Discrete Multi-Dimensional Scaling

Daniel S. Clouse and **Garrison W. Cottrell**
Computer Science & Engineering 0114
University of California, San Diego
La Jolla, CA 92093
{dclouse,gary}@cs.ucsd.edu

Abstract

In recent years, a number of models of lexical access based on attractor networks have appeared. These models reproduce a number of effects seen in psycholinguistic experiments, but all suffer from unrealistic representations of lexical semantics. In an effort to improve this situation we are looking at techniques developed in the information retrieval literature that use the statistics found in large corpora to automatically produce vector representations for large numbers of words. This paper concentrates on the problem of transforming the real-valued cooccurrence vectors produced by these statistical techniques into the binary- or bipolar-valued vectors required by attractor network models, while maintaining the important inter-vector distance relationships. We describe an algorithm we call *discrete multidimensional scaling* which accomplishes this, and present the results of a set of experiments using this algorithm.

Introduction

Our goal is to develop a connectionist model of lexical access and word sense disambiguation that incorporates a more realistic model of lexical semantics than current models. For the most part, current connectionist models rely either on hand-crafted vector representations of the meanings of words (Kawamoto, 1993; Plaut & Shallice, 1993) or randomly generated vector representations (Cottrell & Plunkett, 1995; Plaut, 1995).

Hand-crafted representations may impose some structure on the fields of the vector (Gallant, 1991; Plaut & Shallice, 1993) in an attempt to provide consistency, or they may be built ad hoc (Kawamoto, 1993). These techniques are motivated by the desire to relate the lexical representations used in the simulation to real words, so that parallels may be drawn to psycholinguistic experiments, and to the intuitions of researchers. There are two main problems with hand-crafted representations. First, they require lots of work to develop, so that the number of words used in a simulation is limited. Second, it is easy to develop a representation without specifying what principles were important in its design.

Randomly-generated representations may be used to avoid both of these problems. The motivation here is usually to generate representations which maintain some interesting set of distance relationships. The parameters can be adjusted to achieve the set of relationships desired. The method is principled and reproducible. Also, the number of vectors which can be generated using this method is essentially unlimited. Unfortunately, using this technique it is not possible to relate the representations to real words, so parallels to specific psycholinguistic stimuli are not possible.

Recently, a third method for generating vector representations of lexical semantics has been gaining acceptance (Schutze, 1993). This method involves gathering word cooccurrence statistics from a large text corpus. Across all occurrences in the corpus of word X, we can count how many times word Y occurs nearby. This number is the cooccurrence count for X and Y. If we gather cooccurrence counts for all pairs of words which occur above a certain frequency, we are left with a large cooccurrence matrix. The row of this matrix corresponding to word X is a vector containing the number of times each other word occurred near word X in the corpus. Insofar as context can be represented by such a "bag of words," this row captures the average context in which word X is seen. Various researchers have proposed methods for refining and reducing the size of the initial cooccurrence vectors. Our method of refinement is to replace each count in the matrix with the mutual information between the row word and column word, then use principal components analysis to shorten the vectors to a reasonable size.

The claim has been made that cooccurrence vectors capture something of the semantics of words. This claim is supported by a number of experiments. Landauer & Dumais (1994), used cooccurrence vectors to pass a portion of the TOEFL (Test of English as a Foreign Language) exam, a test used to evaluate a foreign student's command of English for entry to U.S. colleges and universities. The portion of the test attempted requires the student to choose, from a short list of words, the word which is most similar in meaning to some cue word. To pass the test using cooccurrence vectors, the distance between the cooccurrence vectors for each pair of words was calculated, and the word which was closest to the cue word was chosen. So the distance between the representations of two words serves as a measure of semantic distance.

Schutze (1993) used cooccurrence vectors to tag the senses of ambiguous words in a corpus of text. For each occurrence of an ambiguous word of interest in the text, the cooccurrence vector representations for all nearby words are summed together to produce a context vector. An automatic clustering technique is used to separate these context vectors into groups. All occurrences of the ambiguous word corresponding to context vectors in a single group are tagged with the same sense. This technique works as well as any automatic word disambiguation technique in the literature (89 to 95 percent correct on a short list of words), competing favorably with techniques which start with a more refined source of semantic information such as an on-line thesaurus or dictionary.

Cooccurrence vectors have also been used to model se-

mantic priming. Lund, Burgess & Atchley (1995) present the results of a comparison between a prediction of the size of the semantic priming effect derived using the distances between cooccurrence vectors, and a semantic priming experiment using human subjects. Both the prediction and the experiment used the same set of materials, and show strikingly similar results.

The success of cooccurrence vectors in a variety of semantic tasks suggests them as a good representation in the development of a model of lexical access. The method of generating them is principled and reproducible, the representations can be linked to specific words, and the number of words for which such vectors can be generated is potentially unlimited. So this method appears to combine the best of both the hand-crafted and randomly-generated vector representations.

Unfortunately, there is a problem with the use of cooccurrence vectors for connectionist modeling. A number of recent, successful models of lexical phenomena (Kawamoto, 1993; Plaut & Shallice, 1993; Plaut, 1995) employ attractor networks in their simulations. In these networks, the current output does not rely solely on the input at the current time step, but may be influenced by internal state which has developed over the course of many earlier time steps. The networks are trained to build stable attractors into which activation will settle over time. The time to settle can be measured in different conditions, and compared to reaction time performance of human subjects. This settling performance has been used to account for a number of human priming results: frequency, time course of activation of ambiguous words (Kawamoto, 1993), and semantic versus associative priming (Plaut, 1995). Similar models also exist to explain the effects of neurological damage such as deep dyslexia (Plaut & Shallice, 1993).

Attractor networks tend to work much better when the representations to which they are trained to settle are bit vectors rather than real-valued vectors. We will use the term *bit vector* to refer to a vector whose elements may take on two values. Most often, the two values are either 0 and 1, resulting in a *binary vector*, or -1 and 1, resulting in a *bipolar vector* [1]. When the representations to be stored are bit vectors, the extreme values allowed by the squashing function can be chosen to match those of the bit representation. In the extreme range, a large change in the input to a node has little effect on the output of the node. Thus these extreme ranges make good places to build stable attractors.

So now, we finally get to the main point of this paper! If cooccurrence vectors are to be used as the semantic representation in an attractor network model, we need a way to transform the real-valued cooccurrence vectors into bit vectors. This transformation must be accomplished while maintaining the original distances between vectors in the real space. The remainder of this paper is a report on our attempts to develop an algorithm which performs this transformation, and the performance results of the method which we have found most effective. In the next section we look at possible cost functions to be used by an optimization algorithm. The following section discusses optimization algorithms. Next, we present results of running our algorithm on a number of problems. Finally, we present our conclusions.

[1] Similarly, we will use the term *bit space* to refer to the space spanned by a set of either binary or bipolar axes without specifying which.

Cost Functions

An important constraint on the final representation is that it contain as few bits as possible while still maintaining the original distances between vectors. Thus we share, with the *multidimensional scaling* (MDS) literature (Shepard, 1962; Kruskal, 1964; Borg & Lingoes, 1987) the desire to represent a large amount of data in a smaller space. MDS reproduces a set of proximities (similarities or dissimilarities) defined on some unknown space down to a lower number of dimensions. MDS methods can be divided into metric and non-metric techniques. Metric MDS assumes that the given set of proximities are taken from a metric space. Non-metric MDS makes the weaker assumption that the data was taken from a semi-metric space (i.e. the triangle inequality does not necessarily hold).

The MDS formulation differs from our problem in that, with MDS, the target space is real-valued, and the number of dimensions in the target space is small. In our problem, the target space is a bit space, and the number of bit dimensions required to build an accurate reproduction of the original space may be quite large. Despite the differences, we can easily state our problem in the terms used in this literature.

Let $P = \{(i,j), 1 < i,j < n\}$ be an ordered set of (i,j) pairs that designates the vector pairs for which we have proximities. We will index the elements of P by subscript as p_k. Let $\delta = \{\delta_k : \delta_k = prox(w_i, w_j), p_k = (i,j) \epsilon P\}$ be the set of proximities between items w_i and w_j in the original space, where i and j are designated by P. Then, given a distance metric in s-dimensional bit space, $dist : \mathcal{B}^s \times \mathcal{B}^s \mapsto \mathcal{R}$, $\mathcal{B} = \{0,1\}$ or $\{-1,1\}$, our goal is to produce a set of n vectors, V, such that $C(\delta, d)$ is minimized, where $d = \{d_k : d_k = dist(v_i, v_j), p_k = (i,j) \epsilon P\}$ is the set of distances in the new space corresponding to the δs, and C is a cost function which tells how good is the match between the δs and the ds. The cost function, C determines what it means for the original distances to be preserved in bit space. Therefore, our choice of a cost function is very important.

One fairly general specification of an MDS cost function is the following.

$$K(d,\delta) = \frac{\sum_k \left(d_k - \hat{d}_k\right)^2}{\sum_k d_k^2}$$

If the original proximities, δ_k, are sorted so that $\forall k, \delta_k < \delta_{k+1}$, and $\hat{d}_k$ is chosen to be the monotonically increasing function on δ_k which produces the smallest possible value in the numerator of K, then K is the non-metric cost function known as *Kruskal's stress* (Kruskal, 1964). This cost function is one of the best known in the MDS literature, and may serve as a good choice in non-metric applications.

If $\hat{d}_k$ is chosen to be the regression function, which serves as the best sum of squares prediction of d_k given δ_k, then K is a metric cost function (Borg & Lingoes, 1987 p.42), which we will refer to as *metric stress*, or K_m. For our problem, the proximities are derived from a metric space, so it makes sense to use a metric cost function. We have found K_m to be useful when the distance metric in bit space, $dist$, is chosen to be Euclidean distance. This should also work well with

other Minkowski metrics (including Hamming or city-block distance), but we have not tried using these.

If, instead of using the regression function, we define $\hat{d}_k = \delta_k$, then K is still a metric cost function, but constrains the final distances to match the original distances as closely as possible. We will refer to this as the *exact match* cost function, or K_x. K_x has proven to be a useful cost function when both the distance metric in the original real space, *prox*, and in bit space, *dist*, are the cosine of the angle between two vectors[2]. This distance metric is used extensively in the information retrieval literature, and works well with our cooccurrence vectors. We have found that K_m only works well with the cosine metric when the number of bits in the final representation is small compared to the number of vectors being reproduced. If the number of bits in the final representation is large, K_m can be minimized by making all the output vectors orthogonal to each other. K_x avoids this problem with the cosine metric by setting the slope of the regression line to a constant 1. This is essentially equivalent to minimizing the sum of squared errors[3]. The results reported in the *Results* section are generated using K_x.

One advantage that K_m and K_x have over Kruskal's stress is that they can easily be computed incrementally. Though the time to calculate any of these cost functions from scratch is $O(n^2m)$ where n is the number of vectors, and m is the width of a vector, the time to calculate the change in cost when a single bit is changed is $O(n)$ for K_m and K_x. This is a huge advantage for algorithms which search by changing a single bit at a time, such as those presented in the next section. It may be that a similar savings can be achieved with Kruskal's stress, but the implementation is not obvious.

Search Algorithms

The cost functions described in the previous section allow us to evaluate how good a set of bit vectors are in reproducing the distance relationships in the original set of real vectors. In this section we look at two algorithms for searching the space of possible bit vectors for an optimal solution. Both of these are discrete-space algorithms, meaning only points in the final bit-space are considered as candidate solutions. We are also considering continuous-space algorithms, which consider points in real space as intermediate candidate solutions, but we have had little success with these, to date.

The simplest discrete-space search method we have looked at is a Monte-Carlo method we will call `random-walk`. This algorithm maintains a current set of bit vectors, which we will call the current configuration. At all times, the current configuration contains the set of bit vectors which produce the best cost function value so far. From the current configuration, we search for a better configuration by randomly choosing a single bit of a single vector and seeing what happens to the cost function if that bit is flipped. If flipping the bit results in an improvement in the cost function, then the flipped bit is accepted into the current configuration and the search continues from this new point. If flipping the bit does not result in an improvement, we stay with the current configuration and continue looking at other randomly chosen bits. This kind of iterative improvement continues until there exists no bit which improves the current configuration. At this point the algorithm halts, and the current configuration is returned as the optimal set of bit vectors.

Though you can use a random bit vector as the initial configuration, this algorithm runs faster if the initial configuration is a fairly good one. For bipolar bits, we designate a field of bits to correspond to each real-valued element of the original vectors, and set every bit in the field to the sign of its element. Similarly for binary bits, we set to 1 all bits whose corresponding element is greater than 0.5, and the rest to 0. This, then serves as our initial configuration. There is no constraint in the algorithm to maintain a correspondence between elements in the original vectors and particular fields in the new vectors. Nevertheless, imposing such a correspondence provides a useful method for generating the initial configuration. As an example of how much difference the initial configuration makes, for the 144 bit problem presented in the *Results* section, starting from a random initial configuration required almost twice as much computation as starting using the method presented here, and produced essentially the same cost function value.

A related algorithm is the `optimal-walk` algorithm. In this algorithm, the current configuration is not changed until all possible bit changes have been evaluated. The single bit change which results in the largest improvement in the cost function is accepted into the current configuration, and the search continues from this new point. Like `random-walk`, the algorithm completes when no more improvements can be found. As you might expect, `optimal-walk` requires more time to run than `random-walk`. We had hoped that the the solutions it found might be better than `random-walk`, but this does not appear to be the case. In one trial, we found that `optimal-walk` flipped about 1/3 the number of bits as `random-walk` in the course of finding a solution, but required 50 times as much time overall to run. The final cost function value of the two solutions were almost identical, so `random-walk` appears to perform well compared to `optimal-walk` while requiring much less computation. The results reported in the *Results* section are generated using `random-walk`.

Results

In this section, we present the results of running the `random-walk` algorithm using the cosine K_x cost function on a set of cooccurrence vectors for 233 words which were used as stimuli in Chiarello *et al.* (1990). We started with a 233 by 30000 matrix of cooccurrence counts collected from Internet news groups by Keven Lund and Curt Burgess. We refined this matrix by replacing each cooccurrence count by its mutual information value, then used principal components analysis to reduce each of the 233 vectors to 36 elements. We

[2] Instead of $cos(v, w)$, we actually use $\frac{1-cos(v,w)}{2}$. Using this function, large numbers mean the two vectors are far apart, which is an assumption used in the denominator of K.

[3] In the formula for K and K_m, the $\hat{d}$ function is chosen with knowledge of the d_k. If all output vectors are identical, and thus all $d_k = 0$, it is simple to choose a $\hat{d}$ function which minimizes the numerator. The denominator is included to penalize these degenerate solutions. With K_x, the $\hat{d}$ function is fixed, so there is no longer any need for the denominator. Without the denominator, K_x is exactly the sum of squared errors ($\sum_k \left(d_k - \hat{d}_k\right)^2$). The results reported in this paper include the vestigial denominator.

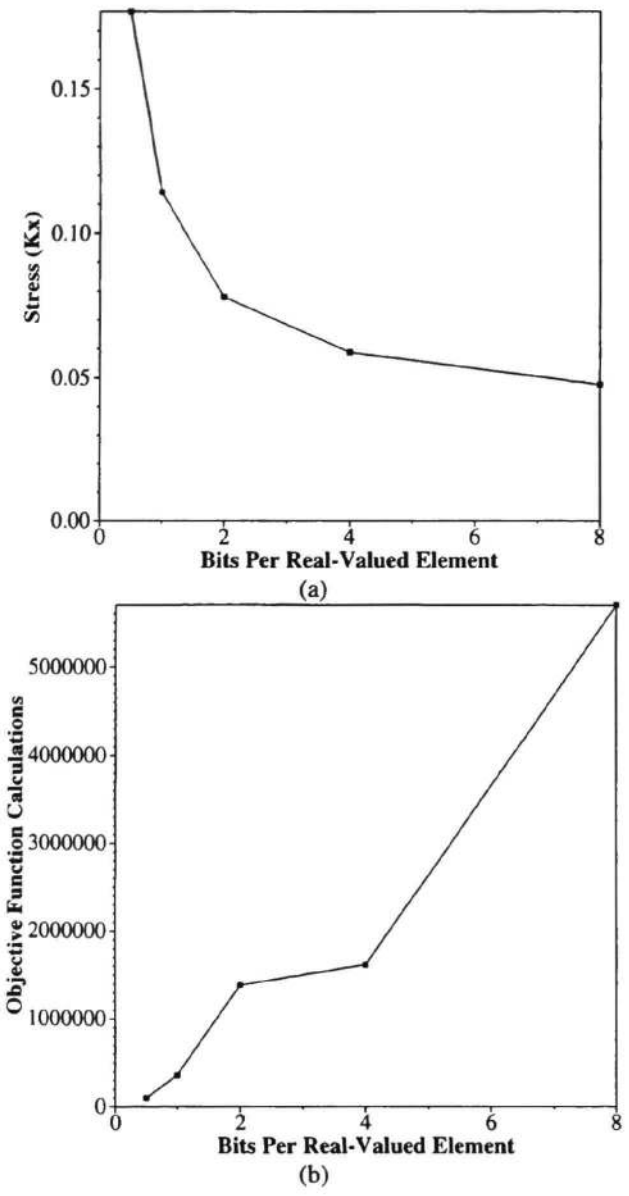

Figure 1: (a) Final Stress at Various Bit Sizes (b) Bit Tests Required

have run the algorithm on other sets of vectors as well with similar results.

Figure 1 (a) plots the minimum value of K_x achieved by a single run of the `random-walk` algorithm at five different bit vector sizes. The x-axis plots the number of bits in each output vector per original real-valued vector element. Since there were 36 real elements in the original vectors, there were 18, 36, 72, 144 and 288 bits respectively in the output vectors portrayed in this graph. Note that by adding more bits to the output representation, we can reduce K_x down to a value somewhat above zero.

Figure 1 (b) plots the number of calculations of the cost function required to generate the stress values in figure 1 (a). This provides a good measure of how the algorithm is affected by the output vector size. Using our incremental technique, the time required for one calculation of the cost function does not depend upon how many bits are included in the output representation, so the dependency on output vector size is simply an indication that that there are more bit combinations to try out with the larger output sizes. On our SparcStation 20, the 18 bit problem required 24 seconds of CPU time to solve. The 288 bit problem required 30 minutes of CPU time.

Figure 2 shows scatter plots of the final configuration for the 36 and 288 bit solutions. Here we plot the distances between every pair of vectors in the original vector space versus the distances between the corresponding vectors in the output space. The horizontal lines in the plots occur because only a limited number of distances are possible between vectors in bit space. Regression lines are also plotted here. Note that as K_x is reduced, the points plotted are pulled in tighter to the regression line. The tightness of the 288 bit solution serves as fairly convincing evidence that we are reproducing the original distances.

We can visualize the success of our algorithm by looking at cluster diagrams. The additive cluster trees (Sattath & Tversky, 1977; Corter, 1982) shown in figure 3 are taken from the original vector space, and the 288 bit vector space. The tree from the original space is on the left. These clusters are extracted from the larger trees which each contain 233 words. Note that the algorithm has maintained the structure of the original trees at two levels. First, these 13 words clustered together in both trees. Second, at a finer level, each tree contains 3 subclusters. Only two words, GOWN and SILK, have changed subclusters between the two trees.

Conclusions

We have demonstrated an algorithm which is capable of transforming real-valued vector representations into bit vector representations while maintaining the intervector distance relationships. This algorithm is capable of generating both binary and bipolar vector representations. It also works with a number of distance metrics, including cosine and Euclidean distance. The algorithm is acceptably fast, and, as we have shown, is capable of finding good solutions. We intend to use this algorithm to help develop realistic semantic representations towards the development of an improved attractor network model of lexical access. One drawback to the current method is that if new words are added, the algorithm must be reapplied. We are currently investigating learning maps between the two spaces that would generalize to nove!

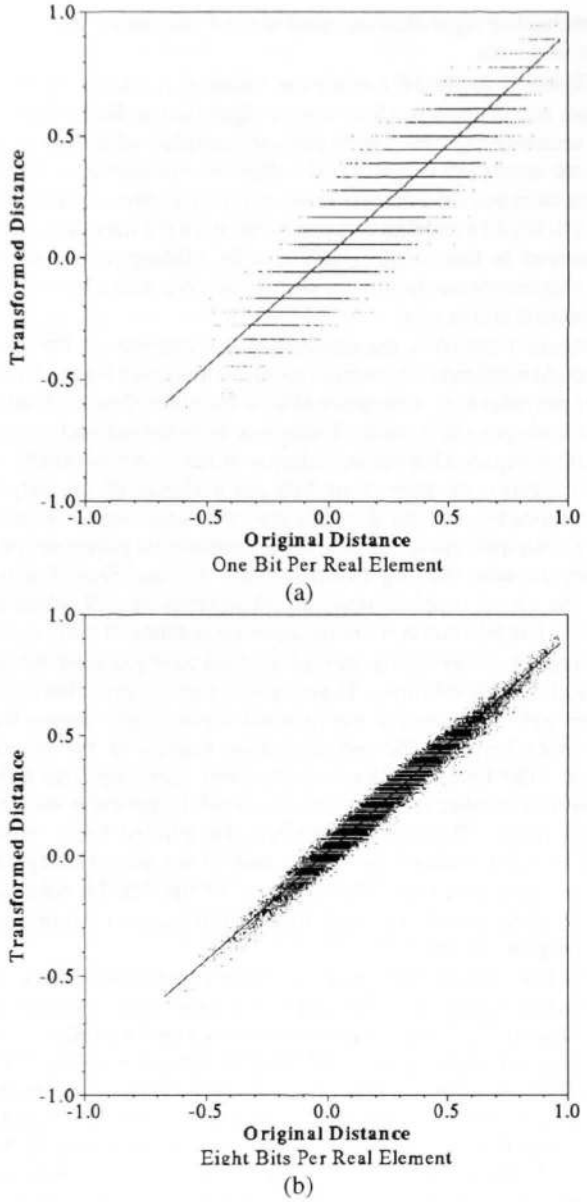

Figure 2: Scatter Plots for Solutions of Different Vector Size

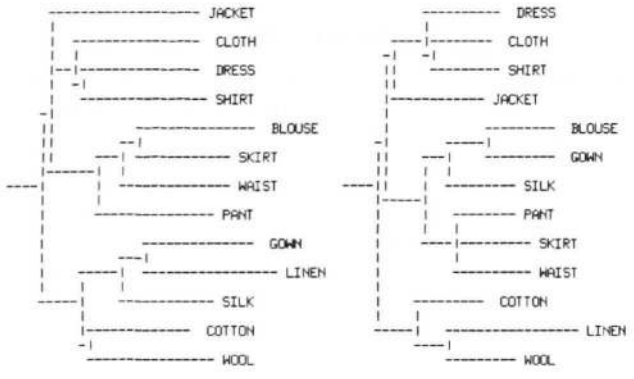

Figure 3: Cluster Trees Before and After Transformation

words.

Acknowledgments

We would like to thank Curt Burgess and Kevin Lund for providing us with the cooccurrence counts we used to develop cooccurrence vectors. Also thanks to the GEURU research group for helpful comments. Daniel Clouse was supported in part by a McDonnell-Pew Predoctoral Fellowship from the San Diego Center for Cognitive Neuroscience, and by a USPHS Predoctoral Traineeship.

References

Borg, I. & Lingoes, J. (1987). *Multidimensional Similarity Structure.* Springer-Verlag, New York.

Cottrell, G.W. & Plunkett, K. (1995). Acquiring the mapping from meaning to sounds. *Connection Science*, 6(4):379–412.

Corter, J.E. (1982). ADDTREE/P: A PASCAL program for fitting additive trees based on Sattath and Tversky's ADDTREE algorithm. *Behavior Research Methods and Instrumentation*, 14(3):353–354.

Gallant, S.I. (1991). A practical approach for representing context and for performing word sense disambiguation using neural networks. *Neural Computation*, 3.

Kawamoto, A.H. (1993). Nonlinear dynamics in the resolution of lexical ambiguity: A parallel distributed processing account. *Journal of Memory and Language*, 32:474–516.

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

Landauer, T.K. & Dumais, S. (1994). Memory model reads encyclopedia, passes vocabulary test. Presented at the Psychonomics Society.

Lund, K. & Burgess, C. & Atchley, R.A. (1995). Semantic and associative priming in high-dimensional semantic space. In Moore and Lehman (1995).

Moore, J.D. & Lehman, J.F., editors (1995). *Proceedings of the 17th Annual Conference of the Cognitive Science Society*, July 22 - 25, 1995, University of Pittsburgh, 1995. Lawrence Erlbaum Associates: Hillsdale, NJ.

Plaut, D.C. (1995). Semantic and associative priming in a distributed attractor network. In Moore and Lehman (1995).

Plaut, D.C. & Shallice, T. (1993). Deep dyslexia: A case study of connectionist neuropsychology. *Cognitive Neuropsychology*, 10(5).

Sattath, S. & Tversky, A. (1977). Additive similarity trees. *Psychometrika*, 42(3):319–345.

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

Shepard, R.N. (1962). Multidimensional scaling with an unknown distance function I & II. *Psychometrika*, 27:219–246 & 390–398.