Gatsby Computational Neuroscience Unit
University College London
http://www.gatsby.ucl.ac.uk

17 Queen Square, London
WC1N 3AR, United Kingdom
+44 20 7679 1176

Funded in part by the Gatsby Charitable Foundation.

March 18, 2000

GCNU TR 2000–002

Learning Distributed Representations of Concepts using Linear Relational Embedding

Alberto Paccanaro **Geoffrey Hinton**

Gatsby Unit

Abstract

In this paper we introduce Linear Relational Embedding as a means of learning a distributed representation of concepts from data consisting of binary relations between concepts. The key idea is to represent concepts as vectors, binary relations as matrices, and the operation of applying a relation to a concept as a matrix-vector multiplication that produces an approximation to the related concept. A representation for concepts and relations is learned by maximizing an appropriate discriminative goodness function using gradient ascent. On a task involving family relationships, learning is fast and leads to good generalization.

Learning Distributed Representations of Concepts using Linear Relational Embedding

Alberto Paccanaro **Geoffrey Hinton**

Gatsby Unit

1 Introduction

Given data which consists of concepts and relations among concepts, our goal is to correctly predict unobserved instances of relationships between concepts. We do this by representing each concept as a vector in a Euclidean space and the relationships between concepts as linear operations.

To illustrate the approach, we start with a very simple task which we call the *number problem*. The data consists of integers and operations among integers. In the *modular* number problem the numbers are integers in the set $V = [0 \ldots m-1]$ and the set of operations is $\mathcal{R} = \{+1, -1, +2, -2, +3, -3, +4, -4, +0\}_m$, where the subscript indicates that the operations are performed modulo m. The data then consists of all or some of the triplets (num_1, op, num_2) where $num_1, num_2 \in V$, $op \in \mathcal{R}$, and num_2 is the result of applying operation op to number num_1; for example, for $m = 10$, $\{(1, +1, 2), (4, +3, 7), (9, +3, 2), \cdots\}$.

The main idea in Linear Relational Embedding (LRE) is to represent concepts using n-dimensional vectors, relations as $(n \times n)$ matrices, and the operation of applying a relation to a concept (to obtain another concept) as a matrix-vector multiplication. Within this framework, one could easily hand-code a solution for the number problem with $n = 2$ and $m = 10$, where the numbers are represented by vectors having unit length and disposed as in fig.1a, while relations are represented by rotation matrices $R(\Theta)$, where the rotation angle Θ is a multiple of $2\pi/10$ (first row of table 1). The result of applying, for example, operation $+3$ to number 4, is obtained by multiplying the corresponding matrix and vector, which amounts to rotating the vector located at 144 degrees by 108 degrees, thus obtaining the vector at 252 degrees, which corresponds exactly to the vector representing the number 7.

In this paper, we show how LRE finds an equivalent solution, which is presented in fig. 1b and in the second row of table 1. LRE can find this solution when many of the triplets are omitted from the training set and once it has learned this way of representing the concepts and relationships it can complete all of the omitted triplets correctly. Moreover, LRE works well not only on toy problems like the one presented above, but also in other symbolic domains where the task of generalizing to unobserved triplets is non-trivial.

In the next section we briefly review related work on learning distributed representations. LRE is then presented in detail in section 3. Section 4 presents the results obtained using LRE on the number problem and the family tree task (Hinton, 1986), as well as the results obtained on a much

† The authors would like to thank Peter Dayan, Sam Roweis, Zoubin Ghahramani, Carl van Vreeswijk, Hagai Attias and Marco Buiatti for many useful discussions.

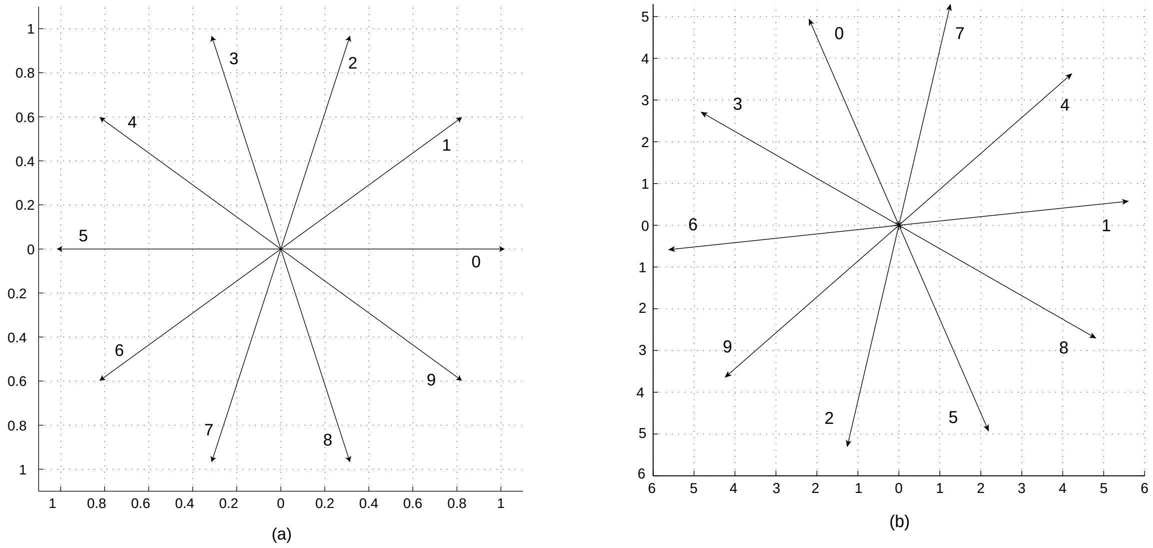

Figure 1: (a) vectors of the hand-coded solution for the number problem when $n = 2$ and $m = 10$ (b) vectors of the solution found by Linear Relational Embedding. Only 70 out of the 90 possible triplets were used for training. During testing, the system was able to correctly complete all the triplets.

OPERATION	-4	-3	-2	-1	+0	+1	+2	+3	+4
Hand-coded Solution	-144	-108	-72	-36	0	36	72	108	144
LRE Solution	72.00	-35.97	-144.01	108.01	0.00	-108.02	144.02	35.98	-71.97

Table 1: Angles, expressed in degrees, of the rotation matrices of the solutions to the number problem with $n = 2$ and $m = 10$, corresponding to the vectors in fig.1. The rotations in the LRE solution differ very slightly from multiples of 36 because only 70 triplets randomly chosen out of the 90 were used during training.

larger version of the family tree problem that uses data from a real family tree. We also compare these results to the results obtained using Principal Components Analysis. In section 5 we examine how a solution obtained from an impoverished data set can be modified to include information about new concepts and relations. Section 6 indicates ways in which LRE could be extended and section 7 presents a final discussion of the method.

2 Related work

Several methods already exist for learning sensible distributed representations from relational data. Multidimensional Scaling (Kruskal, 1964; Young and Hamer, 1987) finds a representation of concepts as vectors in a multi-dimensional space, in such a way that the dissimilarity of two concepts is modeled by the Euclidean distance between their vectors. Unfortunately, dissimilarity is the only relationship used by multidimensional scaling so it cannot make use of the far more specific information about concepts contained in a triplet like “John is the father of Mary”.

Latent Semantic Analysis (LSA) (Deerwester et al., 1990; Landauer and Dumais, 1997; Lan-

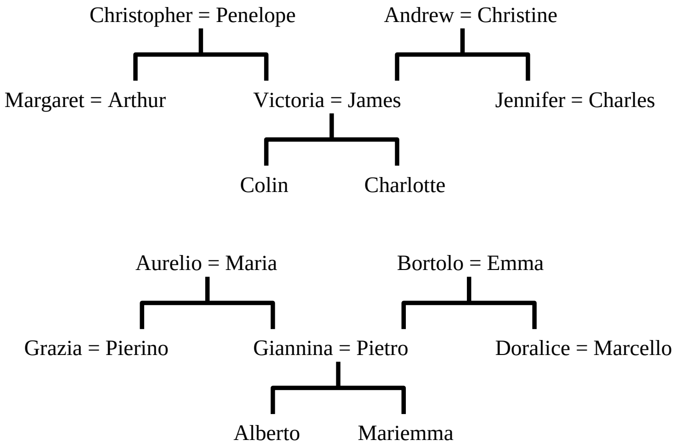

Figure 2: Two isomorphic family trees. The symbol "=" means "married to"

dauer et al., 1998) assumes that the meaning of a word is reflected in the way in which it co-occurs with other words. LSA finds features by performing singular value decomposition on a large matrix and taking the eigenvectors with the largest eigenvalues. Each row of the matrix corresponds to a paragraph of text and the entry in each column is the number of times a particular word occurs in the paragraph or a suitably transformed representation of this count. Each word can then be represented by its projection onto each of the learned features and words with similar meanings will have similar projections. Again, LSA is unable to make use of the specific relational information in a triplet.

Hinton (1986) showed that a multilayer neural network trained using backpropagation (Rumelhart et al., 1986) could make explicit the semantic features of concepts and relations present in the data. Unfortunately, the system had problems in generalizing when many triplets were missing from the training set. This was shown on a simple task called the *family tree problem*. In this problem, the data consists of persons and relations among persons belonging to two families, one Italian and one English, shown in figure 2. All the information in these trees can be represented in simple propositions of the form *(person1, relation, person2)*. Using the relations *father, mother, husband, wife, son, daughter, uncle, aunt, brother, sister, nephew, niece* there are 112 of such triplets in the two trees. The network architecture used by Hinton is shown in fig.3. It had two groups of input units, one for the role *person1* and one for the role *relation*, and one group of output units to represent *person2*. Inputs and outputs were coded so that only one unit was active at the time, standing for a particular person or relation. The idea was that the groups of 6 units on the second and fourth layer should learn important features of persons and relations that would make it easy to express regularities of the domain that were only implicit in the examples given. Figure 4 shows the activity level for input *Colin aunt* in the network after learning. Notice how there are 2 units with a high activation in the output layer, marked by black dots, corresponding to the 2 correct answers, because *Colin* has 2 aunts (*Jennifer* and *Margaret*). Figure 5 shows the diagrams of the weights on the connections from the 24 input units to the 6 units that were used for the network's internal, distributed representation of *person1*, after learning. It is clear that unit number 1 is primarily concerned with the distinction between English and Italian. Unit 2 encodes which generation a person belongs to. Unit 6 encodes which branch of the family a person belongs to. Notice how these semantic features are important for expressing regularities in the domain, but were never explicitly specified. Similarly, relations were encoded in terms of semantic features

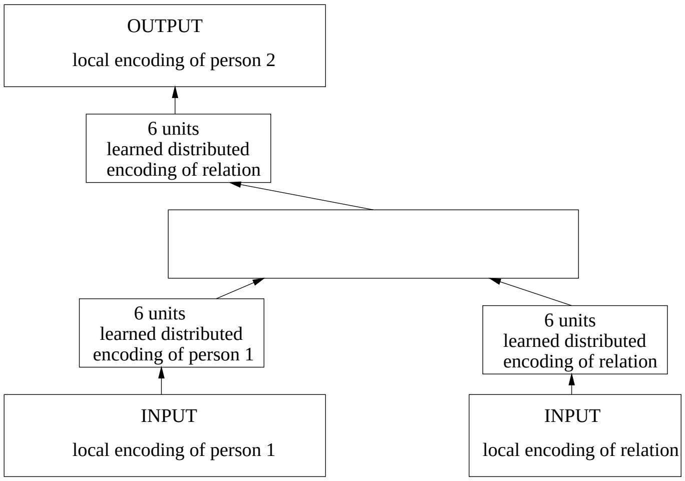

Figure 3: The architecture of the network used for the family tree task. It has three hidden layers of 6 units in which it constructs its own internal representations. The input and output layers are forced to use localist encodings.

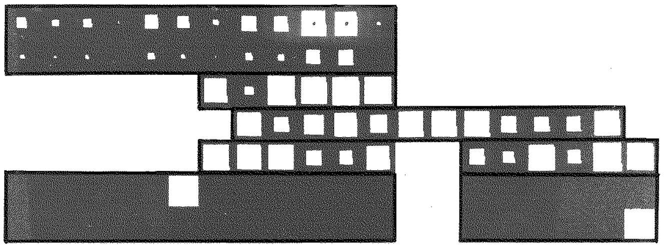

Figure 4: The activity levels in the network after it has learned. The bottom layer has 24 input units on the left for representing *person1* and 12 units on the right for representing the *relation*. The white squares inside these two groups show the activity levels of the units. There is one active unit in the first group (representing *Colin*) and one in the second group (representing *aunt*). Each of the two groups of input units is totally connected to its own group of 6 units in the second layer. These two groups of 6 must encode the input terms as distributed pattern of activity. The second layer is totally connected to the central layer of 12 units, and this layer is connected to the penultimate layer of 6 units. The activity in the penultimate layer must activate the correct output units, each of which stands for a particular person2. In this case, there are two correct answers (marked by black dots) because *Colin* has two aunts. Both the input and the output units are laid out spatially with the English people in one row and the isomorphic Italians immediately below. From Hinton (1986).

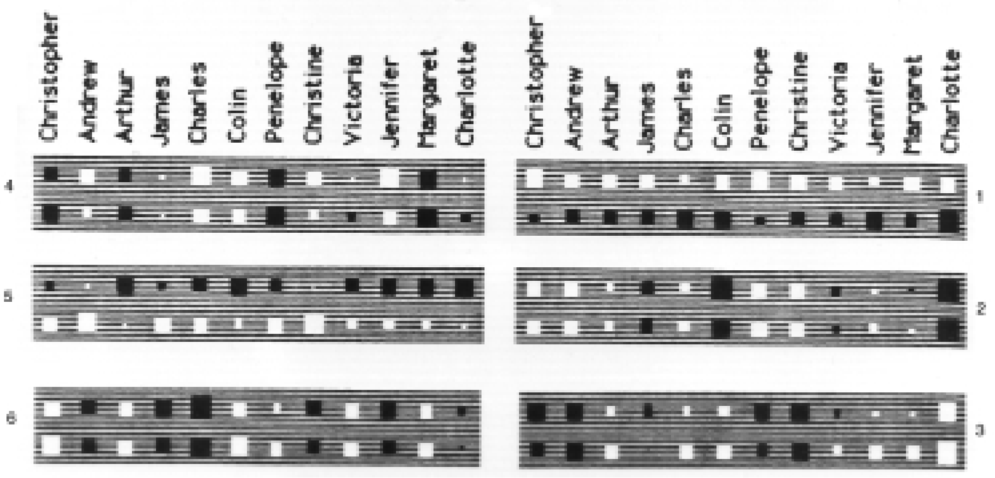

Figure 5: Diagrams of the weights from the 24 input unit that represent people to the 6 units in the second layer that learn distributed representations of people. White rectangles stand for excitatory weights, black for inhibitory weights, and the area of the rectangle encodes the magnitude of the weight. The weights from the 12 English people are in the top row of each unit. Beneath of each of these weights is the weight from the isomorphic Italian. From Hinton (1986).

in the other group of 6 units of layer 2.

The discovery of these semantic features, gave the network some degree of generalization. When tested on four triplets which had not been shown during training, the network was usually able to find the correct answers. Notice how any learning procedure which relied on finding direct correlations between the input and the output vectors, would generalize very badly on the family tree task: the structure that must be discovered to generalize correctly, is not present in the pairwise correlations between input and output units.

The biggest limitation of the system was that generalization was limited, and the system had problems in generalizing when more than 4 triplets were missing from the training set. Moreover there was no guarantee that the semantic features learned for *person1* in the second layer would be the same as the ones found in the fourth layer for *person2*.

The network used by Hinton (1986) is restricted to completing triplets from their first two terms. A more flexible way of applying the backpropagation learning procedure is to have a recurrent network which receives a sequence of words, one at a time, and continually predicts the next word. The states of the hidden units must then learn to capture all the information in the word string that is relevant for predicting the next word. Elman (1990) presented a version of this approach in which the backpropagation through time that is required to get the correct derivatives is curtailed after one time step to simplify the computation. Bridle (1990) showed that the forward dynamics of a particular type of recurrent neural network could be viewed as a way of computing the posterior probabilities of the hidden states of an HMM, and the relationship between recurrent neural networks and Hidden Markov Models has been extensively studied by Cleermans *et. al.* (1989) , Giles *et. al.* (1992) and others.

Hidden Markov Models are interesting because it is tractable to compute the posterior distribution over the hidden states given an observed string. But as a generative model of word strings, they assume that each word is produced by a single hidden node and so they do not seem appropriate if our goal is to learn real-valued distributed representations of the concepts denoted by words.

Linear dynamical systems seem more promising because they assume that each observation is generated from a real-valued vector in the hidden state space. However, the linearity of linear dynamical systems seems very restrictive. This linearity shows up in both the dynamics and the output model:

$$x(t+1) = Rx(t) + \epsilon \tag{1}$$

$$y(t) = Cx(t) + \eta \tag{2}$$

where x is the hidden state, y is the visible state, R is the linear dynamics, C is the linear output model, ϵ is the noise in the dynamics and η is the noise in the output model.

Linear Relational Embedding can be viewed as a way of overcoming these apparent restrictions so that linear dynamical systems can be profitably applied to the task of modeling discrete relational data. First, we eliminate the linear output model by using a discrete observation space and assuming that there is a noise-free table that relates vectors in the hidden state space to discrete observations. The entries in this table change during learning, but with the table fixed the “hidden” state is precisely specified by the observed discrete symbol.

The linearity in the dynamics can be made far less restrictive by using a *switching* linear dynamical system. Instead of treating the relational term in a triplet as an observation produced by the hidden state, we treat it as a completely different kind of observation that provides information about the dynamics, R, rather than the hidden state, x. Again, there is a learned, noise-free table that exactly specifies the linear dynamics associated with each relational term. We allow an additional Gaussian noise process in the dynamics, ϵ. This ensures that there is always some probability density of arriving at any point in the hidden space wherever we start and whatever the dynamics. In particular it ensures that, starting from the point in the hidden space specified by the first term in a triple and using the dynamics specified by the relational term there is some probability of arriving at the point in the hidden space specified by the third term. Learning can then adjust the tables so that the probability of arriving at the point specified by the third term is much greater than the probability of arriving at any of the points that would be specified by other possible third terms.

The linear dynamical systems perspective is useful in understanding how Linear Relational Embedding relates to recurrent neural networks as a proposal for learning from relational data, but LRE is sufficiently different from a standard linear dynamical system that this perspective can also be confusing. In the next section we therefore present LRE as a technique in its own right.

3 Linear Relational Embedding

Let us assume that our data consists of C triplets (*concept1, relation, concept2*) containing N distinct concepts and M binary relations. As anticipated in section 1, the main idea of Linear Relational Embedding is to represent each concept with an n-dimensional vector, and each relation with an $(n \times n)$ matrix. We shall call: $V = \{\mathbf{v}_1, ..., \mathbf{v}_N\}$ the set of vectors, $\mathcal{R} = \{R_1, ..., R_M\}$ the set of matrices and $\mathcal{D} = \{(\mathbf{a}^c, R^c, \mathbf{b}^c)\}_{c=1}^{C}$ the set of all the triplets, where $\mathbf{a}^c, \mathbf{b}^c \in V$ and $R^c \in \mathcal{R}$. The operation that relates a pair $(\mathbf{a}^c, R^c)$ to a vector $\mathbf{b}^c$ is the matrix-vector multiplication, $R^c \cdot \mathbf{a}^c$, which produces an approximation to $\mathbf{b}^c$.

The goal of learning is to find suitable vectors and matrices such that for each triplet $(\mathbf{a}^c, R^c, \mathbf{b}^c) \in \mathcal{D}$, $\mathbf{b}^c$ is the vector closest to $R^c \cdot \mathbf{a}^c$. The obvious approach is to minimize the squared distance between $R^c \cdot \mathbf{a}^c$ and $\mathbf{b}^c$, but this is no good because it causes all of the vectors or matrices to collapse to zero. In addition to minimizing the squared distance to $\mathbf{b}^c$ we must also maximize the squared distances to other concept vectors that are nearby. This can be achieved by imagining that $R^c \cdot \mathbf{a}^c$ is a noisy version of one of the concept vectors and maximizing the probability that it is a noisy version of the correct answer, $\mathbf{b}^c$, rather than any of the other possibilities. If we assume spherical Gaussian noise with a variance of $1/2$ on each dimension, the probability that concept i would generate $R^c \cdot \mathbf{a}^c$ is proportional to $\exp(-||R^c \cdot \mathbf{a}^c - \mathbf{v}_i||^2)$ so a sensible discriminative goodness function is:

$$G = \sum_{c=1}^{C} \frac{1}{K_c} \log \frac{e^{-\|R^c \cdot \mathbf{a}^c - \mathbf{b}^c\|^2}}{\sum\limits_{\mathbf{v}_i \in V} e^{-\|R^c \cdot \mathbf{a}^c - \mathbf{v}_i\|^2}} \tag{3}$$

where K_c is the number of triplets in $\mathcal{D}$ having the first two terms equal to the ones of c, but differing in the third term. To understand why we need to introduce this factor, let us consider a set of K triplets, each having the same first two terms, $\mathbf{a}$ and R, but differing in the third term, which we shall call $\mathbf{b}_i$ with $i = 1 \dots K$. We would like our system to assign equal probability to each of the correct answers, and therefore the discrete probability distribution that we want to approximate can be written as:

$$\mathcal{P}_{\mathbf{x}} = \frac{1}{K} \sum_{i=1}^{K} \delta(\mathbf{b}_i - \mathbf{x}) \tag{4}$$

where δ is the discrete delta function and $\mathbf{x}$ ranges over the vectors in V. Our system implements the discrete distribution:

$$\mathcal{Q}_{\mathbf{x}} = \frac{1}{Z} \exp(-\|R \cdot \mathbf{a} - \mathbf{x}\|^2) \tag{5}$$

where

$$Z = \sum_{\mathbf{v}_i \in V} \exp(-\|R \cdot \mathbf{a} - \mathbf{v}_i\|^2) \tag{6}$$

is the normalization factor. The Kullback-Leibler divergence between $\mathcal{P}_{\mathbf{x}}$ and $\mathcal{Q}_{\mathbf{x}}$, can be written as:

$$KL(\mathcal{P}\|\mathcal{Q}) = \sum_{\mathbf{x}} \mathcal{P}_{\mathbf{x}} \cdot \log \frac{\mathcal{P}_{\mathbf{x}}}{\mathcal{Q}_{\mathbf{x}}} = \sum_{\mathbf{x}} \frac{1}{K} \sum_{i=1}^{K} \delta(\mathbf{b}_i - \mathbf{x}) \cdot \log \frac{\frac{1}{K}\sum\limits_{i=1}^{K} \delta(\mathbf{b}_i - \mathbf{x})}{\frac{1}{Z} \exp(-\|R \cdot \mathbf{a} - \mathbf{x}\|^2)} \tag{7}$$

Thus, minimizing $KL(\mathcal{P}\|\mathcal{Q})$ amounts to minimizing:

$$-\sum_{\mathbf{u}} \frac{1}{K} \cdot \log(\frac{1}{Z} \exp(-\|R \cdot \mathbf{a} - \mathbf{u}\|^2)) \tag{8}$$

for every $\mathbf{u}$ that is a solution to the triplet, which is exactly what we do when we maximize eq.3.

The results which we present in the next section were obtained by maximizing G using gradient ascent. All the vector and matrix components were updated simultaneously at each iteration. One effective method of performing the optimization is scaled conjugate gradient (Møller, 1993). Learning was fast, usually requiring only a few hundred updates and learning virtually ceased as the probability of the correct answer approached 1 for every data point. We have also developed an alternative optimization method which is less likely to get trapped in local optima when the task is difficult. The objective function is modified to include a temperature that divides the exponents in eq. 3. The temperature is annealed during the optimization. This method uses a line search in the direction of steepest ascent of the modified objective function. A small amount of weight decay helps to ensure that the exponents in eq. 3 do not cause numerical problems when the temperature becomes small.

In general, different initial configurations and optimization algorithms caused the system to arrive at different solutions, but these solutions were almost always equivalent in terms of generalization performance.

4 Results

We shall first present the results obtained applying LRE to the number problem and to the family tree problem. After learning a representation for matrices and vectors, we checked, for each triplet c, whether the vector with the smallest Euclidean distance from $R^c \cdot \mathbf{a}^c$ was indeed $\mathbf{b}^c$. We checked both how well the system learned the training set and how well it generalized to unseen triplets. Unless otherwise stated, in all the experiments we optimized the goodness function using scaled conjugate gradient. Two conditions had to be simultaneously met in order for the algorithm to terminate: the absolute difference between the values of the solution at two successive steps had to be less than 10^{-4} and the absolute difference between the objective function values at two successive steps had to be less than 10^{-8}. All the experiments presented here were repeated several times, starting from different initial conditions and randomly splitting training and test data. In general the solutions found were equivalent in terms of generalization performance. The algorithm usually converged within a few hundred iterations, and rarely got stuck in poor local minima.

4.1 Results on the number problem

Let us consider the modular number problem which we saw in section 1. With numbers $[0 \dots 9]$ and operations $\{+1, -1, +2, -2, +3, -3, +4, -4, +0\}_{10}$, there exist 90 triplets *(num1, op, num2)*.

LRE was able to learn all of them correctly using 2-dimensional vectors and matrices ($n = 2$). Figure 1 shows a typical solution that we obtained after training with 70 triplets randomly chosen out of the 90. The scaled conjugate gradient algorithm converged within the desired tolerance in 125 iterations. We see that all the vectors have about the same length, and make an angle of about $2\pi/10$ with each other. The matrices turn out to be approximately orthogonal, with all their row and column vectors having about the same length. Therefore each can be approximately decomposed into a constant factor which multiplies an orthonormal matrix. The degrees of rotation of each orthonormal matrix are shown in the second row of table 1. The matrices' multiplicative factor causes the result of the rotation to be longer than the second vector of the triplet. Because the concept vectors lie at the vertices of a regular polygon centered at the origin, this lengthening increases the squared distance from the incorrect answers by more than it increases the squared distance from the correct answer, thus improving the discriminative goodness function in Eq. 3.

A 2-dimensional matrix has 4 degrees of freedom. The matrices we obtain after learning have only 2 degrees of freedom: the extent of the rotation and the multiplication factor. It is interesting to see how, for this simple problem, LRE often finds appropriate vectors and matrices just by using rotation angles, without having to use the extra degree of freedom offered by the matrices

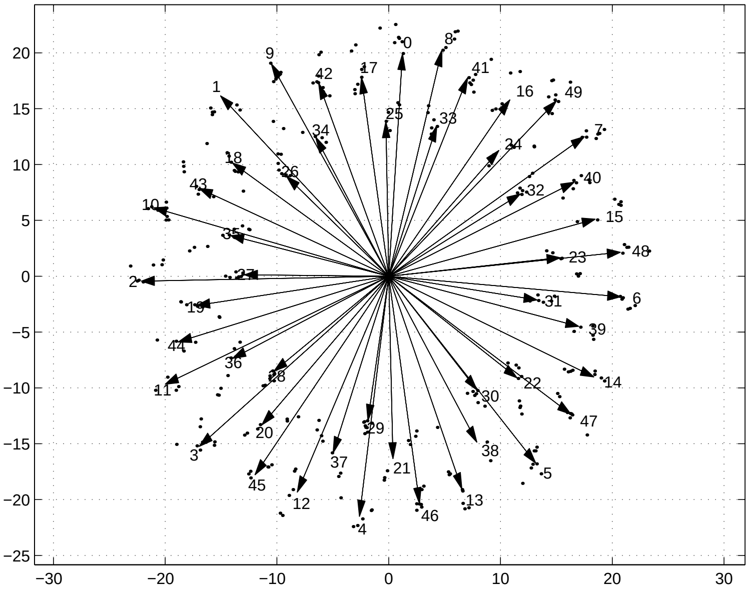

Figure 6: Vectors obtained after learning the modular number problem with numbers $[0 \ldots 49]$, operations $\{+1, -1, +2, -2, +3, -3, +4, -4, +0\}_{50}$ in two dimensions. The dots are the result of the multiplication $R^c \cdot \mathbf{a}^c$ for each triplet, c. This solution was obtained optimizing the goodness function (eq.3) using scaled conjugate gradient for 2050 iterations. 350 triplets randomly chosen out of 450 were used for training.

multiplicative factor - which is kept the same for every matrix (all the vectors can then also be kept of the same length). But as the problem becomes more complicated, the system will typically make use of this extra degree of freedom. For example if we try to solve the modular number problem with numbers $[0 \ldots 49]$ and operations $\{+1, -1, +2, -2, +3, -3, +4, -4, +0\}_{50}$ in two dimensions, we shall usually find a solution similar to the one in fig. 6, which was obtained training the system using 350 triplets randomly chosen out of the 450 constituting the data set. The optimization algorithm met the convergence criteria after 2050 iterations.

Let us now consider a *non-modular* version of the number problem with numbers $[1 \ldots 50]$ and operations $\{+1, -1, +2, -2, +3, -3, +4, -4, +0\}$. When the result of the operation is outside $[1 \ldots 50]$ the corresponding triplet is simply omitted from the data set. In two dimensions LRE was able to find the correct solution for all the 430 valid triplets of the problem, after training on 330 randomly chosen triplets for few hundred iterations. Figure 7 shows a typical vector configuration after learning. For the non-modular number problem, LRE increases the separation between the numbers by using different lengths for the concept vectors so that the numbers lie on a spiral. In the figure we also indicated with a cross the result of multiplying $R \cdot \mathbf{a}$ when the result of the operation is outside $[1 \ldots 50]$. Notice how the crosses are clustered, on the "ideal" continuation of the spiral - the answer to $49 + 3$ is located at almost exactly the same point as the answers to $48 + 4$, $50 + 2$, and so on. The system anticipates where the vectors representing numbers outside the given interval ought to be placed, if it had some information about them. We shall discuss this in the next section.

Now consider the non-modular numbers problem with numbers $[1 \ldots 50]$ and operations $\{+1, -1, +2, -2, +3, -3, +0, \div 3, \div 2, \times 1, \times 2, \times 3\}$. When we tried to solve it in 2 dimensions LRE

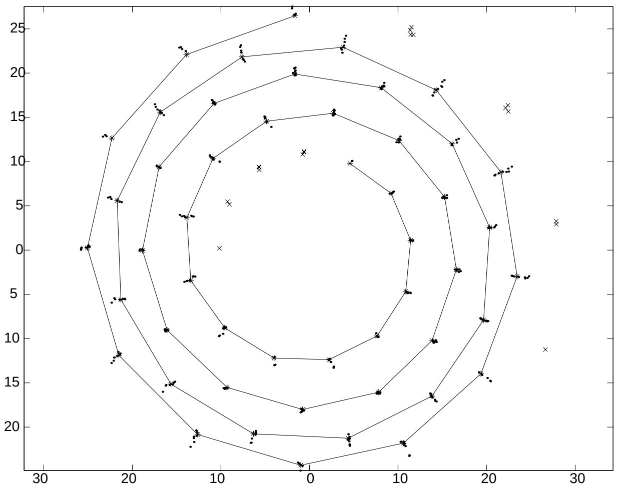

Figure 7: Vectors obtained after learning the non-modular number problem with numbers $[1 \dots 50]$, operations $\{+1, -1, +2, -2, +3, -3, +4, -4, +0\}$ in two dimensions. Vector endpoints are marked with stars and a solid line connects the ones representing consecutive numbers. Smaller numbers are in the center of the spiral. The dots are the result of the multiplication $R^c \cdot \mathbf{a}^c$ for each triplet, c. The crosses are the result of the multiplication $R \cdot \mathbf{a}$ when the result of the operation is outside $[1 \dots 50]$. This solution was obtained optimizing the goodness function (eq.3) using scaled conjugate gradient for 1485 iterations. 330 triplets randomly chosen out of 430 were used for training

could not find a solution that satisfied all the triplets. Using gradient ascent to optimize the modified goodness function while annealing the temperature, LRE found a solution that gave the correct answer for all the addition and subtraction operations but the matrices representing multiplications and divisions mapped all vectors to the origin. In 3 dimensions, however, LRE is able to find a perfect solution. For numbers in $[1 \dots 30]$ the solution found optimizing eq.3 using scaled conjugate gradient, is shown in Figure 8. The optimization algorithm met the convergence criteria after 726 iterations. [1].

Generalization results

LRE is able to generalize well. In 2 dimensions, with numbers $[0 \dots 9]$, and operations $\{+1, -1, +2, -2, +3, -3, +4, -4, +0\}_{10}$ we were able to train the system with just 70 of the 90 triplets in the training set, and yet achieve perfect results during testing on all the 90 cases. On the same problem, but using numbers $[0 \dots 49]$ and training with 350 triplets randomly chosen out of the 450, we usually got very few errors during testing, and occasionally no errors. The solution shown in fig.6, gave correct answers in 446 cases. The solution to the non-modular number problem, with numbers $[1 \dots 50]$ and operations $\{+1, -1, +2, -2, +3, -3, +4, -4, +0\}$ shown in fig.7, was trained on 330 out of the total 430 triplets, and yet is able to achieve perfect results during testing. It is worth pointing out that in order to do these generalizations the system had to discover structure implicit in the data.

[1] A similar result is obtained with numbers in $[1 \dots 50]$, but the figure is more cluttered

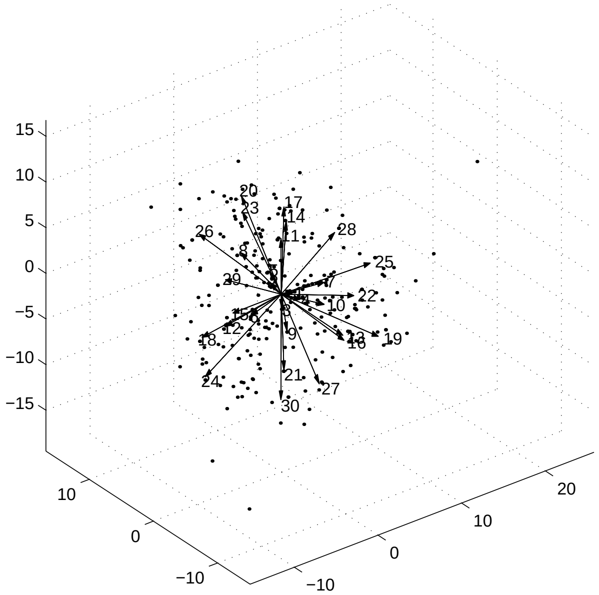

Figure 8: Vectors obtained after learning the non-modular number problem with numbers $[1 \dots 30]$, operations $\{+1, -1, +2, -2, +3, -3, +0, \div 3, \div 2, \times 1, \times 2, \times 3\}$ in three dimensions. The dots are the result of the multiplication $R^c \cdot \mathbf{a}^c$ for each triplet, c. This solution was obtained optimizing the goodness function (eq.3) using scaled conjugate gradient for 726 iterations.

4.2 Results on the Family Tree Problem

Our first attempt was to use LRE on a modified version of the family tree problem, which used the same family trees as fig.2, but only 6 sexless relations instead of the original 12 of Hinton (1986). These relations were: *spouse, child, parent, sibling, nipote, zii* (the last 2 being the Italian words for “either nephew or niece” and “either uncle or aunt”). As before there were 112 triplets.

Using 2 dimensions LRE was not able to complete all the 112 triplets, while it obtained a perfect solution using 3 dimensions. However, the 2-dimensional solution is quite interesting, so let us analyzing that. Fig.9a,b show the weight diagrams of matrices and vectors found after training on all the data, and fig.9c is a drawing of the concept vectors in 2-space, with vectors representing people in the same family tree being connected to each other.

We can see that nationality is coded using the sign of the second component of each vector, negative for English people, and positive for Italian people. The first component of each vector codes the generation to which that person belongs (a three valued feature): for the English people the first generation has negative values, the third generation has large positive values, while the second generation has intermediate positive values. The representations of the two families are linearly separable, and the two families are exactly symmetric with respect to the origin. An interesting fact is that some of the people were coded by identical vectors (for the English people, *Christopher* and *Penelope*, *Andrew* and *Christine*, *Colin* and *Charlotte*). This is clever if you notice that these people have the same children, same nephews and nieces and same uncles and aunts. Clearly this has as side effect that each of them is spouse or sibling of the correct person, but also of himself. This fact, together with the fact that the other people of each family are very close to each other, causes 14 errors when the 112 triplets were tested.

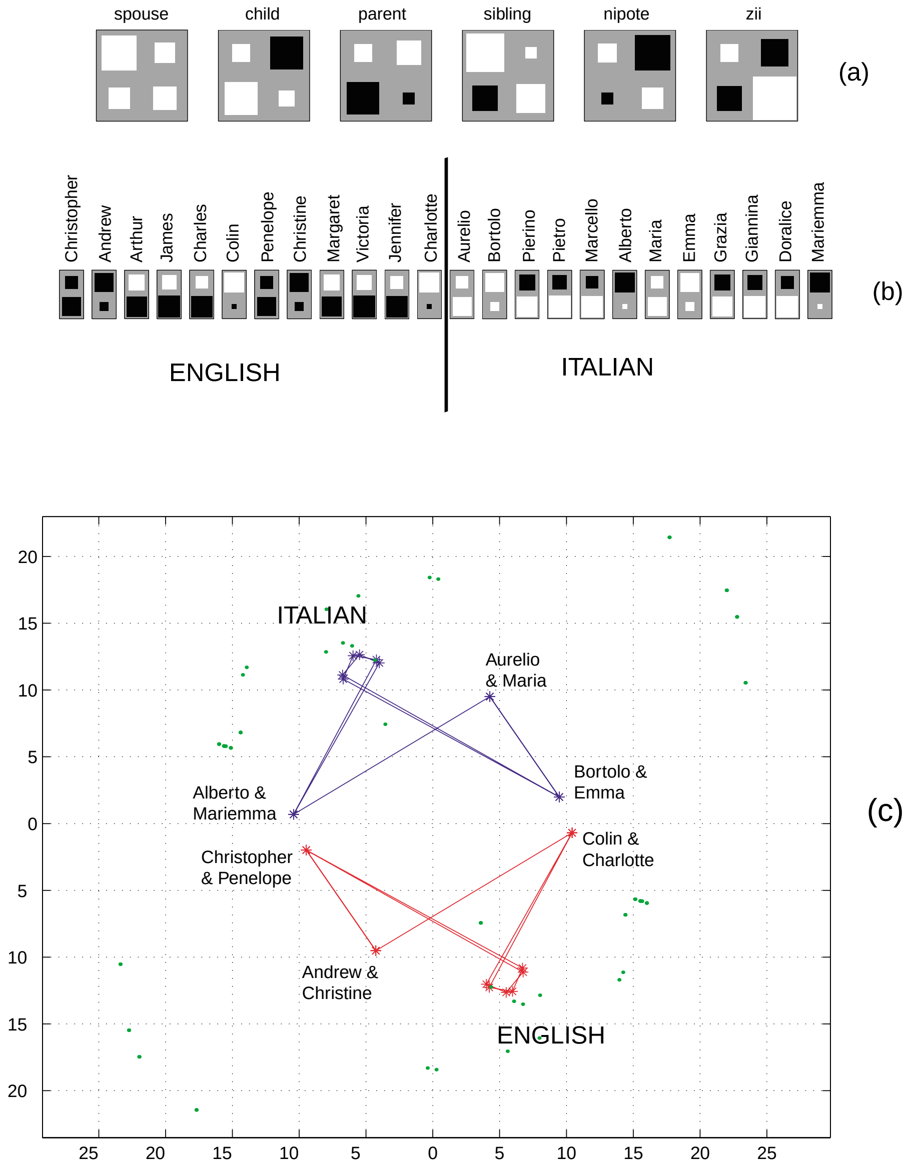

Figure 9: (a) Diagrams of the matrices and (b) the vectors obtained for the modified family tree problem. (c) Layout of the vectors in 2D space. Vectors are represented by *, the ones in the same family tree are connected to each other. The dots are the result of the multiplication $R^c \cdot \mathbf{a}^c$ for each triplet c. The solution shown here was obtained using gradient ascent to optimize the modified goodness function while the temperature was annealed.

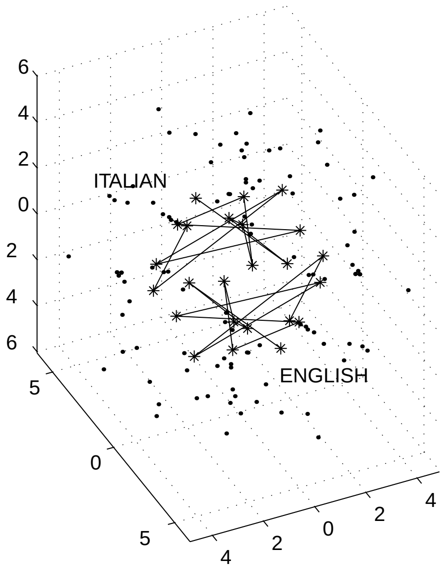

Figure 10: Layout of the vectors in 3D space obtained for the family tree problem. Vectors are represented by *, the ones in the same family tree are connected to each other. The dots are the result of the multiplication $R^c \cdot \mathbf{a}^c$ for each triplet, c. The solution shown here was obtained using gradient ascent to optimize the modified goodness function while the temperature was annealed.

We used LRE in 3 dimensions on the *family tree problem*. When trained on all the data, LRE could correctly complete all 112 triplets and the resulting concept vectors are shown in figure 10. We can see that the Italian and English families are symmetric with respect to the origin and are linearly separable. When more than one answer was correct (as in the aunts of *Colin*) the two concept vectors corresponding to the two correct answers were always the two vectors closest to $R^c \cdot \mathbf{a}^c$. Table 2 reports the distances of the each vector from the result of multiplying concept *Colin* and relation *aunt*.

Generalization results

On the modified version of the family tree problem, in 3 dimensions the system generalized perfectly on 8 new cases, while it got 1 wrong when it was tested on 12 new cases. On the original family tree problem, in 3 dimensions LRE generalized perfectly when 12 triplets were held out during training. In particular, even when all the information on “who are the aunts of Colin” (i.e both triplets *(Colin, aunt, Jennifer)* and *(Colin, aunt, Margaret)*) was held out during training, the system was still able to answer correctly. Notice how, in order to do this, the system had first to use the implicit information in the other triplets to figure out both the meaning of the relation *aunt* and the relative position of *Colin, Margaret* and *Jennifer* in the tree, and then use this information to make the correct inference.

The generalization achieved by LRE is much better than the neural networks of Hinton (1986) and O’Reilly (1996) which typically made one or two errors even when only 4 cases were held out during training.

PERSON	DISTANCE
Jennifer	1.6064
Margaret	1.6549
Emma	3.0603
Charlotte	3.0865
Penelope	3.2950
Bortolo	3.7471
Christopher	3.9597
Giannina	4.1198
Marcello	4.4083
Alberto	5.1281
Arthur	5.2167
Colin	5.2673
Pierino	5.4619
James	5.4858
Charles	5.5943
Pietro	5.6432
Andrew	6.3581
Aurelio	6.3880
Mariemma	6.5021
Victoria	6.6853
Christine	6.6973
Maria	6.7626
Grazia	7.1801
Doralice	7.4230

Table 2: Distance of each concept vector from the result of multiplying concept *Colin* and relation *aunt* for the solution to the family tree problem shown in figure 10.

4.3 Results of Principal Components Analysis on number and family tree problems

We have used Principal Components Analysis (PCA) to complete the triplets of the modular and non-modular number problem and of the family tree problem, in order to see how it compares with LRE. For the number problems, we used numbers $[0 \dots 9]$ and operations $\{+1, -1, +2, -2, +3, -3, +4, -4, +0$ For each concept and relation we used a one-out-of-n codification, thus each triplet for the number problems was a point in $10 + 9 + 10$ dimensions, while a triplet for the family tree problem was a point in $24+12+24$ dimensional space. Having chosen a certain number of principal components, we tried to complete the triplets. For each triplet c, given the first 2 terms ($\mathbf{a}^c$, R^c) we choose as completion the concept $\mathbf{b}$ such that the point $(\mathbf{a}^c, R^c, \mathbf{b})$ was the closest to the PCA plane for every possible choice of $\mathbf{b}$. [2] Figure 11 shows the number of triplets which were correctly completed vs. the number of principal components which were used for the modular, non-modular and family tree problem respectively when 0, 10 and 20 triplets were omitted from the training set. Notice how PCA had an excellent performance on the non-modular numbers problem but not on the modular version. In general the performance of this method is much worse than LRE.

[2] We also tried to reconstruct the third term of a triplet by setting all its components to zero, then projecting the resulting triplet into principal components space, and then back into the original space, but the results that we obtained were not as good.

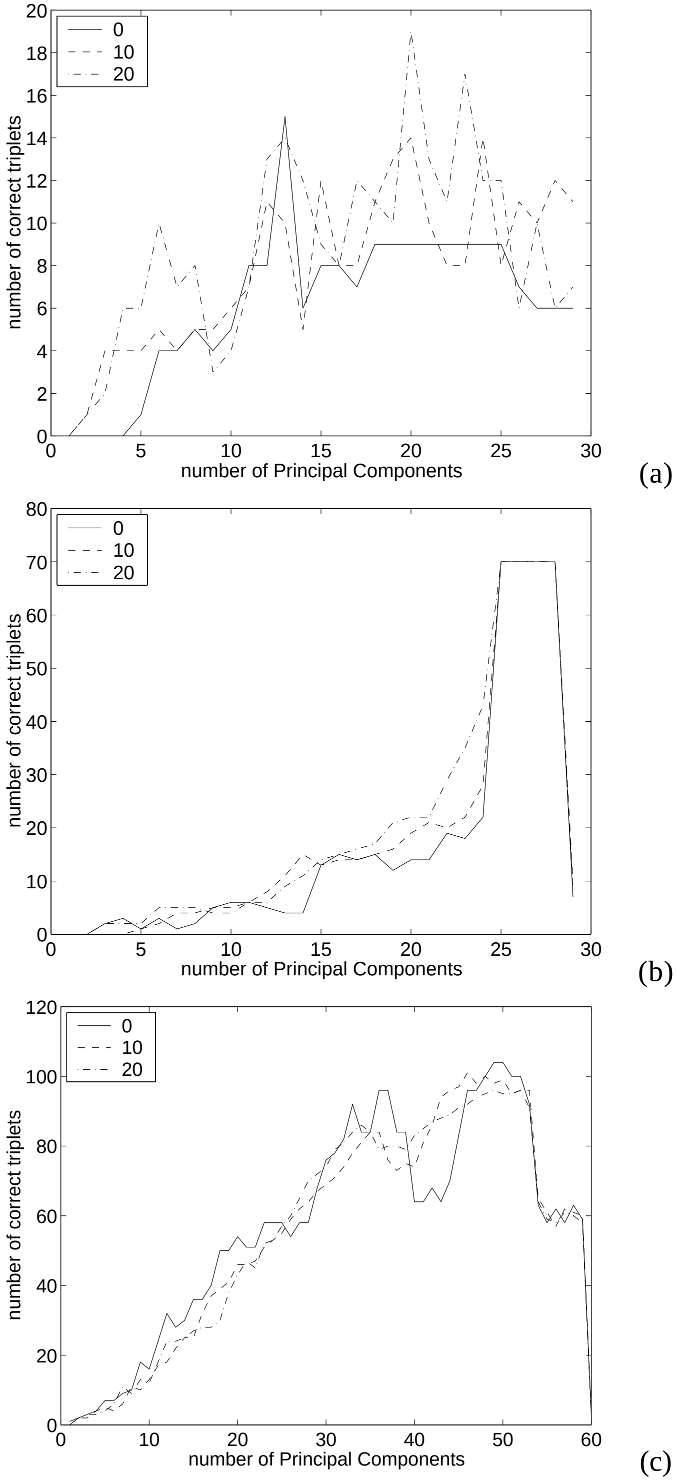

Figure 11: Number of triplets which were correctly completed vs. the number of principal components used. The solid lines were obtained when all triplets were used for training; the dashed lines were obtained omitting 10 triplets from the training set; the dash-dotted lines were obtained omitting 20 triplets from the training set. (a) modular number problem with numbers $[0 \ldots 9]$, operations $\{+1, -1, +2, -2, +3, -3, +4, -4, +0\}_{10}$ (b) number problem with numbers $[0 \ldots 9]$, operations $\{+1, -1, +2, -2, +3, -3, +4, -4, +0\}$ (c) family tree problem.

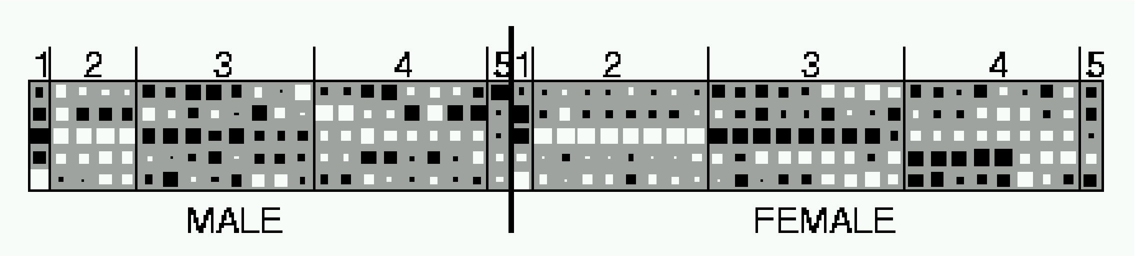

Figure 12: Diagram of the components of the vectors, obtained after learning the family tree problem with real data for 2000 iterations using scaled conjugate gradient. All 644 triplets were used during training. When testing, the system correctly completed 635 triplets. Each column represents a person. The first 22 vectors represent the males and the others the females in the family tree. The numbers denote the generation of the tree they belong to.

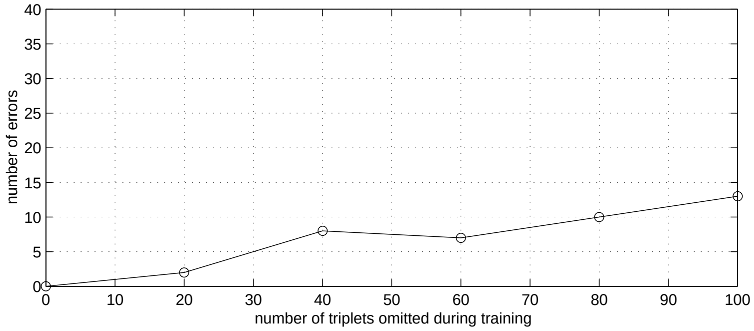

Figure 13: Plot of the errors made by the system when tested on the whole set of 644 triplets vs. the number of triplets which were omitted during training. Omitted triplets were chosen randomly.

4.4 Results on the Family Tree Problem with real data

We have used LRE to solve a much bigger family tree task. The tree is a branch of the real family tree of one of the authors containing 49 people. Using the 12 relations seen earlier it generates a data set of 644 triplets. LRE was able to learn the tree with as few as 6 dimensions using scaled conjugate gradient.

When we used a small number of dimensions, it was sometimes possible to recognize some of the semantic features in the learned representation. Figure 12 shows the diagram of the components of the vectors, each column represents a person. The first 22 vectors represent the males and the others the females in the family tree. The numbers denote the generation of the tree they belong to. We can see that the sign and magnitude of the third component of each vector codes the generation that person belongs to: the first, third and fifth generation have negative sign, and decreasing magnitudes; the second and forth have positive sign and decreasing magnitude.

The generalization performance was very good. Figure 13 is the plot of the number of errors made by the system when tested on the whole data set after being trained on a subset of it using 10 dimensions.

Triplets were extracted randomly from the training set and the system was run for 5000 iterations, or until the convergence criteria were met. The results shown are the *median* of the number of errors over 3 different runs, since the system very occasionally failed to converge. We can see that the performance degrades slowly as an increasing number of triplets is omitted from the training data.

5 Generalization to new concepts and relations

As pointed out earlier, the system anticipates where the vectors for the numbers outside the learned range ought to be placed, if it had some information about them. In this section we investigated how a solution we obtain after learning on a set of data, can be modified to include information about new concepts and relations.

Let us start by training a system using LRE on the non-modular number problem, with numbers $[1 \dots 15]$ and operations $\{+1, -1, +2, -2, +3, -3, +4, -4, +0\}$ but omitting all the information about a certain number from the training set, i.e. omitting all the triplets which contain that number either as the first or third term. Figure 14a shows the vectors which are found after 242 iterations of scaled conjugate gradient when learning in two dimensions after having eliminated all the information about number 10 from the training data. Notice how a point is clearly "missing" from the spiral in the place where number 10 should go. If we now add some information about the missing number to the training data and continue the training, we see that in very few iterations the vector representing that number is placed exactly where it is supposed to go. It is interesting that a single triplet containing information about a new number is enough for the system to position it correctly. This happens both when we allow all the vectors and matrices to continue learning after we have added the extra data point, or when we keep all vectors and matrices fixed, and we allow only the new vector to learn. Figure 14b,c shows the solution obtained starting from the solution shown in fig.14a and training using only triplet $(10, +1, 11)$. After having learned the position of the new number from that one triplet, the system is then able to generalize, and answers correctly to all the triplets in the complete data set.

We also tried to learn a new relationship. Clearly this is more difficult since a new matrix to be learned has more degrees of freedom. In general we saw that in the number problem several triplets were necessary in order to learn a new matrix that would be able to correctly complete all the triplets in the data.When we trained a system using LRE on the non-modular number problem in 2 dimensions, with numbers $[1 \dots 15]$ and operations $\{+1, -1, +2, +3, -3, +4, -4, +0\}$, it was usually possible to learn the matrix for the -2 operation using four triplets when all the vectors and matrices were allowed to learn. Six triplets were usually necessary if only the new matrix was allowed to learn, while everything else was kept fixed.

Finally we tried the same experiment on the family tree with real data. Here the situation is not as straightforward as for the numbers, since not all triplets contain the same amount of information about a concept or a relation. The triplet "Pietro has wife Giannina" makes it possible to locate *Pietro* exactly on the family tree. But the triplet "Pietro has nephew Giulio" leaves a lot of uncertainty about *Pietro*, who could be a sibling of one of *Giulio*'s parents or someone married to one of the sibling of *Giulio*'s parents. Similarly, "Giannina has son Alberto" has more information about relation *son* than "Giannina has aunt Virginia" has about relation *aunt*. For these reason the performance of the system after learning a new person vector or a new relation matrix depended on which triplets were added for training.

The father of one author is mentioned in 14 triplets in the data set. If such information was omitted, LRE was able to complete all the remaining triplets correctly after having been trained for 1501 iterations. Then it was sufficient to add a single triplet stating who that person is married to, in order to locate him correctly in the family tree. On the other hand it made 5 errors if it

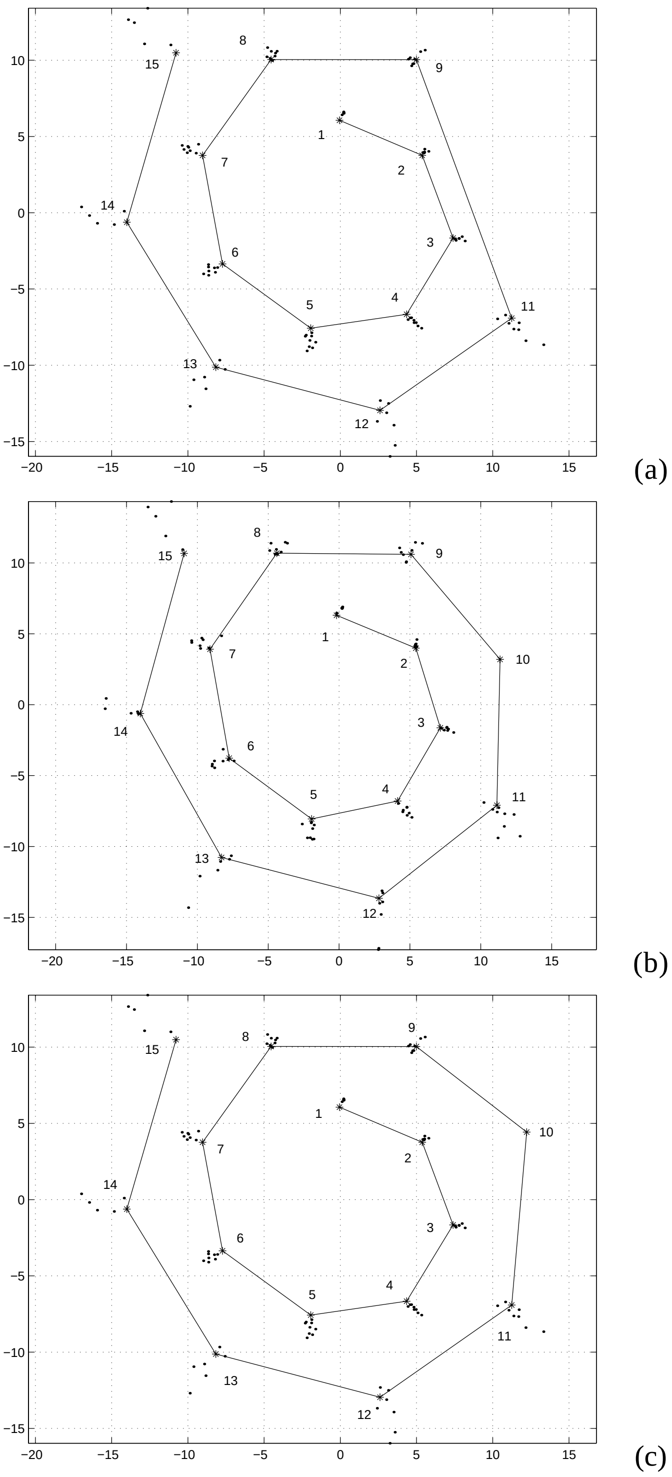

Figure 14: Vectors obtained after learning the number problem with numbers $[1 \dots 15]$, operations $\{+1, -1, +2, -2, +3, -3, +4, -4, +0\}$. Vector endpoints are marked with stars and a solid line connects the ones representing consecutive numbers. The dots are the result of the multiplication $R^c \cdot \mathbf{a}^c$ for each triplet, c. (a) The information about number 10 was omitted. The optimization algorithm met the convergence criteria after 242 iterations using scaled conjugate gradient. Triplet $(10, +1, 11)$ was then added to the data set and the system was trained starting from this configuration. (b) all matrices and vectors were allowed to learn, the algorithm met the convergence criteria after 239 iterations (c) only the vector representing number 10 was allowed to learn, while everything else was kept fixed, the algorithm met the convergence criteria after 2 iterations

was trained adding only a triplet that specified one of his nephews. When we tried to learn a new matrix, the high dimensionality required by the problem means that a very high number of triplets was necessary for learning.

6 Further developments

A minor modification, which we have not tried yet, should allow the system to make use of negative data of the form "Christopher is not the father of Colin". This could be handled by minimizing G instead of maximizing it, while using Christopher as the "correct answer".

One limitation of the version of LRE presented here is that it always picks some answer to any question even if the correct answer is not one of the concepts presented during training. This limitation can be overcome by using a threshold distance so that the system answers "don't know" if the vector it generates is further than the threshold distance from all known concepts. Preliminary experiments with the non-modular number problems have been very successful. Instead of ignoring triplets in which the operation produces an answer outside the set of known numbers, we include these triplets, but make the correct answer "don't know". If, for example, the largest known number is 10, LRE must learn to make the answer to $9+3$ be further than the threshold distance from all the known numbers. It succeeds in doing this and it locates the answer to $9+3$ at almost exactly the same point as the answers to $10+2$ and $8+4$. In a sense, it has constructed a new concept. See figure 15.

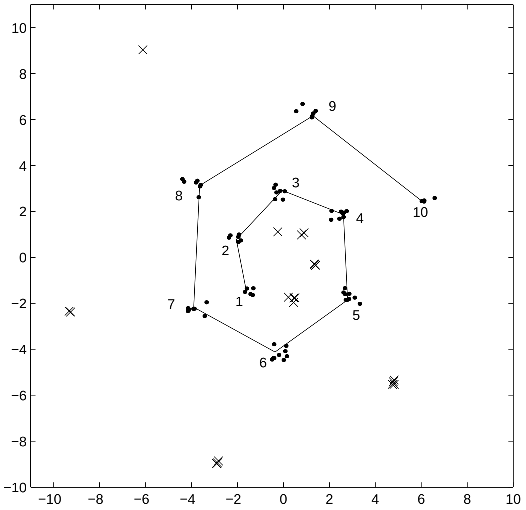

Figure 15: Vectors obtained after learning the number problem with numbers $[1 \ldots 10]$, operations $\{+1, -1, +2, -2, +3, -3, +4, -4, +0\}$. Vector endpoints are marked with stars and a solid line connects the ones representing consecutive numbers. The dots are the result of the multiplication $R^c \cdot \mathbf{a}^c$ for each triplet c such that the answer is among the known concepts. The crosses are the result of the multiplication $R^k \cdot \mathbf{a}^k$ for each triplet k such that the correct answer is "don't know". In all cases the system answered correctly to all the questions in the data set. All the triplets were used for training.

Another limitation is that a separate matrix is needed for each relation. This requires a lot of parameters because the number of parameters in each matrix is the square of the dimensionality of the concept vector space. When there are many different relations it may be advantageous to model their matrices as linear combinations of a smaller set of learned basis matrices. This has some similarity to the work of Tenenbaum and Freeman (1996).

In this paper, we have assumed that the concepts and relations are presented as arbitrary symbols so there is no inherent constraint on the mapping from concepts to the vectors that represent them. LRE can also be applied when the "concepts" already have a rich and informative representation. Consider the task of mapping pre-segmented intensity images into the pose parameters of the object they contain. This mapping is non-linear because the average of two intensity images is not an image of an object at the average of the positions, orientations and scales of the objects in the two images. Suppose we have a discrete sequence of images $I(1) \dots I(t) \dots I(T)$ of a stationary object taken with a moving camera and we know the camera motion $M(t, t+1)$ between each successive image pair.

In an appropriately parameterized space of pose parameters, the camera motion can be represented as a transformation matrix, $R(t, t+1)$, that converts one pose vector into the next:

$$R(t, t+1)v(t) \; = \; v(t+1) \tag{9}$$

The central assumption of LRE is therefore exactly satisfied by the representation we wish to learn. So it should be possible to learn a mapping from intensity images to pose vectors and from sensory representations of camera motions to transformation matrices by backpropagating the derivatives obtained from Eq. 3 through a non-linear function approximator such as a multilayer neural network. Preliminary simulations by Sam Roweis (personal communication) show that it is feasible to learn the mapping from preprocessed intensity images to pose vectors if the mapping from camera motions to the appropriate transformation matrices is already given.

7 Discussion

Linear Relational Embedding is a new method for discovering distributed representations of concepts and relations from data consisting of binary relations between concepts. On the task on which we tried it, it was able to learn sensible representations of the data, and this allowed it to generalize well.

In the family tree task with real data, the great majority of the generalization errors were of a specific form. The system appears to believe that "brother of" means "son of parents of". It fails to model the extra restriction that people cannot be their own brother. This failure nicely illustrates the problems that arise when there is no explicit mechanism for variable binding.

A key technical trick that was required to get LRE to work was the use of the discriminative goodness function in Eq. 3. If we simply minimize the squared distance between $R^c \cdot \mathbf{a}^c$ and $\mathbf{b}^c$ all the concept vectors rapidly shrink to 0. It may be possible to apply this same technical trick to other ways of implementing relational structures in neural networks. Pollack's RAAM (Pollack, 1990) and Sperduti's LRAAM's (Sperduti, 1994) minimize the squared distance between the input and the output of an autoencoder and they also learn the representations that are used as input. They avoid collapsing all vectors to 0 by insisting that some of the symbols (terminals and labels) are represented by fixed patterns that cannot be modified by learning. It should be possible to dispense with this restriction if the squared error objective function is replaced by a discriminative function which forces the output vector of the autoencoder to be closer to the input than to the alternative input vectors.

References

Bridle, J. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In Soulié, F. F. and Hérault, J., editors, *Neurocomputing: Algorithms, Architectures and Applications*, pages 227–236. Springer-Verlag, New York.

Cleeremans, A., Servan-Schreiber, D., and McClelland, J. (1989). Finite state automata and simple recurrent neural networks. *Neural Computation*, 1(3):372–381.

Deerwester, S., Dumais, S. T., Furnas, G., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407.

Elman, J. (1990). Finding structure in time. *Cognitive Science*, 14:179–211.

Giles, C., Miller, C., Chen, D., Chen, H., Sun, G., and Lee, Y. (1992). Learning and extracting finite state automata with second order recurrent neural networks. *Neural Computation*, 4(3)::380.

Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12. Erlbaum, NJ.

Kruskal, J. B. (1964). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29, 1:1–27.

Landauer, T. K. and Dumais, S. T. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychological Review*, 104, 2:211–240.

Landauer, T. K., Laham, D., and Foltz, P. (1998). Learning human-like knowledge by singular value decomposition: A progress report. In Jordan, M. I., Kearns, M. J., and sara A. Solla, editors, *Advances in Neural Processing Information Systems 10*, pages 45–51. The MIT Press, Cambridge Massachusetts.

Møller, M. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525–533.

O'Reilly, R. C. (1996). *The LEABRA model of neural interactions and learning in the neocortex*. PhD thesis, Department of Psychology, Carnegie Mellon University.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, 46:77–105.

Rumelhart, D. E., Hinton, G. E., and Williams, R. (1986). Learning internal representation by error propagation. In Rumelhart, D. E., McClelland, J. L., and the PDP research Group, editors, *Parallel Distributed Processing*, volume 1, pages 283–317. The MIT Press.

Sperduti, A. (1994). Labeling RAAM. *Connection Sci.*, 6:429–459.

Tenenbaum, J. B. and Freeman, W. T. (1996). Separating style and content. In Mozer, M. C., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Processing Information Systems 9*, pages 662–668. The MIT Press, Cambridge Massachusetts.

Young, F. W. and Hamer, R. M. (1987). *Multidimensional Scaling: History, Theory and Applications*. Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers,.