellowship.

References

Blair, A.D. & J.B. Pollack, 1997. Analysis of Dynamical Recognizers, *Neural Computation* **9**(5), 1127–1142.

Casey, M. 1996. The Dynamics of Discrete-Time Computation, with Application to Recurrent Neural Networks and Finite State Machine Extraction, *Neural Computation* **8**(6).

Clouse, D.S., C.L. Giles, B.G. Horne & G.W. Cottrell, 1997. Representation and Induction of Finite State Machines using Time-Delay Neural Networks, *Neural Information Processing Systems* **10**, 403–409.

Das, S. & M.C. Mozer, 1994. A Unified Gradient-Descent/Clustering Architecture for Finite State Machine Induction, *Neural Information Processing Systems* **6**, 19–26.

Giles, C.L., C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun & Y.C. Lee, 1992. Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks, *Neural Computation* **4**(3), 393–405.

Hinton, G.E., 1986. Learning Distributed Representations of Concepts, *Proc. Eighth Annual Conference of the Cognitive Science Society*, Amherst, MA, 1–12.

Kolen, J.F. 1993. Fool's Gold: Extracting Finite State Machines from Recurrent Network Dynamics, *Neural Information Processing Systems* **6**, 501–508.

Lang, K.J. 1992. Random DFA's can be Approximately Learned from Sparse Uniform Examples, *Proc. Fifth ACM Workshop on Comp. Learning Theory*, 45–52.

Manolios, P. & R. Fanelli, 1994. First order recurrent neural networks and deterministic finite state automata, *Neural Computation* **6**(6), 1155–1173.

Moore, C., 1997. Dynamical Recognizers: Real-time Language Recognition by Analog Computers, *Theoretical Computer Science* (to appear).

Omlin, C.W. & C.L. Giles, 1996. Extraction of Rules from Discrete-Time Recurrent Neural Networks, *Neural Networks* **9**(1), 41.

Pollack, J.B. 1991. The Induction of Dynamical Recognizers, *Machine Learning* **7**, 227–252.

Tomita, M. 1982. Dynamic construction of finite-state automata from examples using hill-climbing, *Proc. Fourth Annual Conference of the Cognitive Science Society*, Ann Arbor, MI, 105–108.

Trakhtenbrot, B.A. & Ya.M. Barzdin' 1973. *Finite Automata; Behavior and Synthesis* (North-Holland).

Watrous, R.L. & G.M. Kuhn, 1992. Induction of Finite State Languages Using Second-Order Recurrent Networks, *Neural Computation* **4**(3), 406–414.

Williams, R.J. & D. Zipser, 1989. A learning algorithm for continually running fully recurrent neural networks, *Neural Computation* **1**(2), 270.

Zeng, Z., R.M. Goodman & P. Smyth, 1994. Learning Finite State Machines with Self-Clustering Recurrent Networks, *Neural Computation* **5**(6), 976–990.