Learning Distributed Representations of Concepts from Relational Data

Alberto Paccanaro A.PACCANARO@QMUL.AC.UK
Dept. of Medical Microbiology, Queen Mary University of London, UK
http://www.gatsby.ucl.ac.uk/~alberto

Abstract

In this paper we discuss methods for generalizing over relational data. Our approach is to learn distributed representations for the concepts that coincide with their semantic features and then to use these representations to make inferences. We present Linear Relational Embedding (LRE), a method that learns a mapping from the concepts into a feature-space by imposing the constraint that relations in this feature-space are modeled by linear operations. We then show that this linearity constrains the type of relations that LRE can represent. Finally, we introduce Non-Linear Relational Embedding (NLRE), and show that it can represent any relation. Results of NLRE on a small but difficult problem show that generalization is much better than the one obtained using backpropagation.

1. Introduction

Let us imagine a situation in which we have a set of concepts and a set of relations among these concepts, and our data consists of few instances of these relations that hold among the concepts. We want to be able to infer other instances of these relations. For example, if the concepts are the people in a certain family, the relations are kinship relations, and we are given the facts "Alberto has-father Pietro" and "Pietro has-brother Giovanni", we would like to be able to infer "Alberto has-uncle Giovanni". Our ultimate aim is to be able to take a large set of facts about a domain and to be able to infer other "common-sense" facts without having any prior knowledge about the domain.

Recently we have introduced a method called Linear Relational Embedding (LRE) (Paccanaro & Hinton, 2001) for solving this kind of problems. The approach is grounded in the observation that real world relations do not associate concepts in a random fashion, but rather concepts relate to each other depending on "what" they are, i.e. according to their semantic features. For example, the relation `''has-mother''` relates a person to another which must have the features of being one generation *older* than that person and *female*. This gives us a starting point to try to solve our problem: if concepts relate to each other according to their features, then if we could obtain a representation of each concept in terms of its features, together with the correct rules for how such features interact, it should possible to infer how concepts relate. Our approach to solving this problem is therefore to assume the existence of structure in the world and to learn a set of features for each concept together with the rules on how such features interact. We then use a representation of the concepts in terms of these learned features to infer new instances of the relations.

These ideas can be traced back to a paper by Hinton (1981), one of the first attempts to represent relational data in a connectionist network using distributed representations. Hinton chose the distributed representation of each concept to be the set of the semantic features of that concept, and showed that a system can exploit such distributed representations to provide automatic generalization. However he used hand-coded representations and did not provide a satisfactory method for learning them from the data. Few methods for devising such pattern for relational data have been proposed, but are either limited in the type of relations that they can represent, or cannot generalize very well (see for example Hinton, 1986; Miikkulainen & Dyer, 1989). In the next sections we shall present methods that learn distributed representations for concepts and relations that coincide with their semantic features and we shall see that they provide excellent generalization performance.

2. Linear Relational Embedding

Let us begin considering the case in which all the relations are binary. Our data then consists of triplets $(concept_1, relation, concept_2)$, and the problem we are trying to solve is to infer missing triplets when we are given only few of them. Inferring a triplet is equivalent to complete it, that is to provide one of its elements given the other two. In the following we shall train systems that provide the third element, given the first two.

LRE learns a distributed representation for the concepts and the relations by embedding the concepts in a space where the relations between them are linear transformations of their distributed representations. In other words, it represents each concept as a learned vector in a Euclidean space and each relationship between the two concepts as a learned matrix that maps the first concept into an approximation to the second concept.

To get a flavour for this approach, we start with a very simple task in which the data consists of concepts which are integers in the set $\mathfrak{D} = [\mathtt{0} \ldots \mathtt{9}]$ and relations are the following modulo 10 operations among integers $\mathfrak{L} = \{\mathtt{+1}, \mathtt{-1}, \mathtt{+2}, \mathtt{-2}, \mathtt{+3}, \mathtt{-3}, \mathtt{+4}, \mathtt{-4}, \mathtt{+0}\}$. Our data will consist of triplets of the kind (num_1, op, num_2) where $num_1, num_2 \in \mathfrak{D}$, $op \in \mathfrak{L}$, and num_2 is the result of applying operation op to number num_1; for example, for $\{(\mathtt{1}, \mathtt{+1}, \mathtt{2}), (\mathtt{4}, \mathtt{+3}, \mathtt{7}), (\mathtt{9}, \mathtt{+3}, \mathtt{2}), \cdots\}$. There are 90 triplets for this problem. We would like to have a system that, when trained on some of these triplets, say half of them, is able to infer the other ones. As we anticipated earlier, LRE represents concepts as 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. If we think for a moment, we realize that solutions of this form do exist for this problem, since it is easy to hand-code one for $n = 2$. In fact the numbers can be represented by vectors having unit length and disposed as in figure 1a, while the relations can be 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 $\mathtt{+3}$ to number $\mathtt{4}$, is then obtained by multiplying the corresponding matrix and vector, which amounts to rotating the vector located at 144 degrees by 108 degrees, thus obtaining a vector at 252 degrees, which represents number $\mathtt{7}$. LRE is able to find a solution that is equivalent to this hand-coded one, which is shown in figure 1b and in the second row of table 1. LRE learns this solution by training on only half of the triplets randomly chosen from the complete data set and once it has learned this way of representing the concepts and relationships it can complete all the triplets correctly.

There are 2 different phases of learning in LRE. First we learn a distributed representation for the concepts and the relations. In order to use these distributed representations to infer new triplets, we then need to embed these representations into a probabilistic model. In the second phase of the learning we learn the parameters of this probabilistic model.

2.1. Learning the distributed representations

Let us assume that our data consists of C such triplets containing N distinct concepts and M binary relations. We shall call this set of triplets $\mathcal{C}$; $\mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_N\}$ will denote the set of n-dimensional vectors corresponding to the N concepts, and $\mathcal{R} = \{R_1, \ldots, R_M\}$ the set of $(n \times n)$ matrices corresponding to the M relations. Often we shall need to indicate the vectors and the matrix which correspond to the concepts and the relation in a certain triplet c. In this case we shall denote the vector corresponding to the first concept with $\mathbf{a}$, the vector corresponding to the second concept with $\mathbf{b}$ and the matrix corresponding to the relation with R. We shall therefore write the triplet c as $(\mathbf{a}^c, R^c, \mathbf{b}^c)$ where $\mathbf{a}^c, \mathbf{b}^c \in \mathcal{V}$ 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 $\mathbf{b}^c$. If for every triplet $(\mathbf{a}^c, R^c, \mathbf{b}^c)$ we think of $R^c \cdot \mathbf{a}^c$ as a noisy version of one of the concept vectors, then one way to learn an embedding is to maximize the probability that it is a noisy version of the correct completion, $\mathbf{b}^c$. We imagine that a concept has an average location in the space, but that each "observation" of the concept is a noisy realization of this average location. Assuming that the noise is Gaussian with a variance of 1/2 on each dimension and that all concepts are equally probable, the posterior probability that $R^c \cdot \mathbf{a}^c$ matches concept $\mathbf{b}^c$ given that it must match one of the known concepts is:

$$P(\mathbf{b}^c | R^c \cdot \mathbf{a}^c) = \frac{e^{-\|R^c \cdot \mathbf{a}^c - \mathbf{b}^c\|^2}}{\sum_{\mathbf{v}_i \in \mathcal{V}} e^{-\|R^c \cdot \mathbf{a}^c - \mathbf{v}_i\|^2}} \tag{1}$$

The probability of getting the right completion for each triplet in the data set is:

$$\prod_{c=1}^{C} \frac{e^{-||R^c \cdot \mathbf{a}^c - \mathbf{b}^c||^2}}{\sum_{\mathbf{v}_i \in \mathcal{V}} e^{-||R^c \cdot \mathbf{a}^c - \mathbf{v}_i||^2}} \tag{2}$$

A discriminative goodness function D, that corresponds to the log probability of getting the right com-

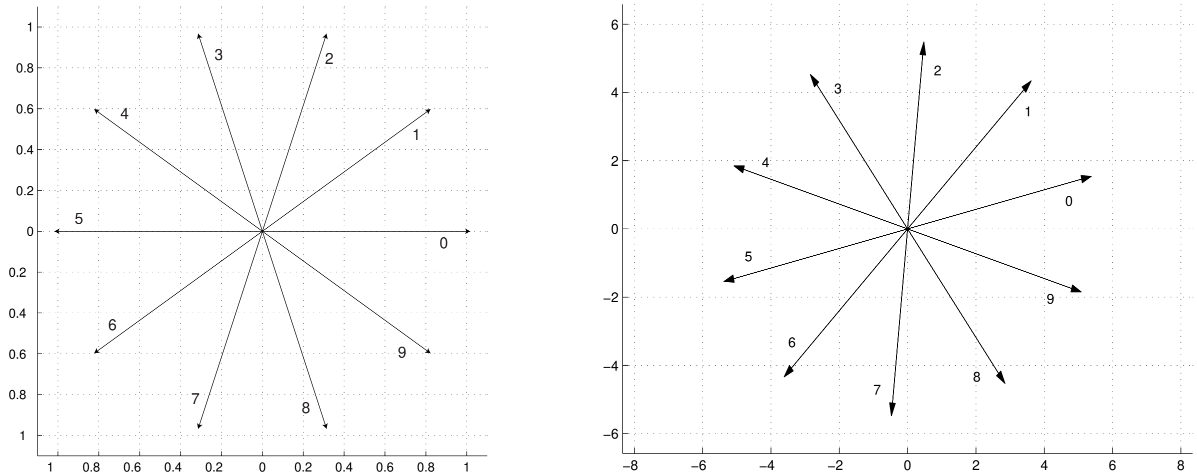

Figure 1. Left: vectors of the hand-coded solution for the Number Problem with $n = 2$. Right: vectors of a solution found by LRE. Only 45 triplets, chosen at random from the 90 possible triplets were used for training. During testing, the system was able to correctly complete all the triplets for this problem.

Table 1. Angles, expressed in degrees, of the rotation matrices of the solutions to the Number Problem with $n = 2$, corresponding to the vectors in figure 1. The small errors of the LRE solution are due to the fact that only 45 triplets were used during training.

OPERATION	-4	-3	-2	-1	+0	+1	+2	+3	+4
Hand-coded Solution	-144	-108	-72	-36	0	36	72	108	144
LRE Solution	-143.95	-107.97	-72.02	36.04	0.00	36.04	72.02	107.97	143.95

pletion, summed over all training triplets is therefore:

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

If one-to-many relations are present in our data, then the goodness function D needs to be slightly modified and becomes:

$$D = \sum_{c=1}^{C} \frac{1}{k_c} \log \frac{e^{-\|R^c \cdot \mathbf{a}^c - \mathbf{b}^c\|^2}}{\sum\limits_{\mathbf{v}_i \in V} e^{-\|R^c \cdot \mathbf{a}^c - \mathbf{v}_i\|^2}} \tag{4}$$

where k_c is the number of triplets in $\mathcal{C}$ having the first two terms equal to the ones of c, but differing in the third term[1].

The distributed representations learned by maximizing D with respect to all the vector and matrix components provide good generalization (Paccanaro & Hinton, 2000). However, when we learn an embedding by maximizing D, we are not making use of exactly the information that we have in the triplets. For each triplet c, we are making the vector representing the correct completion $\mathbf{b}^c$ *more probable* than any other concept vector given $R^c \cdot \mathbf{a}^c$, while the triplet states that $R^c \cdot \mathbf{a}^c$ must be *equal* to $\mathbf{b}^c$. The numerator of D does exactly this, but we also have the denominator, which is necessary in order to stay away from the trivial $\mathbf{0}$ solution[2]. We noticed however that the denominator is critical at the beginning of the learning, but as the vectors and matrices differentiate we could gradually lift this burden, allowing $\sum_{c=1}^{C} \|R^c \cdot \mathbf{a}^c - \mathbf{b}^c\|^2$ to become the real goal of the learning. To do this we modify the discriminative function to include a parameter α, which

[1] We would like our system to assign equal probability to each of the correct completions. The discrete probability distribution that we want to approximate is therefore: $\mathsf{P}_{\mathbf{x}} = \frac{1}{d} \sum_{i=1}^{d} \delta(\mathbf{b}_i - \mathbf{x})$ where δ is the discrete delta function and $\mathbf{x}$ ranges over the vectors in $\mathcal{V}$. Our system implements the discrete probability distribution: $\mathsf{Q}_{\mathbf{x}} = \frac{1}{Z} \exp(-\|R \cdot \mathbf{a} - \mathbf{x}\|^2)$ where Z is the normalization factor. The $1/k_c$ factor in eq.3 ensures that we are minimizing the Kullback-Leibler divergence between P and Q.

[2] The obvious approach to find an embedding would be to minimize the sum of squared distances between $R^c \cdot \mathbf{a}^c$ and $\mathbf{b}^c$ over all the triplets, with respect to all the vector and matrix components. Unfortunately this minimization (almost) always causes all of the vectors and matrices to collapse to the trivial $\mathbf{0}$ solution.

is annealed from 1 to 0 during learning[3]:

$$G = \sum_{c=1}^{C} \frac{1}{k_c} \log \frac{e^{-\|R^c \cdot \mathbf{a}^c - \mathbf{b}^c\|^2}}{[\sum_{\mathbf{v}_i \in \mathcal{V}} e^{-\|R^c \cdot \mathbf{a}^c - \mathbf{v}_i\|^2}]^\alpha} \tag{5}$$

During learning this function G (for Goodness) is maximized with respect to all the vector and matrix components.

2.2. Learning the probabilistic model

Once we have obtained the distributed representations for the concepts and the relations by maximizing G, we need a principled way to use these representations to solve our generalization problem, that is to complete a given triplet. This is made difficult by the fact that for most datasets there exist triplets which cannot be completed using the available data, for which the correct completion should be "don't know"; the system needs a way to indicate when a triplet does not admit a completion.

Therefore, after having found distributed representations for the concepts by maximizing G, in order to use these representations to infer new triplets we embed them into a probabilistic model. The probabilistic model will have, for each relation, a mixture of N identical spherical Gaussians each centered on a concept vector and a Uniform distribution. The Uniform distribution will take care of the "don't know" answers and will be competing with all the other Gaussians, each representing a vector constituting a concept in our data set. For each relation the Gaussians have different variances and the Uniform distribution a different height. The parameters of this probabilistic model are, for each relation R, the variances of the spherical Gaussians σ_R and the relative density under the Uniform distribution, which we shall write as $\exp(-\frac{r_R^2}{2\sigma_R^2})$. These parameters are learned using a validation set, which will be the union of a set of complete-able triplets $\mathcal{P}$ and a set of uncomplete-able ones $\mathcal{N}$; that is $\mathcal{P} = \{\mathbf{a}^p, R^p, \mathbf{b}^p\}_{p=1}^P$ and $\mathcal{N} = \{\mathbf{a}^q, R^q, \perp\}_{q=1}^Q$ where $\perp$ indicates the fact that the result of applying relation R^q to $\mathbf{a}^q$ does not belong to $\mathcal{V}$.

We learn a value for σ_R and r_R, by maximizing the following discriminative goodness function F over the validation set[4]:

$$\begin{aligned} F &= \sum_{q=1}^{Q} \log \frac{U}{U + \sum_{\mathbf{v}_i \in \mathcal{V}} \exp(-\frac{\|R^q \cdot \mathbf{a}^q - \mathbf{v}_i\|^2}{2\sigma_R^2})} \\ &+ \sum_{p=1}^{P} \frac{1}{k_p} \cdot \log \frac{\exp(-\frac{\|R^p \cdot \mathbf{a}^p - \mathbf{b}^p\|^2}{2\sigma_R^2})}{U + \sum_{\mathbf{v}_i \in \mathcal{V}} \exp(-\frac{\|R^p \cdot \mathbf{a}^p - \mathbf{v}_i\|^2}{2\sigma_R^2})} \end{aligned} \tag{6}$$

where $U = \exp(-r_R^2/2\sigma_R^2)$ and, as before, k_p is the number of triplets in $\mathcal{P}$ having the first two elements equal to the ones of p, but differing in the third one. Thus for each triplet in $\mathcal{P}$ we maximize the probability that the vector $R^p \cdot \mathbf{a}^p$ is generated from the Gaussian centered on $\mathbf{b}^p$, while for each triplet in $\mathcal{N}$, we maximize the probability that the vector $R^q \cdot \mathbf{a}^q$ is generated from the Uniform distribution.

Having learned these parameters, in order to complete any triplet $(R, \mathbf{a}, ?)$ we compute the probability distribution over each of the Gaussians and the Uniform distribution given $R \cdot \mathbf{a}$. The system then chooses a concept vector or the "don't know" answer according to those probabilities, as the completion to the triplet.

2.3. Details of the learning procedure

We learn the distributed representations for the concepts and the relations by maximizing G with respect to all the vector and matrix components. These are initialized to small random values. One architectural decision to be made is the choice of the dimensionality of the concept vectors. We normally choose this value empirically, and we have found that the generalization performance of the solutions does not critically depend on choosing a specific dimensionality. Also the annealing schedule for the α parameter does not appear to be a critical factor and we normally got good solutions for a wide variety of settings. In our experiments, when maximizing G all the vector and matrix components were updated simultaneously at each iteration. One effective method of performing the optimization is conjugate gradient.

When learning the probabilistic model for the concepts and the "don't know" answer, we use a simple steepest descent, with a very small step size, and no momentum. We learn both the σ_R and the r_R in the log domain, to make sure that neither of these measures will become negative. The validation set used to learn the parameters of the probabilistic model does not need

[3]For one-to-many relations we must not decrease the value of α all the way to 0, because this would cause some concept vectors to become coincident. This is because the only way to make $R^c \cdot \mathbf{a}^c$ equal to k_c different vectors, is by collapsing them onto a unique vector.

[4]During this phase, only the σ_R and r_R parameters are learned. All the vectors and matrix components are kept fixed, as found at the end of the maximization of G.

to be large: we normally use sets containing just few complete-able and uncomplete-able triplets per relation. With LRE the most difficult part of the learning is to learn the distributed representations for the concepts; once this is done, we just need to adjust the few parameters of the probabilistic model, and a small validation set is sufficient for this purpose.

All the experiments presented in the following section were repeated several times, starting from different initial conditions and randomly splitting training and test data. In general the various solutions found by the algorithm given these different conditions were equivalent in terms of generalization performance. Both phases of learning are very fast: the algorithms usually converged within a few hundred iterations, and rarely got stuck in poor local minima. This took few minutes for the largest problems which we shall present in the next sections when running our Matlab code on a 1.6Ghz Intel Pentium 4.

3. LRE Results

One problem on which LRE has been tested is the Family Tree Problem (Hinton, 1986). In this problem, the data consists of people and relations among people belonging to two families, one Italian and one English (Figure 2, left). Using the relations {`father, mother, husband, wife, son, daughter, uncle, aunt, brother, sister, nephew, niece`} there are 112 triplets of the kind $(person_1, relation, person_2)$. Figure 2 (right) shows the distributed representations for the people obtained after training with LRE using all the 112 triplets. Notice how the Italians are linearly separable from the English people and symmetric to them; the second and third components of the vectors are almost perfect features of generation and nationality respectively.

In a different experiment, after learning a distributed representation for the entities in the data by maximizing G using 88 triplets, we learned the parameters of the probabilistic model by maximizing F over a validation set constituted by 12 complete-able and 12 uncomplete-able triplets. The resulting system was able to correctly complete all the 288 possible triplets $(R, \mathbf{a}, ?)$. Figure 3 shows the distribution of the probabilities when completing one complete-able and one uncomplete-able triplet in the test set. The generalization achieved is much better than that obtained by any other method on the same problem. The neural networks of Hinton (1986) and O'Reilly (1996) typically made one or two errors when 4 triplets were held out during training; Quinlan's FOIL (Quinlan, 1990) could generalize almost perfectly when 4 triplets were omitted from the training set.

LRE seems to scale up well to problems of bigger size. We have used it on the Large Family Tree Problem, a much bigger version of the Family Tree Problem, where the family tree is a branch of the real family tree of the author containing 49 people over 5 generations. Using the same set of 12 relations used in the Family Tree Problem, there is a total of 644 complete-able triplets. After learning using a training set of 524 complete-able triplets, and a validation set constituted by 30 complete-able and 30 uncomplete-able triplets, the system is able to complete correctly almost all the possible triplets. When many completions are correct, a high probability is always assigned to each one of them. Only in few cases is a non-negligible probability assigned to some wrong completions.

Hierarchical Linear Relational Embedding (HLRE) (Paccanaro & Hinton, 2001) is an extension of LRE that can be used effectively to find distributed representations for hierarchical structures. The generalization performance obtained by HLRE is much better than the one obtained using RAAMs (Pollack, 1990) on similar problems.

3.1. Adding new concepts and relations

An important question is how easily we can add new information to a solution that LRE has learned for a certain set of tuples. In other words, let us imagine that we have already found distributed representations for the concepts and the relations in a certain data set. How easily can LRE adapt these representations in order to incorporate new tuples involving new concepts or new relations?

We carried out a series of experiments in which first we learned an embedding using a set of triplets containing a certain set of concepts. Then, we added one or more triplets involving a new concept, and checked how well LRE was able to modify the existing solution to include the new concept. The author's father appears in 14 triplets of the Large Family Tree Problem described above. We created a reduced data set for that problem, by excluding those triplets. The distributed representation obtained in 8D training on the remaining 630 triplets is shown in Figure 4(top). Notice how the system has clearly recognized the division between males and females, and the generation each person belongs to. Several features are partially identifiable with some of the vector components: for example the 7th component (row) is positive for the second generation females who had children, and negative for the other ones; the 5th component is negative for the third generation females who belong to

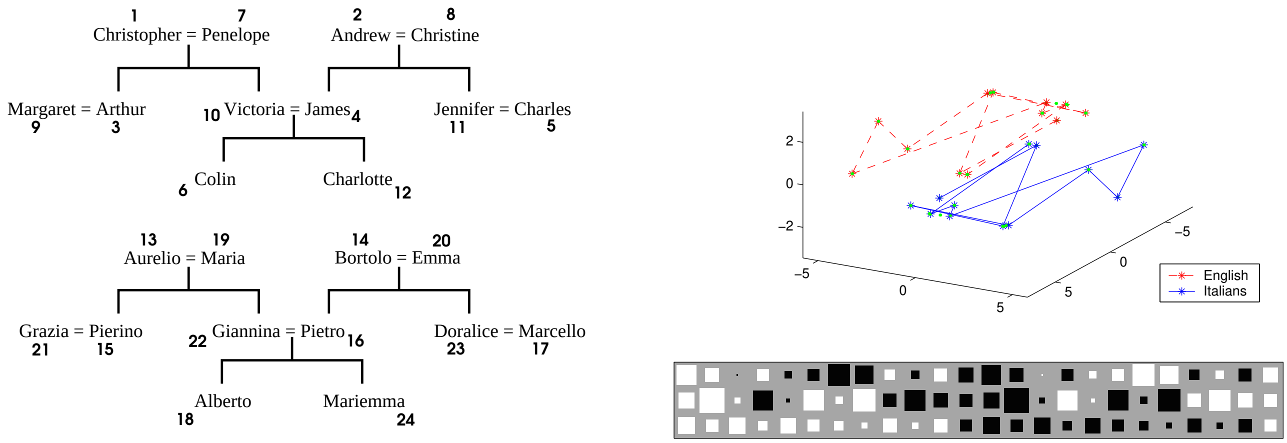

Figure 2. Left: Two isomorphic family trees. The symbol "=" means "married to". Right Top: layout of the vectors representing the people obtained for the Family Tree Problem in 3D. Vectors end-points are indicated by *, the ones in the same family are connected to each other to make them more visible. Right Bottom: Hinton diagrams of the 3D vectors shown above. White squares stand for positive values, black for negative values, and the area of the squares encodes the magnitude of the value. The vector of each person is a column, ordered according to the numbering on the tree diagram on the left.

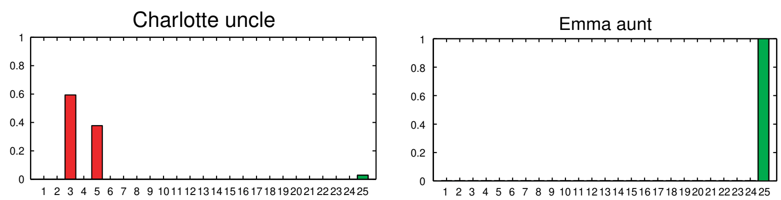

Figure 3. Distribution of the probabilities assigned to each concept for one complete-able (left) and one uncomplete-able (right) triplet written above each diagram. The complete-able triplet has two correct completions but neither of the triplets had been used for training. Black bars from 1 to 24 are the probabilities of the people ordered according to the numbering in Figure 2. The last gray bar on the right, is the probability of the "don't know" answer.

one branch of the tree, and positive for those belonging to the other branch; the exact reverse happens for the last component of the males belonging to the 4th generation. It was then sufficient to add the triplet (`Alberto`, `father`, `Pietro`) and continue training the system for only 80 iterations to obtain the solution shown in Figure 4(bottom). This solution can complete correctly each of the 644 triplets of the Large Family Tree Problem.

We also looked at how well LRE manages to incorporate new relations into an existing solution. We created a reduced data set of the Family Tree Problem containing 11 relations {`husband, wife, mother, son, daughter, brother, sister, nephew, niece, uncle, aunt`}, thus omitting the relation `father`. After using this reduced set to train a solution with LRE, we added to the data set few triplets involving the `father` relation, and continued the learning. Adding just 4 triplets LRE was able to learn the `father` relation and to adapt the existing solution in about 50 iterations. The new solution could complete all the 112 triplets of the Family Tree Problem.

An interesting question is how LRE can adapt a solution to include new sets of data involving the same relations, but a different group of concepts which are not related to the ones previously learned. To do this we tried to learn the English family first, and then to add the Italian family. The results are excellent. After having learned a solution for the 56 triplets relating the English people, less than 100 iterations were always sufficient to learn a distributed representation for all the Italian people, which satisfied all the 112 triplets of the Family Tree Problem. Figure 5(left) shows the solution obtained training LRE using the English family only, while 5(right) shows how the solu-

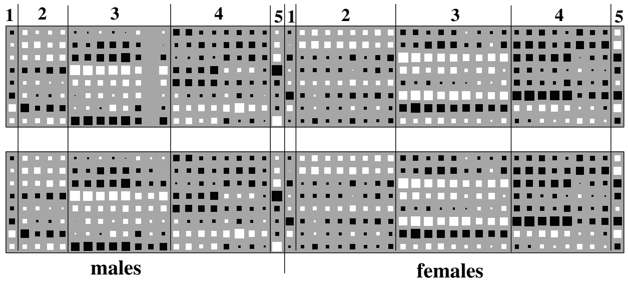

Figure 4. Top: Hinton diagrams of the vectors obtained after learning the Large Family Tree Problem using the 630 triplets that did not contain the author's father. Each column represents a person. The twelfth column, that should have been occupied by the author's father, is filled with zeros (and therefore appears empty). 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. Bottom: the solution found after 80 iterations starting from solution shown above, after having added the triplet (`Alberto`, `father`, `Pietro`) to the data set.

tion was adapted to include the Italian family. Notice how the English family almost did not change while training the Italian family, and that the resulting Italian family is just a rotation of the English family with respect to the origin.

When we tried to learn many families at once, starting from the solution found previously for the English family alone, less than 100 iterations were always sufficient for LRE to adapt the solution to include isomorphic Italian, French and German families simultaneously. Again, each of these families was a rotated and scaled version of the English family previously learned.

Finally we tried to learn a distributed representation for the Italian family starting from the solution found for the English family alone, but using only few of the 56 triplets available for the Italians. As few as 18 triplets proved to be sufficient in order to learn a representation for the Italian family that was able to correctly complete all the 112 triplets in the family tree data set. Again, learning was extremely fast, requiring less than 100 iterations.

3.2. Comparing LRE with FOIL

A different approach to try to solve the problem of generalizing over relational data is the symbolic one. For example, FOIL (Quinlan, 1990) assumes that relational information can be represented as a set of predicates and learns a definition of each predicate in terms of the other predicates and itself. This is particularly interesting when the data contain a set of basic predicates: FOIL is then able to learn predicates that can be expressed as a combination of the basic ones. The definitions which are learned are very similar to Horn clauses and will then hold for any other set of data.

The approach taken by LRE is quite different, since it learns a distributed representation for both relations and concepts for a given, specific training set. Relations are now seen as functions rather than predicates: the result of applying a relation to a concept is another concept. Relations are represented as linear mappings between concepts in some vector space. The idea behind the scenes is that the distributed representation should make explicit the semantic features implicit in the data: each concept is a combination of semantic features, and relations compute an approximation to the features of one concept from the features of another.

Let us see how FOIL and LRE compare in terms of learning and generalization performance. FOIL can learn the definition of a predicate only when it is possible to define that predicate in terms of the predicates available. Thus, if in the family tree example the data contains only the predicates: *parent*, *child* and *spouse*, FOIL could not possibly learn a definition for *mother* since the predicates available do not carry information about the sex of a person, which is fundamental in order to define *mother*. This is irrelevant to LRE, which is able to learn a predicate, no matter what the other predicates are.

As regards generalization performance, given a set of data, LRE is able to learn a representation that provides perfect generalization on that set of data with many more missing triplets than FOIL: as we mentioned earlier, the generalization results of LRE presented here on the Family Tree Problem are better than the 78 out of 80 when testing on 4 triplets obtained by Quinlan (1990) for the same problem. On the other hand, the definitions for the predicates which are found by FOIL have the advantage to be general, not specific to the set of data from which they are learned. Thus it is effortless to use them on new data, while LRE needs a little additional learning for the new concept vectors, as we have seen earlier in this section.

It would be interesting to be able to combine the high generalization capabilities of LRE together with the advantages provided by logical definitions of the predicates. One way to achieve this is to couple LRE with FOIL. First we can use the LRE solution to increase the set of available data. To do this, for every concept vector $\mathbf{a}$ and every relation R we compute the discrete

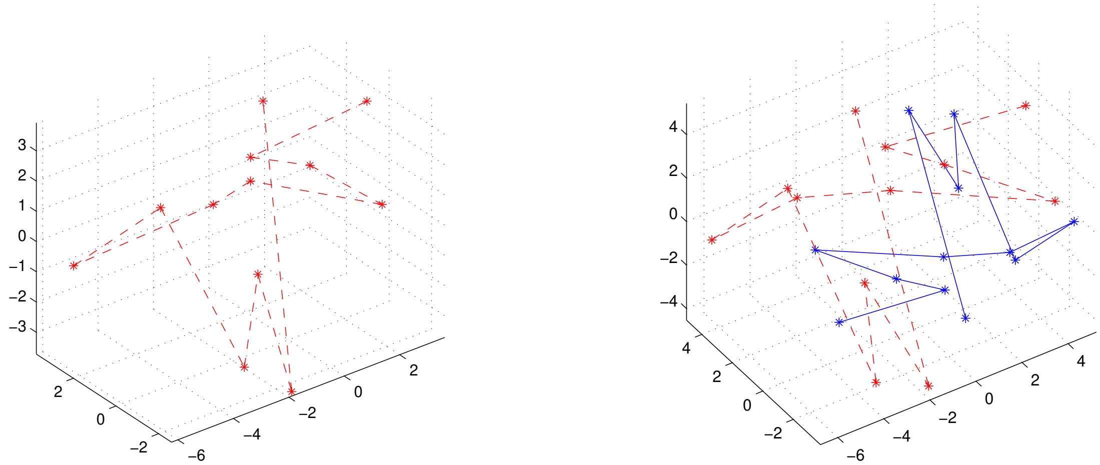

Figure 5. Left: the solution obtained training LRE with the English family alone. Right: the solution obtained after modifying the solution on the left to include the Italian family. Vectors are represented by *, the ones in the same family tree are connected to each other to make them more visible.

probability distribution of each of the concept vectors and the Uniform distribution given $R \cdot \mathbf{a}$ according to our probabilistic model. Then, for each concept vector $\mathbf{k}$ to which the system assigns a sufficiently high probability, the triplet $(\mathbf{a}, R, \mathbf{k})$ could be added the to the data set. Therefore we would end up with a bigger data set of triplets. At this point we could use FOIL to extract logical rules from these triplets. According to the results presented here, using this method we would be able to extract logical rules for the Family Tree Problem from a data set from which 28 triplets are missing, instead of the 4 triplets over which FOIL is able to generalize: the LRE "pre-processing" would come up with all the 112 triplets, and from these FOIL could extract all the Prolog rules. Thus using this method we would obtain the correct logical definitions of the predicates starting from only 88 triplets — thus generalizing 7 times better than if we used FOIL alone.

4. Representational capabilities and limitations of LRE

It is possible to prove that, as long as we consider only *binary* relations, given a sufficient number of dimensions, there always exists an LRE-type of solution that satisfies any set of triplets (Paccanaro, 2002). This means that, no matter how many concepts we have in our data set, and how many binary relations between these concepts hold simultaneously, there will always exist a mapping from the concepts into a space where these binary relations are linear transformations — although the space might need to be very high-dimensional.

The extension to higher arity relations, however, is not straightforward. In this case, our data consists of $(m+1)$-tuples, of the kind $\{concept_1, concept_2, \cdots, concept_{m-1}, Relation, concept_m\}$, and the problem we are trying to solve is again to infer the missing tuples when we are given only few of them, that is to complete the last element of the $(m+1)$-tuples given the m previous ones. We could extend LRE in order to handle m-arity relations by simply concatenating the first $(m-1)$ concept vectors in each tuple into a single vector and then maximizing G as before. However, the fact that the transformations between distributed representations are constrained to be linear, carries with it a fundamental limitation when we try to learn relations of arity higher than 2. To see this, we can think of the solutions that LRE finds as a set of one-layer neural networks of linear units, one network for each relation in our problem; during learning we learn both the weights of these networks, and the input and output codes, with the constraint that such codes must be shared by all the networks. Linear neural networks can only represent and learn functions which are linearly separable in the input variables. This limitation transfers to LRE which will only be able to learn a distributed representation for a small class of relations with arity greater than 2. Let us look at some classical examples. We can think of the Boolean functions as two-to-one (ternary) relations, where the Boolean values $[0, 1]$ are the concepts. LRE can find a solution for the *AND* function (see Figure 6, left). However, LRE cannot learn the *XOR* problem, which would require two disjoint regions with a high probability of obtaining concept 1. As it happens for one layer perceptrons,

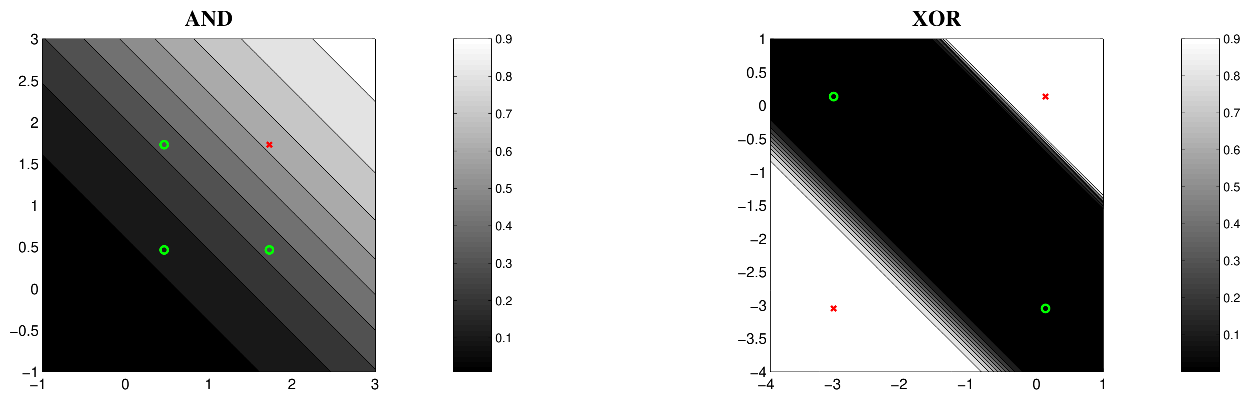

Figure 6. Probability assigned to concept 1 for the points in a region of the plane for a solution to the *AND* Boolean function using LRE (left) and to the *XOR* Boolean function using NLRE (right). The four points corresponding to $(1,1)$, $(0,1)$ $(1,0)$ and $(0,0)$ are marked by a circles and crosses. Probabilities are computed based on a probabilistic model which has a spherical Gaussian with variance equal to $1/2$ centered on each concept.

LRE can only capture the pairwise correlation between input and output variables: it will not be able to learn relations for which such correlation is 0 while all the information is of higher order.

5. Non-Linear Relational Embedding

The standard way to overcome the limitation of one-layer networks, is to add a hidden layer of non-linear units. In a similar way we can obtain a non-linear version of LRE, which we have called Non-Linear Relational Embedding (NLRE). For each relation we introduce an extra layer of hidden units with a bias, $\mathbf{h}$, and an extra set of weights, S (see Figure 7 (left)). The hidden units will have a non-linear activation function — we have used the sigmoid function. Equation 5 becomes:

$$G = \sum_{c=1}^{C} \frac{1}{k_c} \log \frac{e^{-\|S^c\cdot\sigma(R^c\cdot\mathbf{A}^c+\mathbf{h}^c)-\mathbf{b}^c\|^2}}{[\sum_{\mathbf{v}_i\in\mathcal{V}} e^{-\|S^c\cdot\sigma(R^c\cdot\mathbf{A}^c+\mathbf{h}^c)-\mathbf{v}_i\|^2}]^{\alpha}} \qquad (7)$$

where $\mathbf{A^c}$ denotes the vector resulting from concatenating vectors $\mathbf{a_1},\cdots,\mathbf{a_{m-1}}$ representing the first $(m-1)$ concepts in tuple c, and σ indicates the sigmoid function. During learning we maximize G with respect to the matrices, R^c, S^c, the biases of the hidden units, $\mathbf{h}^c$ and the distributed representation of the concepts, while annealing α. As for LRE, after optimizing G we can learn the parameters of a probabilistic model, σ_R and r_R, by maximizing:

$$F = \sum_{q=1}^{Q} \log \frac{\exp(-\frac{r_R^2}{2\sigma_R^2})}{\exp(-\frac{r_R^2}{2\sigma_R^2}) + \sum_{\mathbf{v}_i\in\mathcal{V}} G^q_{(\mathbf{A},\mathbf{R},\mathbf{v_i})}}$$

$$+ \sum_{p=1}^{P} \frac{1}{k_p} \cdot \log \frac{\exp(-\frac{\|S^p\cdot\sigma(R^p\cdot\mathbf{A}^p+\mathbf{h}^p)-\mathbf{b}^p\|^2}{2\sigma_R^2})}{\exp(-\frac{r_R^2}{2\sigma_R^2}) + \sum_{\mathbf{v}_i\in\mathcal{V}} G^p_{(\mathbf{A},\mathbf{R},\mathbf{v_i})}}$$

where:

$$G^q_{(\mathbf{A},\mathbf{R},\mathbf{v_i})} = \exp(-\|S^q\cdot\sigma(R^q\cdot\mathbf{A}^q+\mathbf{h}^q)-\mathbf{v}_i\|^2/2\sigma_R^2)$$
$$G^p_{(\mathbf{A},\mathbf{R},\mathbf{v_i})} = \exp(-\|S^p\cdot\sigma(R^p\cdot\mathbf{A}^p+\mathbf{h}^p)-\mathbf{v}_i\|^2/2\sigma_R^2)$$

NLRE contains LRE as a special case, and with a sufficient number of hidden units is able to represent any relation. Figure 6(right) shows a solution to the *XOR* problem found using NLRE.

We can see NLRE as a variation of the backpropagation algorithm that allows the neural network to learn the weights and the biases of the units as well as a distributed representation of the input and output vectors. This can lead to a much better generalization performance since the system will choose to represents the inputs and the outputs using those features that are relevant to the problem.

To show this, we can use NLRE to solve the Family Tree Problem, this time thinking of each person and each kinship relation as a concept, and having only one "relation" that maps a pair $(person_1, kinship\text{-}relation)$ onto the third person of the triplet. Training using only 100 triplets, NLRE is able to find a solution that can complete all the 112 triplets in the data set using vectors in 3D. Figure 7 (right) shows the vectors representing each of the persons and the kinship relations. It is important to realize that this NLRE architecture is equivalent to the backpropagation architecture used by Hinton (1986) to solve the same prob-

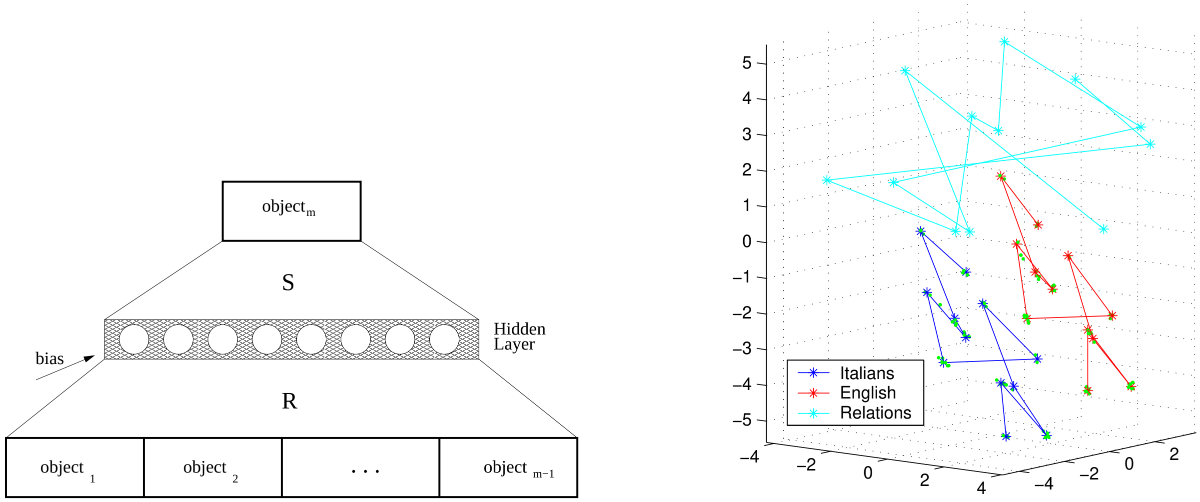

Figure 7. Left: diagram of the neural network corresponding to the implementation of a relation in NLRE. Right: layout of the vectors representing the people and the kinship relations obtained using NLRE for the Family Tree Problem in 3D. Vectors end-points are indicated by *; the ones belonging to the same family and the ones representing the relations are connected to each other to make them more visible.

lem. However, NLRE can generalize 3 times better: this is due to the fact that NLRE learns a distributed representation of the input and output vectors.

The training of NLRE is in general more involved than the training of LRE since the model has many more parameters. Moreover, while for LRE the user has just to choose the size of the distributed representation of the concepts, with NLRE the user has also to choose the size of the hidden layer of the relations, and eventually a weight decay parameter. However, in the experiments that we have carried out, convergence was usually quite fast, and the system seldom got trapped in poor local minima. As for LRE, we can build a hierarchical version of NLRE that can learn distributed representations for structured data (Paccanaro, 2002).

6. Discussion

In this paper we have discussed methods for generalizing over relational data. The approach has been to learn proper distributed representations for concepts and relations that coincide with their semantic features and then to use these representations to make inferences.

Linear Relational Embedding learns a mapping from the concepts into a feature-space by imposing the constraint that relations in this feature-space are modeled by linear operations. LRE shows excellent generalization performance. The results on the Family Tree Problem are far better than those obtained by any previously published method. Results on other problems are similar.

We have discussed the representational capabilities of LRE, and particularly we have shown that extending LRE to relations of arity greater than 2 is limited in the type of relations that it can represent.

We have introduced NLRE, a method that finds distributed representations for the concepts allowing the relations to be non-linear transformations in feature-space. We can see NLRE as a variation of the backpropagation algorithm that allows the neural network to learn a distributed representation of the input and output vectors as well as the weights and the biases of the units. As α approaches 0, NLRE becomes the classical backpropagation algorithm (with the difference that the representations for the input and output vectors are also learned) with sum-of-squares error function — however, nothing prevents us from using a different error function.

We believe that NLRE is a powerful method, that could be quite effective for all those tasks in which input and output vectors belong to certain predefined classes. Often this kind of problem is solved using an MLP trained with backpropagation, after the user has chosen a vector codification for the inputs and the outputs elements in the training pairs — a typical choice being the one-out-of-n codification. If instead of having the user choose such codes, we learn appropriate distributed representations for the input and output elements (together with the weights of the network), this can lead to much better generalization. In fact the system will be performing a clever, automatic kind of feature extraction, in which it chooses to represents the inputs and the outputs using those features that are relevant to the problem. The results on the Family Tree Problem show that generalization using NLRE can be much better than those achieved by just learning the weights of the MLPs.

Moreover, since the system will find a distributed representation according to the relations binding input and output elements, it will assign similar representations to elements which are semantically similar according to those relations. And for many problems we can assume that the non-linear mapping between inputs and outputs will be a smooth function of their feature values, and a small change in the input element features should induce small changes in the output element features. Therefore, although the size of the training set will be limited, learning the input-output weights on a particular training case will improve the performance of a combinatorial number of other cases, not contained in the training set. This point has recently been made very clear by a most interesting work by Bengio et al.(2000). In this paper the authors propose an approach for learning the conditional probability of the next work given few previous words which learns a distributed representation of the words in the text.

Acknowledgements

This research was carried out in collaboration with Geoffrey Hinton who I wish to thank for the many ideas and the exciting discussions. I would also like to thank Zoubin Ghahramani, Jay McClelland and Peter Dayan for many useful discussions.

References

Bengio, Y., Ducharme, R., & Vincent, P. (2001). A neural probabilistic language model. *Advances in Neural Information Processing Systems - NIPS 13*. Cambridge, MA: MIT Press.

Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, 1–12. NJ: Erlbaum.

Miikkulainen, R., & Dyer, M. G. (1989). Encoding input/output representations in connectionist cognitive systems. In D.S.Touretzky, G.E.Hinton and T.J.Sejnowski (Eds.), *Proceedings of the 1988 connectionist models summer school*, 347–356. San Mateo, CA.: Morgan Kaufman.

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

Paccanaro, A. (2002). *Learning distributed representations of relational data using linear relational embedding*. Doctoral dissertation, Computer Science Department, University of Toronto.

Paccanaro, A., & Hinton, G. E. (2000). Learning distributed representations by mapping concepts and relations into a linear space. *Proceedings of the Seventeenth International Conference on Machine Learning - ICML 2000* (pp. 711–718). Stanford University, San Francisco: Morgan Kaufmann Publishers.

Paccanaro, A., & Hinton, G. E. (2001). Learning hierarchical structures with linear relational embedding. *Advances in Neural Information Processing Systems - NIPS 14*. Cambridge, MA: MIT Press.

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

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*, 239–266.