Distributed Ontological Encoding Through Symbol Recirculation

María Jimena Costa[1] and Pablo Duboue[2]

[1] Facultad de Matemática, Astronomía y Física
Universidad Nacional de Córdoba
Córdoba, Argentina
jime_cm@gmx.net,

[2] Computer Science Department
Columbia University
New York, USA
pablo@cs.columbia.edu,
http://www.cs.columbia.edu/~pablo

Abstract. In this work, we investigated an existing technique that obtains distributed representations of hierarchical semantic networks via symbol recirculation [6]. We are interested in symbol recirculation as a viable method for incorporating hierarchical knowledge into existing machine learning methods. In this paper, we will discuss our re-implementation of the symbol recirculation algorithm and analyze its behavior, learning capacity and robustness using a real life ontology such as WordNet.

Key words: Connectionist Natural Language Processing, Neural Networks, Symbol Recirculation.

1 Introduction

Most of the automatic learning techniques used at present (neural nets [7], decision trees [16], hierarchical lists [24], example memorization [25], inductive rule learning [4]) conceive the learning process as a technique to teach the machine to associate input to output vectors. These vectors may have different representations (numbers, characters, etc.). However, the size and nature of those vectors is fixed and, quite often, arbitrary in nature (for example, a vector of randomly generated floating point numbers, characters, etc.).

In the case of Natural Language Processing, words are normally considered entities with an equality operator. The only thing that can be said about words are whether or not a pair of them are exactly the same word. We consider this type of approach as akin to having *empty* words, words without any underlining meaning.

However, incorporating knowledge about the relations among words can impact positively; let us suppose, for example, that we want to learn what type of words may be the subject of a certain verb, for instance, the verb *"eat."* We have a number of examples for training our system, in which the following words appear as subjects of that verb, their frequency denoted between parentheses: **dog** (5), **cat** (6), **government** (1), **person** (11), **platypus** (2). It is clear that metaphoric uses of the words (such as *"the government eats its words"*) must be discarded. Other examples of rather unusual words (like *platypus*) should be discarded as well. However, if we know that **dog, cat, person** and **platypus** are related because they are all subclasses of the concept ANIMAL, then we should be able to use the training examples from the most frequent classes as evidence for the sporadic ones.

We are interested in investigating a method to incorporate this hierarchical information (such as **dog** IS-A[3] ANIMAL) by replacing the empty symbol, e.g., **dog**, with a vector of floating point numbers that keeps a certain relationship with other vectors that represent semantically similar symbols. These symbols will then each contain a distributed representation of the semantic network that they constitute, together with their relationships.[4]

Therefore, it is the aim of this work to investigate a technique for obtaining distributed representations of hierarchical knowledge structures by means of symbol recirculation, as a viable method to incorporate such knowledge into existing methodologies for automatic learning. In this paper, we will focus specifically on symbol recirculation and on its capacity for learning and robustness. We have re-implemented a symbol recirculation technique proposed by Dyer et al. [6] and performed extensive testing with real life ontologies, complementing their work.

We begin by giving some basic definitions and describing the recirculation process and other methods we have used. We then focus the discussion on the experiments that we performed, followed by our results, and the conclusions that we were able to draw from them, together with possible further work.

1.1 Related Work

The closest method to obtain *intelligent* symbols is ***latent semantic analysis*** (LSA) [5], where the counts of words that co-occur in a context with a given word are recorded as the columns of a large (and sparse) matrix. When all words (columns) are put together, the resulting matrix decomposed via its spectral decomposition. The most dominant n autovectors (a given parameter) are kept as representing the semantic for each word. The idea behind LSA is that similar words will be used in similar contexts [20]. LSA has been shown to help learning in several situations (e.g., [12]). The success of LSA encourages us for seeking new sub-symbolic representation via symbol recirculation. A difference with LSA, although, is that the symbols obtained via recirculation encode an existing ontology, instead of relying on word contexts.

[3] Also called *hyponym,* a semantic relationship between word meanings denoting subordination.

[4] This type of information receives the name of **selectional preferences**, and have been showed to positively impact Word Sense Disambiguation [18].

Our use of WordNet as a semantic network is shared with uses of WordNet to measure the ***semantic similarity*** between words [1]. This technique has also been successful in providing semantic features for machine learning (e.g., [8]). However, WordNet structure is designed as a lexicographic knowledge base, where the number of links between two concepts does not necessarily reflect a semantic distance between them. Symbol recirculation can profit from WordNet to provide information-loaded symbols and then let the machine learning machinery decide upon a problem-dependent similarity metric.

A related sub-symbolic representation proposed in the literature is to use random vectors for each word. This approach does surprisingly help learning [13]. Symbol recirculation tries to pick this line of work, providing more meaningful vectors.

Our final goal of incorporating hierarchical information into machine learning systems is also the goal of ***graph data mining*** [23] and ***inductive logic programming*** (ILP) [14]. Graph data mining is a very new field that we hope may find the sub-symbolic representations obtained via symbol recirculation of great use. ILP is a logic programming formalism that offers an alternative, purely symbolic, approach.

2 Methods

We may define a knowledge network or **ontology** as a directed graph whose nodes, all of which belong to the set $\{\ldots A_i \ldots\}$, are related by means of arrows $\{\ldots r_j \ldots\}$.

Our main goal is to make use of an existing method (recirculation, proposed in [6]) in order to obtain *intelligent* representations of the concepts that constitute the ontology, as opposed to the representations arbitrary in nature that most methods have used until now. The intelligence behind the symbols is the hierarchical information about their role in the ontology. This fact make our codifications of the concepts aware of their underlining semantics. The representations thus obtained are expected to have a positive influence on the process of machine learning, although we are still on the process of performing experiments to justify that claim.

The essence of the recirculation method consists of keeping two types of feed-forward neural networks: several STMs (*Short Term Memories*) and a LTM (*Long Term Memory*), all of which have their *input*, *output* and *hidden* layers. Each STM$_i$ acts as a short term memory for a given node A_i in the ontology. That is to say, if node A_i participates in the following relationships in the semantic net:

$$\begin{gathered} r_1(A_i, A_p) \\ r_2(A_i, A_q) \\ r_3(A_i, A_r) \end{gathered}$$

then we train the net STM$_i$ so that, presented with the representations of r_1, r_2 or r_3, it outputs the representations of nodes A_p, A_q and A_r, respectively. In order to achieve that, we use the technique of back-error propagation [19].

LTM acts as a long term memory and codificates the entire semantic network. The LTM is an autoassociative neural network in which the output of the hidden neurons *compresses* the inputs. The term ***handle*** will denote the pattern that is formed in the hidden layer of the LTM when, once trained, it is presented with a particular input. If we takes as input i of the LTM the weights of STM$_i$, then the LTM *compresses* each STM's weights into a *handle* and *memorizes* those weights implicitly in its own weight matrix. The need for compression should be clear from the fact that the LTM receives as input the weight matrix of the STM and produces handles in the LTM's hidden layer, but these handles are the output of the STM. The relationship existing between the number of nodes in the output layer of a neural network and the total number of weights in the network makes clear the level of compression performed by the LTM.

We also have a Distributed Symbol Memory (*DSM*) which stores the *handles.* As the recirculation process takes place, the *handles* will change until they converge to a final representation for each symbol. It is called *distributed* because the symbols that are stored on it contain information about the entire ontology and the role each of them plays in it.

Initially, the relationships r_j are represented as a random array of 0's and 1's. In order to minimize interferences and accelerate the initial learning process, linearly independent representations are chosen.

The algorithm may be summed up as follows:

1. Train each STM$_i$ with all the *input-output* pairs (r_j, A_k) associated with node A.
2. Once the STM$_i$ are trained, store their weight matrices in the *arrays* S_i.
3. Train the LTM to autoassociate the set $\ldots S_i \ldots$. Once this step is over, the floating point vector number generated at the hidden layer for each S_i is saved in the DSM as the new *handle* of the node that the STM$_i$ represented. The *handles* that represent isolated nodes are not modified. At this point, if the hidden layer of the LTM is presented with the newly updated *handles* in the DSM, we will be able to reconstruct the STMs with the generated output.
4. At the end of the training process, we present the LTM with each of the S_i vectors and the activation patterns (*handles*) formed at the hidden layer are stored in the DSM. These patterns will be the new representations of the semantic net symbols.
5. Steps 1 through 4 constitute an ***epoch***. The *handles* of the last two *epochs* are compared and, if the sum of the squared differences is smaller than a certain predefined value, the *handles* are considered to have converged, and that is the final representation for each of the symbols in the ontology. Otherwise, another *epoch* begins with the training process of the STMs and the LTM with the new *handles.*

In this work, we used the semantic network WordNet [11] as our source for ontologies. We chose it because it is quite a remarkable electronic lexical database (98,538 English words), in which English nouns, verbs, adjectives, and adverbs are organized into sets of synonyms (called ***synsets***), each representing a lexicalized concept. Each word meaning is mapped to a conceptual node (that is, a *synset*), and all the *synsets* are related by means of a variety of semantic relationships, such as KIND-OF (hypernym/hyponym), PART-OF (meronym-holonym), synonyms and antonym. The relationship IS-A is the one that groups *synsets* into hierarchies.

Our implementation of the entire recirculation process is synthesized in Figure 1.

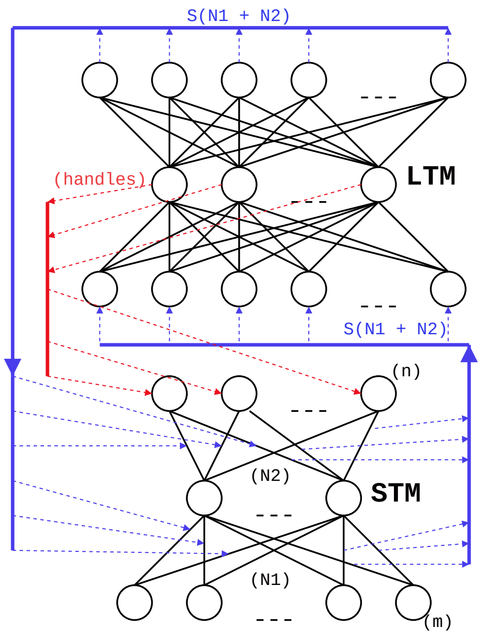

Fig. 1. The recirculation process, adapted from [6].

Figure 2 depicts the full system's implementation. We took several decisions leading to improvements in the recirculation method:

Many to many relationships. Many to many relationships introduce difficulties in the recirculation system. Let us take, for example, the relationship HAS-PART (meronym). If we ask WordNet for the related concepts of the word "car," we obtain 52 results.

Let us suppose that we codificate the first sense of "car," the relationship HAS-PART, and the 52 concepts mentioned before.When we create the STM associated to "car," we find that the neural network must produce 52 different outputs (a different *handle* for each concept) when presented with one input (the symbolic representation of the relationship HAS-PART).

For this reason, we decided that the algorithm for selecting ontologies should choose, in these situations, any one of the related concepts. This way, it is made

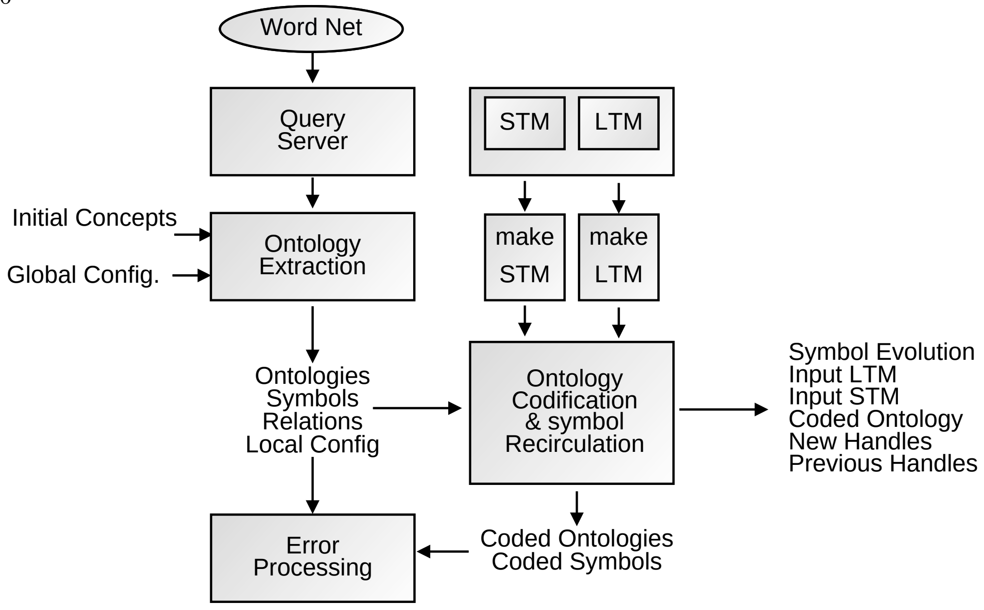

Fig. 2. Full scheme of the system's implementation.

sure that the subgraph so obtained (the ontology) does not present this kind of ambiguities. This is a preliminary solution to the problem posed by many to many relationships.

Improvements to the back-propagation algorithm. In the spirit of increasing the speed of convergence and avoiding undesirable oscillation, we implemented certain improvements to the back-propagation algorithm:

– Each connection weight is given a ***momentum*** rate, so that it tends to change in the direction of the *average* of the "down hill" force, instead of oscillating randomly with each small step. In this manner, the effective learning rate can be increase without divergent oscillations taking place. We chose the parameter ($\alpha = 0.3$) so that the slope of the sigmoid function would be soft, and it does not resemble the step function.
– Selecting an appropriate parameter η for each problem is not an easy task. Even values that are fairly "good" at the beginning of the training process are not as good later on. We decided to implement an adaptive control of the parameters if the weight updates actually decrease the cost function. If it is reduced, then η is augmented. Otherwise, η is decreased. We adopted the de Vogl et al. [22] rule, with constant parameters $\phi > 1$ and $\beta < 1$. Following Reed [17]'s advice, we defined $\phi = 1.05$ and $\beta = 0.7$

Network architecture. We choose three-layer STM and LTMs (with one input, one hidden and one output layer each). The hidden neurons compute the ERF functions on their input, and the outer layer has LINEAR neurons. The initial symbols are arrays of 0's and 1's and, although they are modified through the *epochs*, they remain in that interval.

LTM is an autoassociative neural network, with as many hidden neurons as output neurons each STM has, and as many LINEAR output neurons as weights for each STM.

The choice for the architecture is a consequence of a careful analysis of the characteristics of the nets and the values they handle. The recirculation process showed greater convergence (both quantitively and qualitatively) after the STMs and LTM were constructed so as to reproduce the chosen scheme.

Since the initial codifications of the relationships are linearly independent, we decided to work with as many input neurons as relationships there are in each ontology.

Ontology extraction. We used the ***automatic spread activation*** technique [15, 3, 2]. We developed an approximation to this method was created and incorporated ***depth first search*** [21] with bounded depth for the selection of ontologys from large semantic networks, such as WordNet. For this purpose, we have used the **noun** subtree and the *hypernim* relationship, although the algorithm may be used in any category and with any other relationship.

3 Experiments

We performed our tests using sets of 190 words randomly selected with a uniform distribution, according to their amount of initial meanings (size of their synsets, in WordNet parlance). Words with only one meaning (i.e., words that do not need to be disambiguated) were discarded. For each word, we picked ontologies of increasing size, beginning with a size equal to twice the number of initial senses of that word, until a maximum of 49 symbols. The lower bound allows us to make sure that each sense of the initial (root) word has at least one concept related to it.

We carried out a preliminary study so as to establish the optimum size for the representations with respect to time and efficiency. Results showed that *handles* consisting of 3, 4 and 5 floating point numbers were desirable.

During the process of recirculation, we measured:

Convergence error for one neural net. Each time an STM or LTM computes its output for the training patterns, the sum of the squared differences with the desired outputs is computed as a measure of convergence for that neural network.

Convergence error for the recirculation process. We compute the sum of the squared differences between the *handles* of the last two cycles (*epochs*), as a measure of the "stabilization" process of those *handles*. Recirculation is considered to have converged when this value reaches a point below a certain number that must be fixed at the beginning of the process.

Evolution error. This error measure is computed at the end of each *epoch* by collecting the generated values in the outer layers of each STM for each relationship-output pair that constitutes its training set. The values are then compared to the *handles* in the DSM. The average is considered as an indicator of the degree of convergence of the whole process. The smaller its value, the more consistent the representations in the DSM are, and the *handle* set is therefore closer to converge.

Degenerative error. Once the recirculation process has finished, we measure the degenerative error by constructing a chain of increasing errors that represent the decay of the symbols that constitute the ontology and that are saved implicitly in the LTM. The successive differences sugest a measure of the degradation of the patterns that are recovered through recirculation.

The mean and variance for the degenerative error are shown in Figure 3. Note that it increases more rapidly with 3-component *handles* than with 5-component *handles*. This may be attributed to the fact that 3 floating point numbers have less expressive power than 5, although they have the advantage of converging faster and converging for slightly larger ontologies. Figure 4 shows the evolution error for 3 and 5-component *handles*.

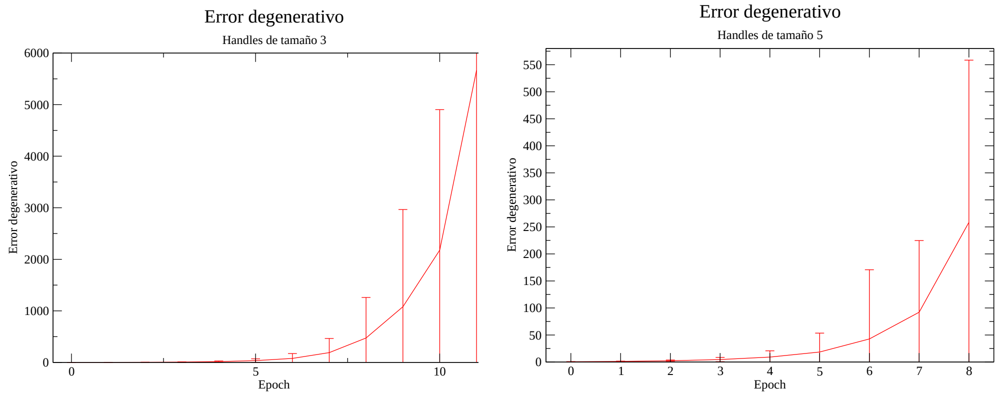

Fig. 3. Degenerative error for 3-component and 5-component *handles*.

The examples that did not converge allow us to conclude that the most critical step of the whole system is the first learning process that takes place in the LTM. If the LTM converges after the first *epoch*, we have an (empiric) warranty of convergence. In successive *epochs*, the time taken to autoassociate decreases considerably, which leads to the conclusion that the generalization that takes place in the LTM is indeed a very good one. The process of recirculation takes only a few fractions of second more than the time taken by the LTM during the first *epoch*. The STMs learn functions so simple that not only they converge in every case, but their convergence times are also minimal. For a 700Mhz processor with an upper bound of 200,000 iterations of Backpropagation (before discarding the example), the maximum convergence time is about a minute.

We attempted to perform a study on the types of large ontologies that the system learned. We could not develop an automatic method, since we could not find a preliminary classification of the ontologies that can be extracted from WordNet. However, we were able to conclude that the ontologies that converge better are “shallow”; that is to say, ontologies whose broadness is greater than their depth (i.e., ontologies originated from words with many initial meanings). This fact was not unexpected, since for that kind of ontology, relatively few STMs are created, and therefore the LTM has fewer patterns to learn.

The process used in this work has showed to converge in reasonably short times, turning it into a viable method of codification, as far as time is concerned, for ontologies of around 20 symbols.

For each word, approximately 20 related concepts are learned, including the initial meanings. After a small study of the maximum sizes for the ontologies that can be extracted from WordNet, we were led to the conclusion that most of them contain around 40 symbols. By means of recirculation we have been able to codificate 50% of that information.

The method has the advantage of permitting the incorporation of meaning into a symbol. That is to say, concepts need no longer be represented by random, meaningless symbols, but rather by *intelligent* ones that incorporate a significant amount of information.

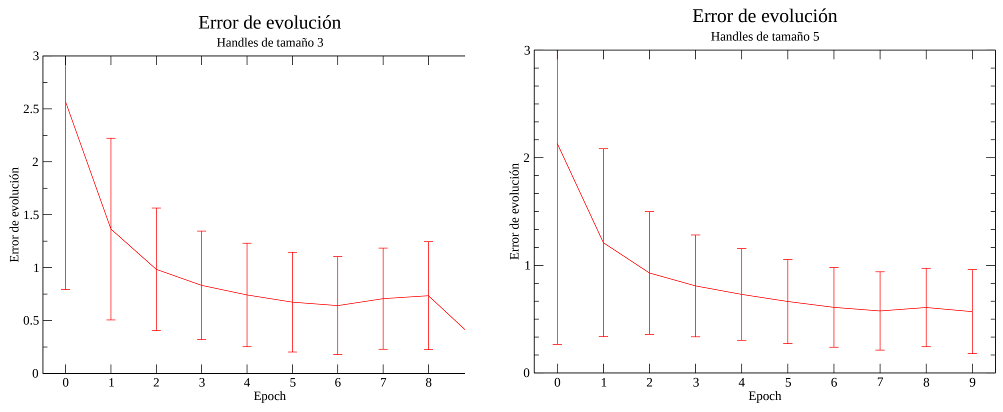

Fig. 4. Evolution error example.

The **convergence** issue is central to the process. The remarkable characteristics of the representations that are obtained by means of recirculation are useless if the set of *handles* stored in the DSM do not converge. Unfortunately, there is no theoretic demonstration of convergence for recirculation methods to this day.

Another problem is that of **oscillation**. If the *handles* oscillate through the *epochs*, they never stabilize into consistent patterns, and the process does not converge. Since *back-propagation* is used during recirculation, any inconvenient that arises with that algorithm (for example, local minima) also affects the performance of the whole process.

A logical question arises: why do the *handles* converge? Let us remember that the operation of autoassociation and compression of the weight matrices of the SMTs in the LTM leads the hidden neurons of this last network to find an efficient representation of the structure of each matrix. Although weight matrices that perform similar tasks may result in very different representations, they share the structural relationships that they code within the semantic network. It is this shared structure that seems to be codificated in the hidden layer of the LTM, and it is therefore that structure the one that guides the subsequent learning process during the recirculation of symbols that represent the concepts.

4 Conclusions

In this work we have analyzed a method based on neural networks used to obtain distributed symbols that incorporate hierarchical information about the concepts they represent.

Using small neural networks that model short term memories, and an autoassociative neural network, that plays the role of a long term memory and codificates the information of the short term memories, we generate a recirculation process that involves the representations of concepts found in semantic networks. This process ends with the incorporation of valuable information in the symbols themselves.

We trained the neural networks, both short and long term, using the back-propagation method, with improved adaptive parameters and momentum. The semantic networks or *ontologies* were obtained from WordNet, which is a semantic net publicly available. Subgraphs were obtained from this network by means of a variation of the *automatic spread activation* method that we developed, and they constituted the ontologies used in this investigation.

We evaluated the performance of the method using different parameter values, such as the size of the ontologies, the parameters and architectures of the neural networks, and the size of the representations (*handles*) among others.

Although we have had serious computational limitations, the recirculation method proved to be successful for almost all the examples (183 out of 190 for the 3 hidden neuron case). However, we have detected that, for the architectures used in this work, the process converges only for ontologies of up to 28 symbols. This might be due to the simplicity of the chosen architectures, especially for the Long Term Memory, for which we have experimented the greatest learning difficulties. In particular, the method presented in this work, together with the architectures used, works best for the so called *flat ontologies*, whose depth is considerably smaller than their broadness.

Our main concern with the original recirculation technique as proposed by Dyer et al. [6] was its scalability. Even with the improvements described in Section 2, we were not able to encode more than 30 symbols. Definitely more research is required to encode the 50K+ symbols of WordNet. At any rate, our working system can serve now as a test-bed for new ideas and let us continue our research on the impact of these symbols for machine learning.

4.1 Further work

The difficulties found lead to the proposition of improvements that would be very interesting to analyze, such as the possibility of adding random restarts or the recirculation process when the LTM does not converge, or the incorporation of a greater quantity of hidden layers in the autoassociative neural network. We also want to look at ways to codificate many-to-many relationships. The effect of using LTM networks with more layers could also be investigated, although preliminary tests showed that the learning process turns out to be much more complicated.

We also intend to continue our research in order to increase the codification capacity of this method in order to be are able to obtain distributed representations for the nearly 50,000 concepts that make up WordNet or similar semantic networks, such as Roget or OpenCYC [10]. Further research on many to many relationships would also be very beneficial for the method.

The next step on this work is to evaluate the impact of the symbols in a machine learning environment. We are looking at Word Sense disambiguation [9] as a possible field to test the extracted symbols.

Acknowledgements

The authors would like to thank Dr. Francisco Tamarit for neural networks advise and insights.

References

1. Butanitsky, A., Hirst, G., *Semantic Distance in WordNet: An Experimental, Application-Oriented Evaluati on of Five Measures*, In Proceedings of WordNet and Other Lexical Resources Workshop, 2001.
2. Burke, B., Gregory, J., M.Bautz, Prigozhin, G., Kissel, S., Kosicki, B.,Loomis, A., and Young, D.: *IEEE Transactions on Electron Devices 44*, 1997.
3. Cohen, P.R., Kjeldsen, R. *Information Retrieval by Constrained Spreading Activation in Semantic Networks.* Information Processing and Management, 23(4), 1987, 255-268.
4. Cohen, W. *Fast effective rule induction.* ML95, 1995.

5. Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., Harshman, R., *Indexing by Latent Semantic Analysis.* Journal of the American Society for Information Science, 41-6, 391–407, 1990.
6. Dyer, M. G., Flowers, M., and Wang, Y.-J. A. *Distributed symbol discovery through symbol recirculation: Towards natural language processing in distributed connectionist networks.* In R. G. Reilly and N. E. Sharkey (eds.), Connectionist Approaches to Natural Language Processing, chapter 2. Hillsdale, NJ: Lawrence Erlbaum Associates, 1992.
7. Hertz, J. ,Krogh, A., Palmer, R. G. *Introduction to the Theory of Neural Computation*, Addision Wesley, 1991.
8. Hirst, G., Budanitsky, A., *Correcting real-word spelling errors by restoring lexical cohesion.* To appear in Natural Language Engineering, 2004.
9. Hatzivassiloglou, V.,Duboue, P.A., Rzhetsky, A. *Disambiguating Proteins, Genes, and RNA in Text: A Machine Learning Approach.* Bioinformatics, Vol. 17 Suppl 1:S97-106, 2001.
10. Lenat, D. B., *Cyc: A Large-Scale Investment in Knowledge Infrastructure.* Communications of the ACM 38, no. 11, November, 1995.
11. Miller, G. A., Ed. *WordNet: An on-line lexical database.* International Journal of Lexicography 3, 4 (Winter 1990), 235-312, 1990.
12. Miller, S., Guinness, J., Zamanian, A., *Name Tagging with Word Clusters and Discriminative Training.* In Proceedings of HLT-NAACL 2004, May 2 - May 7, Boston, Massachusetts, USA, 337–342, 2004.
13. Moscoso del Prado, M.F., Baayen, R. H., *Unsupervised extraction of high-dimensional lexical representations from corpora using Simple Recurrent Networks.* In Pirrelli, Montemagni, & Lenci (Eds.) Proceedings of the ESSLLI'2001 workshop on The Acquisition and Representation of Word Meaning. Helsinki, 2001.
14. Nienhuys-Cheng, S., de Wolf, Ronald, *Foundations of Inductive Logic Programming.* Lecture Notes in Computer Science Vol. 1228, Springer-Verlang, 1997.
15. Quillian, M. R. *Semantic memory.* In M. L. Minsky (Ed.), Semantic information processing. Cambridge, MA: MIT Press, 1968.
16. Quinlan, J.R., *Programs for Machine Learning.* Ed. Morgan Kaufman, 1993.
17. Reed, Russell D., Marks, Robert J. *Neural Smitting. Supervised Learning in Feedforward Artificial Neural Networks.* Cambridge MA: MIT Press/Bradford Books, 1999.
18. Resnik, P., *Selectional preference and sense disambiguation*,In Proc. ACL-SIGLEX, 1997.
19. Rumelhart, D.E, McClelland, J. (Ed.). *Parallel distributed processing: Explorations in the microstructure of cognition.* MIT Press/Bradford Books, 1986.
20. Schütze, H., *Dimensions of Meaning.* In Proc. of Supercomputing '92, IEEE Press, 787–796 1992.
21. Tarjan, R.E. *Depth First Search an Linear Graph Algorithms.* SIAM Journal of Computing, Vol. 1, 146-160, 1972.
22. Vogl, T.P., J.K. Mangis, A.K. Rigles, W.T. Zink, y D.L. Alkon. *Accelerating the convergence of the back-propagation method.* Biological Cybernetics. 59:257-263, 1988.
23. Washio, T., Motoda, H.. *State of the art of graph-based data mining.* SIGKDD Explor. Newsl. **5** 59–68, 2003
24. Yarowsky, D. *Hierarchical Decision Lists for Word SenseDisambiguation.* Computers and the Humanities, 34(2):179-186, 2000.
25. Zavrel, J., Daelemans, W., Van der Sloot, K., y Van den Bosch, A. *TiMBL: Tilburg memory based learner version 1.0.* Technical report, ILK TR 98-03, 1998.