Dynamic on-line clustering and state extraction: An approach to symbolic learning

Sreerupa Das
Lucent Technologies, Bell Labs Innovations
11900 North Pecos Street, Room 31K52
Denver, CO 80234

Michael Mozer
Department of Computer Science
University of Colorado
Boulder, CO 80309–0430

August 20, 1997

1 Introduction

Researchers often try to understand the representations that develop in the hidden layers of a neural network during training. Interpretation is difficult because the representations are typically highly distributed and *continuous*. By "continuous," we mean that if one constructed a scatter plot over the hidden unit activity space of patterns obtained in response to various inputs, examination at any scale would reveal the patterns to be broadly distributed over the space. Such continuous representations are naturally obtained if the input space and activation dynamics are continuous.

Continuous representations are not always appropriate. Many task domains might benefit from *discrete* representations—representations selected from a finite set of alternatives. Example domains include finite-state machine emulation, data compression, language and higher cognition (involving discrete symbol processing), and categorization. In such domains, standard neural network learning procedures, which have a propensity to produce continuous representations, may be inappropriate.

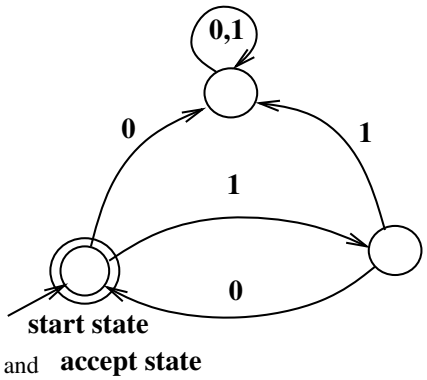

Figure 1: A finite state machine for the language $(10)^*$

The research we describe involves designing an inductive bias into the learning procedure in order to encourage the formation of discrete internal representations.

In the recent years, various approaches have been explored for learning discrete representations using neural networks (McMillan, Mozer, and Smolensky, 1992; Mozer and Bachrach, 1990; Mozer and Das, 1993; Schütze, 1993; Towell and Shavlik, 1992). However, these approaches are domain specific, making strong assumptions about the nature of the task. The research described here is a general methodology that makes no assumption about the domain to which it is applied, beyond the fact that discrete representations are desirable.

2 Finite state machine induction

We illustrate the methodology using the domain of finite-state machine (FSM) induction. An FSM defines a class of symbol strings (also called a language). For example, the language $(10)^*$ consists of all strings with one or more repetitions of 10; 101010 is a positive example of the language, 111 is a negative example. Figure 1 shows an FSM for the language $(10)^*$. The FSM consists of a finite set of states and a function that maps the current state and the current symbol of the string into a new state. Certain states of the FSM are designated "accept" states, meaning that if the FSM ends up in these states, the string is a member of the language. The induction problem is

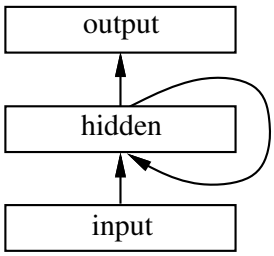

Figure 2: A generic recurrent network architecture that could be used for FSM induction. Each box corresponds to a layer of units, and arrows depict complete connectivity between layers. At each time step, a new symbol is presented on the input and the input and hidden representations are integrated to form a new hidden representation.

to infer an FSM that parsimoniously characterizes the positive and negative exemplars, and hence characterizes the underlying language.

A generic recurrent network architecture, like the one in Figure 2, could be used for FSM emulation and induction. A string is presented to the network, one symbol at a time. The input layer activity pattern indicates the current symbol. Following the end of string presentation, the network is trained to output whether or not the string belongs to the language. Such an architecture, trained by a gradient descent procedure, is able to learn to perform this or related tasks (Elman, 1990; Giles et al., 1992; Pollack, 1991; Servan-Schreiber, Cleeremans, and McClelland, 1991; Watrous and Kuhn, 1992).

If we take a closer look at the hidden unit activations during training such an architecture, we see that the activation values are scattered widely in the activity space during early phases of training. Nontheless, once the network has been trained successfully, the activations tend to achieve stable values. In order to perform the task of correct classification, one might expect that the hidden activity patterns at any point during the string presentation would corresponds to a state of the underlying FSM. For example, when trained on strings from the language $(10)^*$, we would expect to see three distinct activity values for the hidden units as the underlying FSM

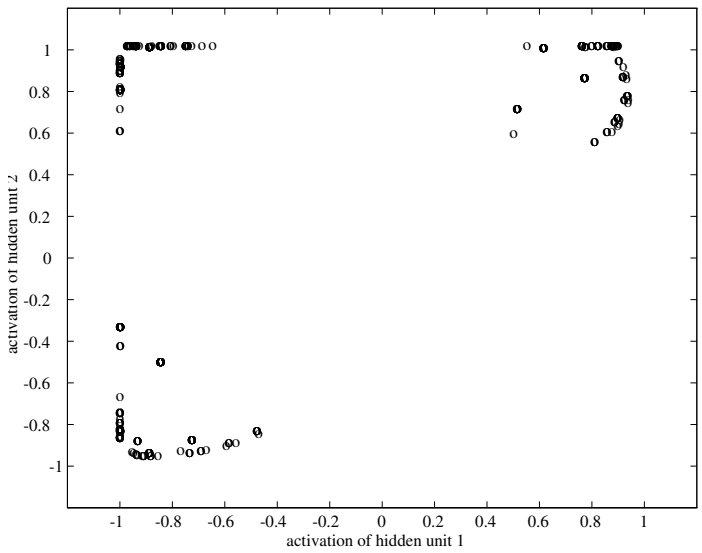

Figure 3: A scatter plot of hidden activity space of a generic recurrent network having two hidden units, trained on the language $(10)^*$. Each circle corresponds to a hidden state vector during the training phase. The three clusters of hidden states correspond to the three states in the inferred FSM.

consists of three states. As a consequence of successful training, the hidden unit activation values indeed tend to form distinct clusters. Figure 3 shows a typical scatter plot of the hidden activity space for a two-hidden-unit network following training, and we see that three distinct clusters have evolved as a result of training. Roughly, these regions of hidden activity space can be identified with states in the FSM; when presented with an input string, the induced states appear to make transitions that correspond to the transitions in the underlying FSM (Figure 1).

However, because the hidden unit activities are continuous, state representations can drift from one state to the other. This is most evident when input strings that are presented are longer than those in the training set. Figure 4 plots the same hidden activity space as in Figure 3 but when presented with strings not contained in the training set. Clearly, the three distinct clusters that emerged during training, diffuse and blend with one another when the test set is presented. Thus, although this model is relatively successful in learning to emulate FSMs, the continuous internal

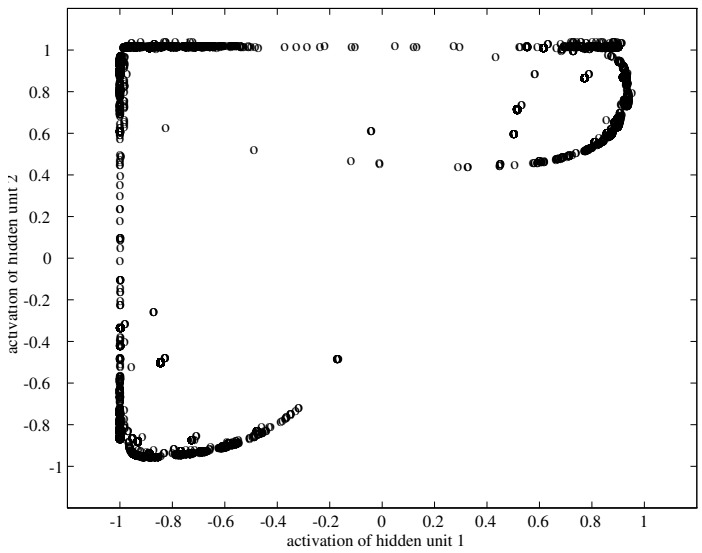

Figure 4: A scatter plot of the hidden activity space for a generic recurrent network trained on the language $(10)^*$, when strings not contained in the training set are presented. The three distinct clusters that had emerged in conjunction with the training set, diffuse and blend with one another when the test set with longer strings is presented.

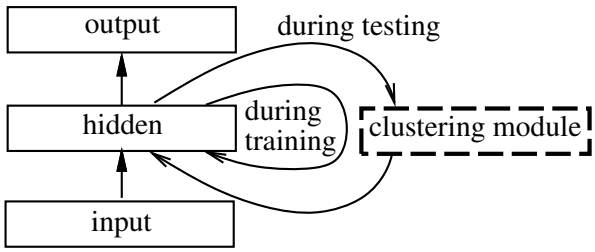

Figure 5: By quantizing hidden activity patterns during the operation of the network, one can hope to rectify the problem of drifting activations.

state dynamics of this network are not well matched to the discrete behavior of FSMs.

3 Prior research

To achieve more robust dynamics, one might consider quantizing the hidden state. Two approaches to quantization have been explored previously. In the first, a net is trained in the manner described above. After training, the hidden state space is partitioned into disjoint regions and each hidden activity pattern is then discretized by mapping it to the center of its corresponding region (Das & Das, 1991; Giles et al., 1992). The basic archtecture is depicted in Figure 5. Das and Das (1991) pursued this approach using a clustering scheme to define the regions. Giles et al. (1992) evenly segmented the activation space, and each segment with non-zero membership was considered a state.

In a second approach, quantization is enforced *during* training by mapping the hidden state at each time step to the nearest corner of a $[0, 1]^n$ hypercube (Zeng, Goodman, & Smyth, 1993). During the testing phase, a hidden activity pattern is discretized by mapping it to the center of the region to which it belongs.

Each of these approaches has its limitations. In the first approach, because learning does not consider the latter quantization, the hidden activity patterns that result from learning may not lie in natural clusters. Consequently, the quantization step may not group together activity patterns

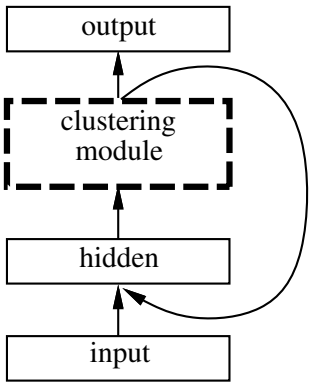

Figure 6: The basic architecture of DOLCE

that correspond to the same state. In the second approach, the quantization process causes the error surface to have discontinuities and to be flat in local neighborhoods of the weight space. Hence, gradient descent learning algorithms cannot be used; instead, even more heuristic approaches are required. To overcome the limitations of these approaches, we have pursued an approach in which quantization is an integral part of the learning process.

4 DOLCE

Our approach incorporates a *clustering module* into the recurrent network architecture, as shown in Figure 6. The hidden unit activities are processed by the clustering module before being passed on to other layers of the network. The clustering module attempts to map regions in hidden activity space to a single point in the same space, effectively partitioning or clustering the space. As with the models described above, each region corresponds to an induced state of the underlying state machine. The regions are adaptive and dynamic, changing over the course of learning. This architecture is called DOLCE, for dynamic on-line clustering and state extraction.

An architecture like DOLCE may be explored along two dimensions: (1) the clustering algorithm used (e.g., the Forgy algorithm, a Gaussian mixture model, the Ball and Hall, 1966, ISODATA algorithm, vector quantization methods), and (2) whether supervised or unsupervised training is used

to identify the clusters. In unsupervised mode, the performance error on the FSM induction task has no effect on the operation of the clustering algorithm; instead, an internal criterion characterizes goodness of clusters. In supervised mode, the primary measure that affects the goodness of a cluster is the performance error. Regardless of the training mode, all clustering algorithms incorporate a pressure to produce a small number of clusters. Additionally, as we elaborate below, the algorithms must allow for soft clustering during training, in order to be integrated into a gradient-based learning procedure.

We have explored two possibilities along the two dimensions for incorporating the clustering module. The first involves the use of Forgy's algorithm in an unsupervised mode; we call this model DOLCE$_u$. The second uses a Gaussian mixture model in a supervised mode, where the mixture model parameters are adjusted so as to minimize the performance error; we call this model DOLCE$_s$.

5 DOLCE$_u$: Inducing finite-state machines using unsupervised learning

The model DOLCE$_u$ continually quantizes the internal state representation using an adaptive clustering scheme that determines both the number of clusters and the partitioning of the state space during training. DOLCE$_u$ then takes as its internal state an interpolation between the actual and quantized states, gradually weighting the interpolation more toward the quantized state as learning progresses. We call this *α-projection*, where α is the weighting factor. Thus, in contrast to prior research, DOLCE$_u$ *gradually* becomes more discrete with training.

DOLCE$_u$ uses a variant of Forgy's (1965) clustering algorithm. The simplicity of Forgy's algorithm was the primary reason for preferring it over other versatile but computationally intensive algorithms, such as ISODATA. Figure 7 outlines the main steps in the version of Forgy's algorithm used in DOLCE$_u$.

1. **iter** = 1, **pat** = 1
2. FORM A NEW CLUSTER with centroid = $\mathbf{x}_{pat}$
3. COMPUTE AVERAGE DISTANCE of $\mathbf{x}_{pat}$ from the centroids of k clusters $\{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_k\}$, i.e.,
$$\overline{d}_{pat} = (1/k) \sum_{j=1}^{k} d(\mathbf{x}_{pat}, \mathbf{c}_j)$$
4. FORM A NEW CLUSTER with centroid = $\mathbf{x}_{pat}$ if $|d(\mathbf{x}_{pat}, \mathbf{c}_i) - \overline{d}_{pat}| > \overline{d}_{pat} T$, where $\mathbf{c}_i$ is the centroid of the cluster closest to $\mathbf{x}_{pat}$, and T is a predefined parameter.
goto step 6 if new cluster was formed succesfully, else goto step 5.
5. CATEGORIZE $\mathbf{x}_{pat}$ into its closest cluster and COMPUTE the cluster's new centroid.
6. **pat** = **pat** + 1; goto step 3.
7. **iter** = **iter** + 1; **pat** = 1 if cluster centers have settled down or an upper bound on **iter** is reached then goto step 8 else goto step 3.
8. STOP

Figure 7: An outline of Forgy's algorithm. The value of k indicates the number of clusters identified by the algorithm. The algorithm begins with $k = 1$, and $\mathbf{c}_1$ equal to the centroid of all the patterns.

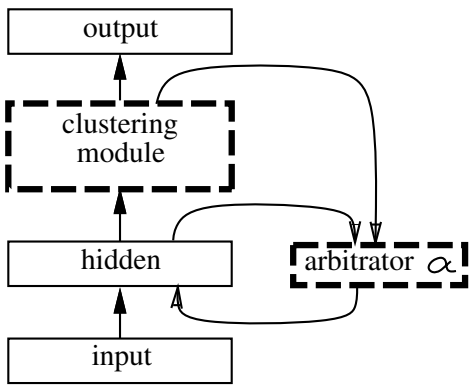

Figure 8: The architecture of DOLCE$_u$

5.1 The architecture

The basic neural network architecture for DOLCE$_u$ is same as Figure 2. However, instead of having standard connectivity between the input (including the context layer) and the hidden layer, the connections into the hidden layer in DOLCE$_u$ are of the second order (inspired by Giles et al., 1992), meaning that the net input to a hidden unit is described by the equation:

$$h_i(t) = \sum_j \sum_k w_{ijk} h_j(t-1) x_k$$

where $h_i(t)$ represents the activity of the i-th hidden unit at time t, x_k represents the activity of the k-th input unit, and w_{ijk} represents the second order weight that connects the pair $h_j(t-1)$ and i_k to $h_i(t)$. The hidden layer is connected to the output layer with standard first order weights. The architecture also includes a clustering module (Figure 8). All connections are feedforward except for the ones that feedback from the hidden layer to itself. At every time step, the hidden layer receives two sources of input: (a) external input representing the current symbol of the input string; and (b) internal input that results from α-projection of the previous hidden state (described in the next section).

The task of the network is to infer the FSM underlying a given set of positive and negative examples. Strings from a regular language are presented, one symbol at a time, as input. Once a complete string has been presented, the network's target output is $+1$ for positive examples, -1 otherwise. The usual sum squared difference between output and target provides the error signal for a given string.

5.2 α-Projection

On-line clustering is incorporated in the network as follows. During every u epochs while training, the hidden activity state is collected at each time step for each training string. The Forgy algorithm is then run on this set of vectors to determine the appropriate number of clusters and the

partitioning of the state vectors into clusters. This results in a cluster membership function that maps any state, $\mathbf{h}$, to a corresponding cluster, $cluster(\mathbf{h})$. For the next u epochs, the cluster membership function is applied to each hidden state at each time t, $\mathbf{h}(t)$, to determine its corresponding partition, $cluster(\mathbf{h}(t))$. The new hidden state, $\hat{\mathbf{h}}(t)$, is found by interpolating the actual hidden state and the centroid of the state's cluster, $\mathbf{c}_{cluster(\mathbf{h}(t))}$:

$$\hat{\mathbf{h}}(t) = (1-\alpha)\mathbf{h}(t) + \alpha \mathbf{c}_{cluster(\mathbf{h}(t))} \tag{1}$$

This state is called the α-*projected* hidden state. The parameter α determines the degree to which DOLCE$_u$ will quantize the states. At early stages of training, when hidden states are volatile and relatively uninformative, the cluster centroids will be noisy and poor indicators of the target internal states. Thus, as training proceeds, it makes sense to increase α gradually, and shift the weighting so as to prefer the quantized state over the raw state. We set α based on the training error:

$$\alpha = e^{-\phi \mathcal{E}},$$

where $\mathcal{E}$ is the average error per string scaled to the range $[0, 1]$ and ϕ is a constant that determines how sensitive is α with respect to the error $\mathcal{E}$. The value of ϕ was determined experimentally and set to 60 (the insensitivity of values of ϕ is discussed later). For values much smaller and larger than this range, α tends to become invariant of $\mathcal{E}$ (α approaches 1 and 0 respectively) thus degrading the performance.

5.3 Test problems

The network was trained on a set of regular languages, listed in Table 1, that are defined on the symbols 0 and 1. These languages were first studied by Tomita (1982) and later used by others, including Giles et al. (1992), Gori, Maggini, and Soda (1993), Pollack (1991), Watrous and Kuhn (1992).

A fixed training corpus of strings was randomly generated for each language, with an equal

Table 1: The Tomita languages

Language	Definition
1	1^*
2	$(10)^*$
3	string does not contain $1^{2n+1}0^{2m+1}$ as a substring
4	string does not contain 000 as a substring
5	string contains an even number of 01's and 10's
6	number of 0's – number of 1's is a multiple of 3
7	$0^*1^*0^*1^*$

number of positive and negative examples. The maximum string length varied from 5 to 10 symbols and the total number of examples varied from 40 to 100, depending on the difficulty of the induction task. Table 2 lists the maximum string length and the number of strings used to learn the Tomita languages listed in Table 1.

5.4 Simulation details

Because all languages had just two symbols, only one input unit was needed; it took on the value 0 or 1 depending on the current input symbol. One output unit was used to represent the judgement of whether or not a string was a member of the language. For each language, we started the simulation with two hidden units and increased the number until there were enough hidden units to learn the training data. The number of hidden units that were necessary to learn the languages are listed in Table 3.

Networks were initialized with random weights in the range $[.25, -.25]$. There are three parameters in DOLCE$_u$: T, ϕ, and u. The values of T and u were set to .5 and 5 respectively. These magnitudes were obtained by experimentation and small variations did not affect DOLCE$_u$'s perfor-

Table 2: Particulars about the training corpus

Language	Maximum string length	Total number of examples
1	5	60
2	8	45
3	7	100
4	10	100
5	10	80
6	6	60
7	10	100

Table 3: Number of hidden units that were necessary to learn the Tomita languages

Language	Number of hidden units
1	2
2	3
3	4
4	4
5	3
6	3
7	4

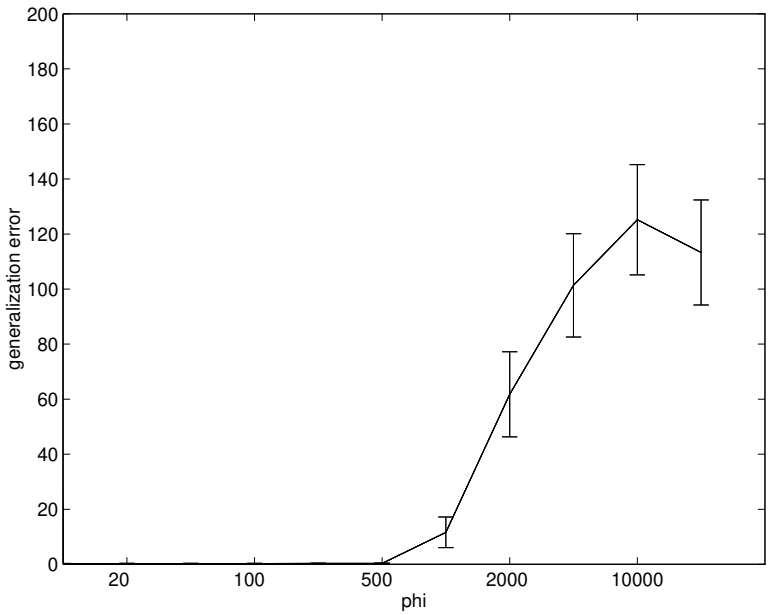

Figure 9: Mean and standard deviation of *generalization error* (average of 10 runs) is plotted against choice of ϕ for Tomita language 2. For values of ϕ in the order of 10 or less, the network does not converge on the training data, hence have been omitted from this graph.

mance. For example, Figure 9 shows the insensitivity of DOLCE$_u$ to the parameter ϕ.

Each string was presented one symbol at a time, after which DOLCE$_u$ was given a target output indicating whether the string was a positive or negative example of the language. Back propagation of error through time (Rumelhart, Hinton, and Williams, 1986) was used to train the network weights. Training continued until DOLCE$_u$ converged on a set of weights. Since the training examples were noise-free (i.e., there were no misclassifications in the training corpus), the error $\mathcal{E}$ on the training set should go to zero when DOLCE$_u$ has successfully learned. If this did not happen on a particular training run, the simulations were restarted with different initial random weights.

While discussing the training process, we point out that due to the discontinuous nature of the expression for $\hat{\mathbf{h}}(t)$ in equation 1, complete gradient descent could not be performed through DOLCE$_u$ because gradient through the clustering algorithm could not be computed. Therefore, we

only propagated error back through the portion of the α-projected hidden state that came from the original hidden state. We therefore assumed that any small changes to the hidden state would not affect the cluster to which a state belonged. The consequence of this assumption is that as the network learns, α goes to 1, and less error is propagated back through the net.

We describe the results from DOLCE$_u$ following the discussion of a second variant of the DOLCE architecture.

6 DOLCE$_s$: Inducing finite-state machines using supervised learning

The basic neural network architecture of DOLCE$_s$ is the same as that of DOLCE$_u$, including the second order weights connecting the input and the hidden layers. However, unlike DOLCE$_u$'s unsupervised clustering scheme, DOLCE$_s$ uses a Gaussian mixture model in conjunction with supervised learning to quantize the hidden activity space. Here we motivate the incorporation of a Gaussian mixture model into DOLCE$_s$, using an argument that gives the approach a solid theoretical foundation. Several assumptions underly the approach. First, we assume that the task faced by DOLCE$_s$ is such that it requires a finite set of internal or *true* states, $\mathbf{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_T\}$. This is simply the premise that motivates this line of work. Second, we assume that any observed hidden state—i.e., a hidden activity pattern that results from presentation of a symbol sequence—belongs to $\mathbf{C}$ but has been corrupted by noise due to inaccuracy in the network weights. Third, we assume that this noise is Gaussian and decreases as learning progresses (i.e., as the weights are adjusted to better perform the task). These assumptions are depicted in Figure 10.

Based on these assumptions, we construct a Gaussian mixture distribution that models the observed hidden states:

$$\mathrm{p}(\mathbf{h}|\mathbf{C}, \boldsymbol{\sigma}, \mathbf{q}) = \sum_{i=1}^{T} \frac{q_i}{(2\pi\sigma_i^2)^{H/2}} e^{-|\mathbf{h}-\mathbf{c}_i|^2/2\sigma_i^2}$$

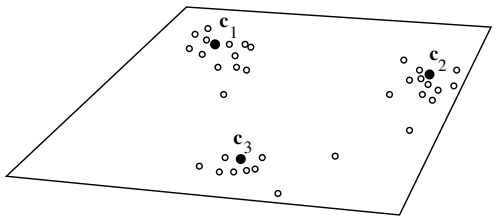

Figure 10: A typical example of a two dimensional hidden activity space. The true states needed to perform the task are $\mathbf{c}_1$, $\mathbf{c}_2$, and $\mathbf{c}_3$ (closed circles), while the observed hidden states (open circles), assumed to be corrupted by noise, have a roughly Gaussian distribution about the $\mathbf{c}_i$.

where $\mathbf{h}$ denotes an observed hidden state, σ_i^2 the variance of the noise that corrupts state $\mathbf{c}_i$, q_i is the prior probability that the true state is $\mathbf{c}_i$, and H is the dimensionality of the hidden state space. For pedagogical purposes, assume for the time being that the parameters of the mixture distribution—T, $\mathbf{C}$, $\boldsymbol{\sigma}$, and $\mathbf{q}$—are known; in a later section we discuss how these parameters are determined.

Given a noisy observed hidden state, $\mathbf{h}$, DOLCE$_s$ computes the maximum a posteriori (MAP) estimator of $\mathbf{h}$ in $\mathbf{C}$. This estimator then replaces the noisy state and is used in all subsequent computation. The MAP estimator, $\hat{\mathbf{h}}$, is computed as follows. The probability of an observed state $\mathbf{h}$ being generated by a given true state i is

$$\mathrm{p}(\mathbf{h}|\text{true state } i) = (2\pi\sigma_i^2)^{-\frac{H}{2}} e^{-|\mathbf{h}-\mathbf{c}_i|^2/2\sigma_i^2}.$$

Using Bayes' rule, one can compute the posterior probability of true state i, given that $\mathbf{h}$ has been observed:

$$\mathrm{p}(\text{true state } i|\mathbf{h}) = \frac{\mathrm{p}(\mathbf{h}|\text{true state } i)q_i}{\sum_j \mathrm{p}(\mathbf{h}|\text{true state } j)q_j}$$

Finally, the MAP estimator is given by $\hat{\mathbf{h}} = \mathbf{c}_{best}$, where $best = \mathrm{argmax}_i \mathrm{p}(\text{true state } i|\mathbf{h})$.

However, because the gradient descent training procedure requires that DOLCE$_s$'s dynamics be differentiable, we use a "soft" version of MAP which involves using $\bar{\mathbf{h}} = \sum_i \mathbf{c}_i \mathrm{p}(\text{true state } i|\mathbf{h})$ instead of $\hat{\mathbf{h}}$ and incorporate a "temperature" parameter into σ_i. The justification and motivation

for this approach is described below.

An important parameter in the mixture model is T, the number of true states (Gaussian bumps). Because T directly corresponds to the number of states in the target FSM, if T is chosen too small, DOLCE$_s$ would not have the capacity to emulate the FSM. Consequently, we set T to a large value and include a technique, described below, for eliminating unnecessary true states. (If the initially selected T is not large enough, the training procedure will not converge to zero error on the training set, and the procedure can be restarted with a larger value of T.)

6.1 Training methodology

The network weights and mixture model parameters—$\mathbf{C}$, $\boldsymbol{\sigma}$, and $\mathbf{q}$—are adjusted by gradient descent in a cost measure, $\mathcal{C}$. This cost includes two components: (a) the performance error, $\mathcal{E}$, which is a squared difference between the actual and target network output following presentation of a training string, and (b) a complexity cost, which is the entropy of the prior distribution, $\mathbf{q}$:

$$\mathcal{C} = \mathcal{E} - \lambda \sum q_i \ln q_i$$

where λ is a regularization parameter. The complexity cost is minimized when only one Gaussian has a nonzero prior, and maximized when all priors are equal. Hence, the cost encourages unnecessary components to drop out of the mixture model.

The particular gradient descent procedure used is a generalization of back propagation through time that incorporates the mixture model. To better condition the search space and to avoid a constrained search, optimization is performed not over $\boldsymbol{\sigma}$ and $\mathbf{q}$ directly but rather over hyperparameters $\mathbf{a}$ and $\mathbf{b}$, where $\sigma_i^2 = \exp(a_i)/\beta$ and $q_i = \exp(-b_i^2)/\sum_j \exp(-b_j^2)$. The expression for q_i in terms of b was inspired by Nowlan and Hinton (1990). The global parameter β scales the overall spread of the Gaussians, which corresponds to the level of noise in the model. As performance on the training set improves, we assume that the network weights are coming to better approximate the target weights, hence the level of noise is decreasing. Thus, β is tied to the performance error

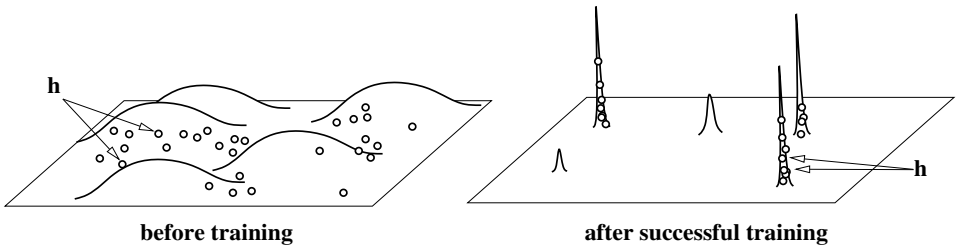

Figure 11: A schematic depiction of the hidden activity space before and after training. The horizontal plane represents the space. The bumps indicate the component Gaussian probability densities that form the mixture model. Observed hidden states are represented by open circles. With training, the Gaussian spreads decrease. The shorter spikes on the right represent the fact that some Gaussians acquire near-zero priors and hence contribute little to the mixture model.

$\mathcal{E}$. We have used various annealing schedules and DOLCE$_s$ appears robust to this variation. In the results below, β is set to $100/\mathcal{E}$. Note that as $\mathcal{E} \to 0$, $\beta \to \infty$ and the probability density under one Gaussian at h will become infinitely greater than the density under any other; consequently, the soft MAP estimator, $\bar{\mathbf{h}}$, becomes equivalent to the MAP estimator $\hat{\mathbf{h}}$, and the transformed hidden state becomes discrete. A schematic depiction of the probability landscape both before and after training is depicted in Figure 11.

6.2 Simulation details

As with DOLCE$_u$, DOLCE$_s$ required a single input unit representing the current input symbol—0 or 1—and one output unit representing the grammaticality judgement. DOLCE$_s$ was trained on the same set of languages studied for DOLCE$_u$.

The training procedure, the number of hidden units, and the initial weight ranges were identical to the previous set of simulations with DOLCE$_u$. At the start of training, each Gaussian center, $\mathbf{c}_i$, is initialized to a random location in the hidden state space. The standard deviations of each Gaussian, σ_i, are initially set to a large value. The priors, q_i, are set to $1/T$. The network weights

are set to initial values chosen from the uniform distribution in [-.25,.25].

For each language, DOLCE$_s$ was trained with different random initial weights. The learning rate and regularization parameter λ were chosen for each language by quick experimentation with the aim of maximizing the likelihood of convergence on the training set. The parameters λ used in the simulations was of the order 0.1, and the learning rate was of the order 0.0001.

7 Results

Figures 12 and 13 show scatter plots of the hidden activity space for typical simulations on the language $(10)^*$. The initial conditions are identical in the two Figures, but on-line clustering has been removed from the simulation depicted in Figure 12. The Figures show that DOLCE$_s$ helps distinct clusters to evolve over time. A very similar pattern occurs with DOLCE$_u$ (not shown). DOLCE is also able to converge on compact state representations; compare the lower right plots in Figures 12 and 13. Once the network was trained, the state representations could be inferred from the activity patterns of the induced clusters. The transitions among states were inferred by walking a trained network through input patterns that forced it to make all possible transitions. Once the states as well as their transitions were identified, it was trivial to extract the induced FSM. Examples of FSMs inferred by DOLCE are depicted in Figure 14. Approximately 60-70% of the times the simulations converged on a set of weights that yielded zero training error. Occasionally, runs would get stuck in a local optimum and the error would stabilize to a nonzero value, in which case the runs would be restarted.

We have performed a comparative study of DOLCE with three other approaches on the Tomita languages: a generic recurrent network, as shown in Figure 2, which used no clustering (referred to by the abbreviation NC); a version with rigid quantization during training (RQ), comparable to the approach taken by Zeng, Goodman, and Smythe (1993); and a version in which the unsupervised Forgy algorithm was used to quantize the hidden state following training (LQ) (Figure 5),

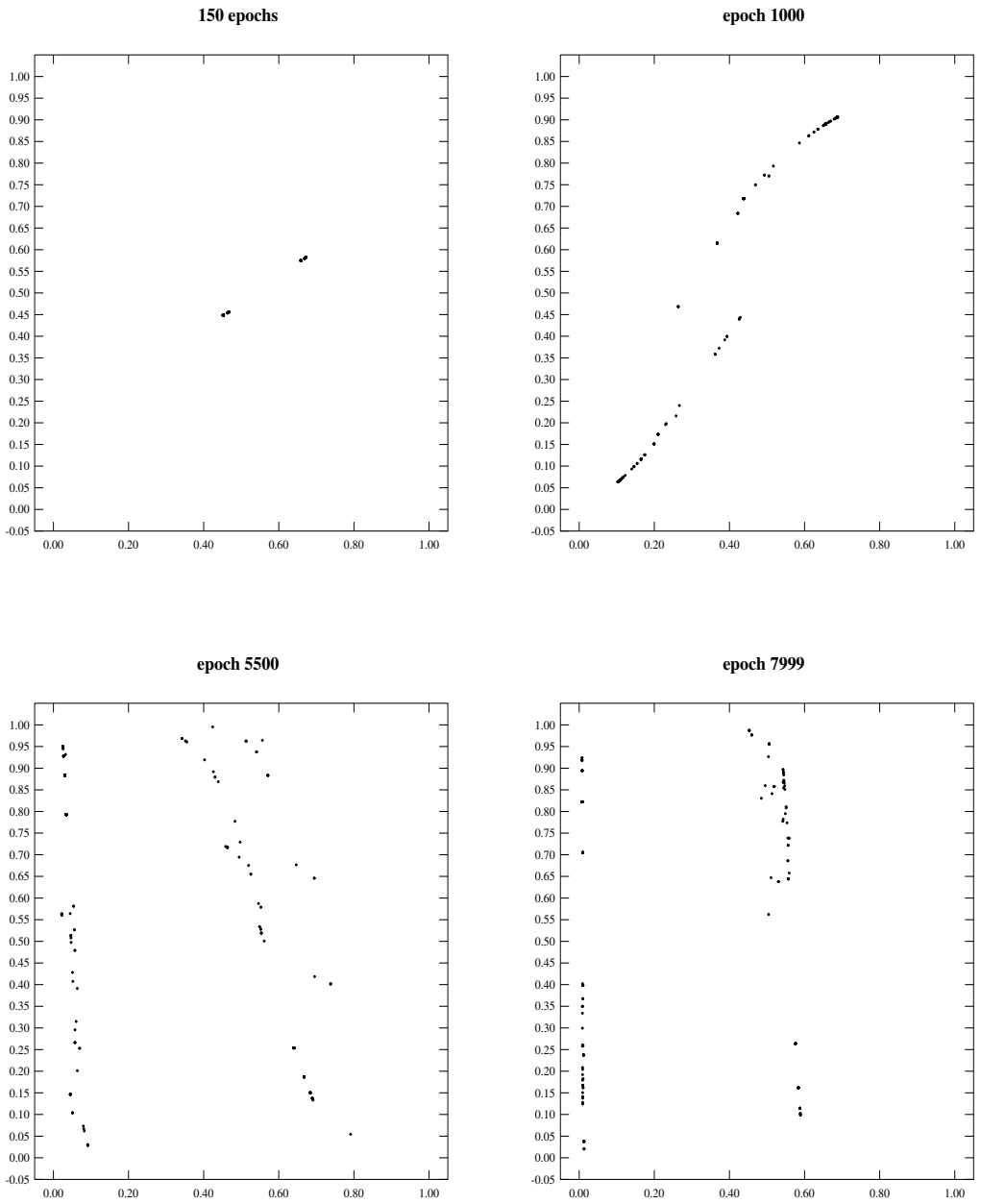

Figure 12: Scatter plot of hidden unit activities at various stages of training for DOLCE$_s$ learning Tomita language 2 *with on-line clustering removed.* The two axes represent the activities of the two hidden units in the network One point is plotted for each time step of each training example, for a total of 240 points, although many of these points are superimposed on one another. The four scatterplots represent the network state after 150, 100, 5500, and 8000 training epochs.

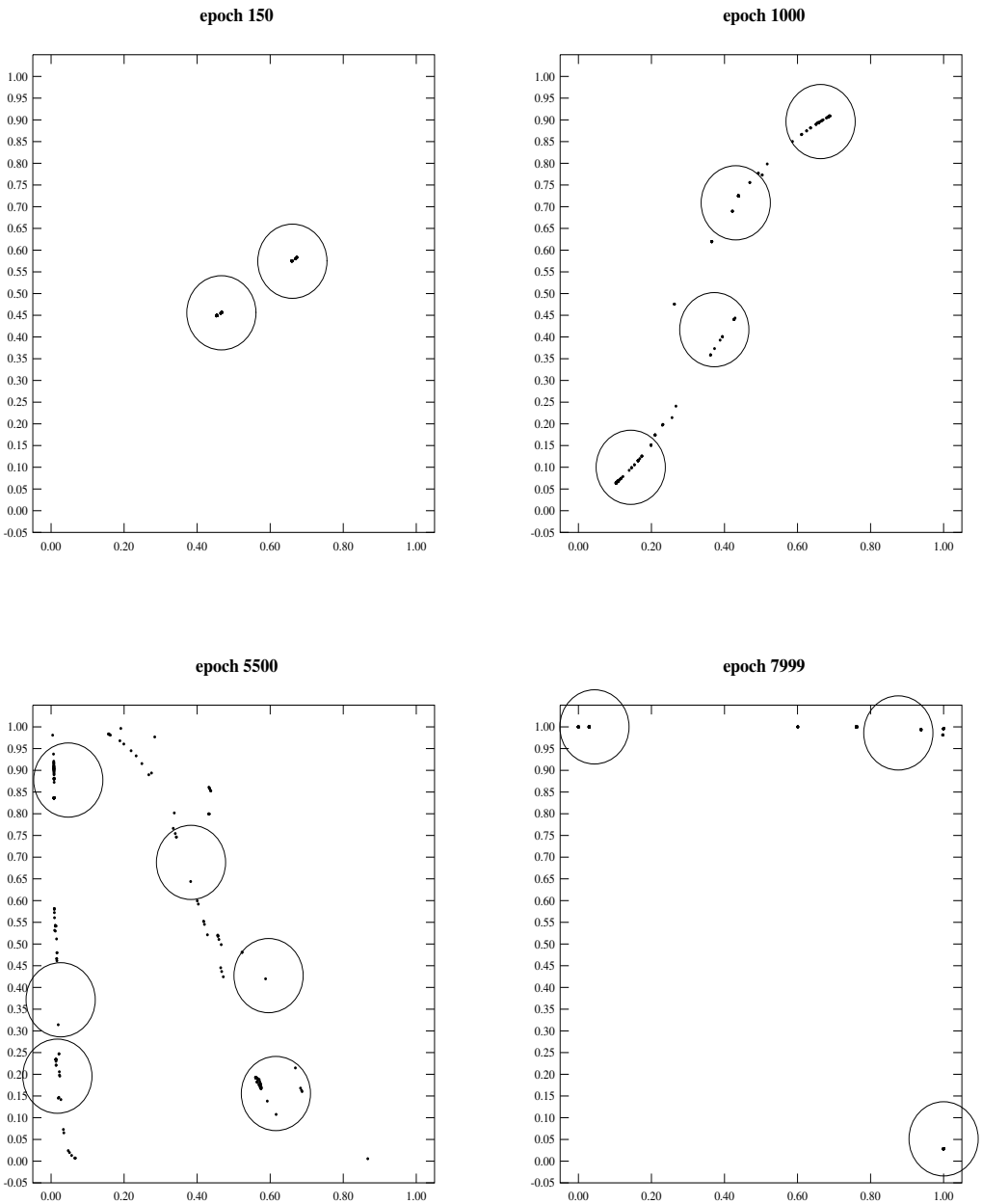

Figure 13: Scatter plot of hidden unit activities at various stages of training for DOLCE$_s$ learning Tomita language 2. The network is started from the same initial conditions as in Figure 13. As in the previous figure, one point is plotted for each time step of each training example, for a total of 240 points, although many of these points are superimposed on one another. The large open circles indicate the cluster centers.

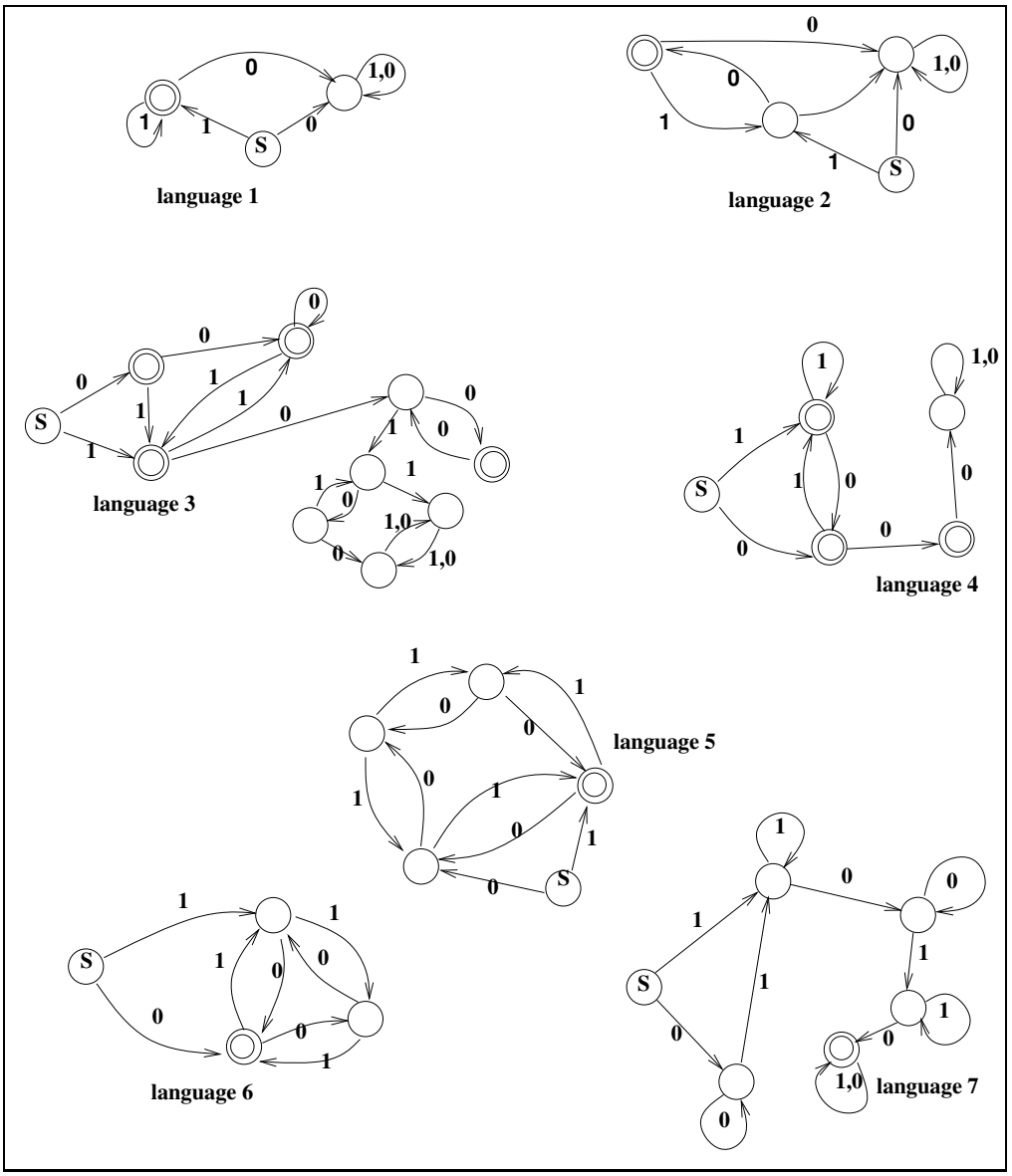

Figure 14: Examples of FSMs inferred by DOLCE for the 7 Tomita languages. Each circle represents a state of the FSM. The circle labeled S is the start state. Double circles represent accept states. A bold symbol on an arrow indicates the input causing the state transition represented by the arrow.

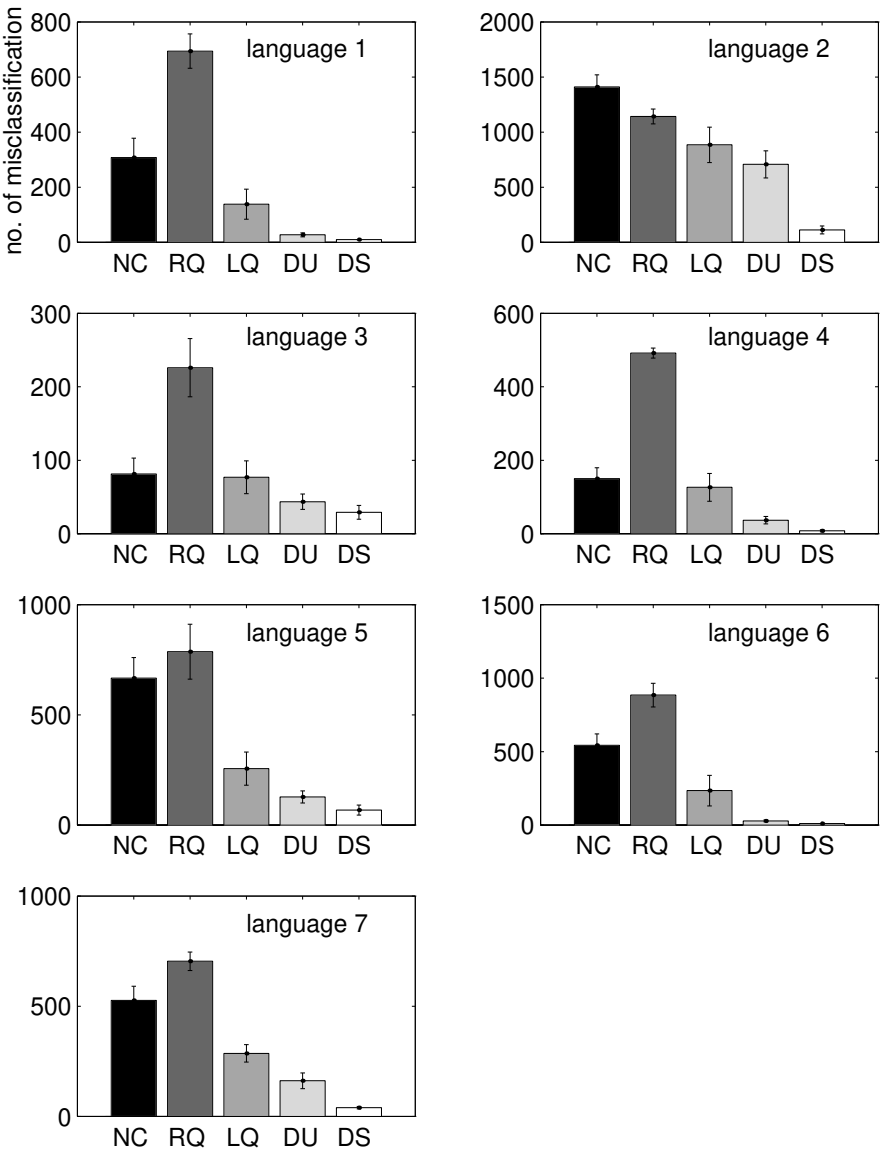

Figure 15: Each bar graph shows generalization performance on one of the Tomita languages for five alternative neural network approaches: no clustering (NC), rigid quantization (RQ), learn then quantize (LQ), DOLCE in unsupervised mode using Forgy's algorithm (DU), DOLCE in supervised mode using a mixture model (DS). The vertical axis shows the number of misclassifications for the 3000 test strings. Each bar is the average result across ten replications with different initial weights. The error bars represent one standard deviation of the mean.

comparable to the earlier work of Das and Das (1991). In these alternative approaches, we used the same architecture as DOLCE except for the clustering procedure. We selected learning parameters to optimize performance on the training set, ran ten replications for each language, replaced runs which did not converge, and used the same training sets for all architectures.

Figure 15 compares the generalization performance of DOLCE$_u$ (DU) and DOLCE$_s$ (DS) to the NC, RQ, and LQ approaches. Generalization performance was tested using 3000 strings not part of the training set, half positive examples and half negative. Both versions of DOLCE outperformed the alternative neural network approaches, and DOLCE$_s$ consistently outperformed DOLCE$_u$.

8 Conclusions

We have described an approach for incorporating bias into a learning algorithm in order to encourage discrete representations to emerge during training. This approach is quite general and can be applied to any domain in which discrete representations are desirable, not just FSM induction and grammaticality judgement. Also, this approach is not specific to recurrent networks; it would work equally well with feedforward networks. Although the technique of incorporating bias appears very attractive and seems to work well on simple problems, it remains to be seen how well it performs on real-world problems. Nonetheless, we believe that if a task is sufficiently well understood and the nature of the problem domain can be clearly identified, it is safe to say that incorporating the knowledge in an appropriate way can only help in the learning process.

9 Acknowledgements

This research was supported by NSF Presidential Young Investigator award IRI–9058450, and McDonnell-Pew Award # 97-18.

References

Ball, G. and Hall, D. (1966). ISODATA: a novel method of data analysis and pattern classification. Technical report, Stanford Research Institute, Menlo Park, California.

Das, S. and Das, R. (1991). Induction of discrete state-machine by stabilizing a continuous recurrent network using clustering. *Computer Science and Informatics*, 21(2):35–40. Special Issue on Neural Computing.

Das, S. and Mozer, M. (1994). A hybrid clustering/gradient descent architecture for finite state machine induction. In Cowan, J., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 19–26. San Mateo, CA: Morgan Kaufmann.

Elman, J. (1990). Finding structure in time. *Cognitive Science*, 14(2).

Forgy, E. (1965). Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768.

Giles, C., Miller, C. B., Chen, H., Sun, G., and Lee, Y. (1992). Learning and extracting finite state automata with second-order recurrent neural network. *Neural Computation*, 4(3):393–405.

McMillan, C., Mozer, M., and Smolensky, P. (1992). Rule induction through integrated symbolic and subsymbolic processing. In Moody, J., Hanson, S., and Lippmann, R., editors, *Advances in Neural Information Processing Systems 4*, pages 969–976. San Mateo, CA: Morgan Kaufmann.

Mozer, M. and Bachrach, J. (1990). Discovering the structure of a reactive environment by exploration. *Neural Computation*, 2(4):447–457.

Mozer, M. and Das, S. (1993). A connectionist symbol manipulator that discovers the structure of context-free languages. In Giles, C., Hanson, S., and Cowan, J., editors, *Advances in Neural Information Processing Systems 5*, pages 863–870. San Mateo, CA: Morgan Kaufmann.

Nowlan, S. and Hinton, G. (1992). Simplifying neural networks by soft weight-sharing. *Neural computation*, 4(4):473.

Pollack, J. (1991). The induction of dynamical recognizers. *Machine Learning*, 7(2/3):123–148.

Schütze, H. (1993). Word space. In Giles, C., Hanson, S., and Cowan, J., editors, *Advances in Neural Information Processing Systems 5*, pages 895–902. San Mateo, CA: Morgan Kaufmann.

Servan-Schreiber, D., Cleeremans, A., and Mcclelland, J. (1991). Graded state machines: the representation of temporal contingencies in simple recurrent network. *Machine Learning*, 7(2/3):161–193.

Towell, G. and Shavlik, J. (1992). Interpretion of artificial neural networks: mapping knowledge-based neural networks into rules. In Touretzky, D., editor, *Advances in Neural Information Processing Systems 2*, pages 977–984. San Mateo, CA: Morgan Kaufmann.

Watrous, R. and Kuhn, G. (1992). Induction of finite state languages using second-order recurrent networks. In Moody, J., Hanson, S., and Lippmann, R., editors, *Advances in Neural Information Processing Systems 4*, pages 309–316, San Mateo, CA. Morgan Kaufmann Publishers.

Zeng, Z., Goodman, R., and Smythe, P. (1993). Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, 5(6):976–990.