Similarity-Based Neural Networks for Applications in Computational Molecular Biology

Igor Fischer

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen,
Sand 1, 72076 Tübingen, Germany
fischer@informatik.uni-tuebingen.de

Abstract. This paper presents an alternative to distance-based neural networks. A distance measure is the underlying property on which many neural models rely, for example self-organizing maps or neural gas. However, a distance measure implies some requirements on the data which are not always easy to satisfy in practice. This paper shows that a weaker measure, the similarity measure, is sufficient in many cases. As an example, similarity-based networks for strings are presented. Although a metric can also be defined on strings, similarity is the established measure in string-intensive research, like computational molecular biology. Similarity-based neural networks process data based on the same criteria as other tools for analyzing DNA or amino-acid sequences.

1 Introduction

In respect to underlying mathematical properties, most artificial neural networks used today can be classified as scalar product-based or distance-based. Of these, multi-layer perceptrons and LVQ [1] are typical representatives.

In distance-based models, each neuron is assigned a pattern to which it is sensitive. Appearance of the same or a similar pattern on the input results in a high activation of that neuron — similarity being here understood as the opposite of distance.

For numerical data, distance-based neural networks can easily be defined, for there is a wide choice of distance measures, the Euclidean distance being certainly the best known. In some applications, however, the data cannot be represented as numbers or vectors. Although it may sometimes still be possible to define a distance on them, such a measure is not always natural, in the sense that it well represents relationships between data.

One such example are symbol strings, like DNA or amino-acid sequences which are often subject to research in computational molecular biology. There, a different measure – similarity – is usually used. It takes into account mutability of symbols, which is determined through complex observations on many biologically close sequences. To process such sequences with neural networks, it is preferable to use a measure which is well empirically founded.

This paper discusses the possibility of defining neural networks to rely on similarity instead of distance and shows examples of such networks for symbol strings. Some results of processing artificial and natural data are presented below.

2 Distance and Similarity

Distance is usually, but not necessarily, defined on a vector space. For $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathcal{X}$, any function $d : \mathcal{X}^2 \to R$ fulfilling the following properties is a distance measure on $\mathcal{X}$:

1. $d(\boldsymbol{x}, \boldsymbol{y}) \geq 0$
2. $d(\boldsymbol{x}, \boldsymbol{y}) = 0 \Leftrightarrow x = y$
3. $d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{y}, \boldsymbol{x})$
4. $d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{z}) \geq d(\boldsymbol{x}, \boldsymbol{z})$

For strings, one such measure is the Levenshtein distance [2], also known as edit distance, which is the minimum number of basic edit operations - insertions, deletions and replacements of a symbol - needed to transform one string into another. Edit operations can be given different costs, depending on the operation and the symbols involved. Such weighted Levenshtein distance can, depending on the chosen weighting, cease to be distance in the above sense of the word.

Another measure for quantifying how much two strings differ is "feature distance" [3]. Each string is assigned a collection of its substrings of a fixed length. The substrings – the features – are typically two or three symbols long. The feature distance is then the number of features in which two strings differ. It should be noted that this measure is not really a distance, for different strings can have a zero distance: consider, for example, strings `AABA` and `ABAA`. Nevertheless, feature distance has a practical advantage over the Levenshtein by being much easier to compute.

A similarity measure is simpler than distance. Any function $s : \mathcal{X}^2 \to R$ can be declared similarity – the question is only if it reflects the natural relationship between data. In practice, such functions are often symmetrical and assign a higher value to two identical elements than to distinct ones, but this is not required.

For strings, similarity is closely related to alignment.

> A (global) alignment of two strings S1 and S2 is obtained by first inserting chosen spaces (or dashes), either into or at the ends of S1 and S2, and then placing the two resulting strings one above the other so that every character or space in either string is opposite a unique character or a unique space in the other string. [4]

The spaces (or dashes) are special symbols (not from the alphabet over which the strings are defined) used to mark positions in a string where the symbol from the other string is not aligned with any symbol. For example, for strings `AABCE` and `ABCD`,

```
AABC-E
-ABCD-
```

is an alignment, not necessarily optimal. Each alignment can be assigned a score according to certain rules. In most simple cases, a similarity score is assigned to each pair of symbols in the alphabet, as well as for pairs of a symbol and a space. The score for two aligned strings is computed as the sum of similarity scores of their aligned symbols. Such similarity measure is called "additive". There are also more complex schemes, for example charging contiguous spaces less then isolated. The similarity of the strings is defined as the score of their highest-scoring alignment.

In computational molecular biology, similarity is most often computed for DNA or amino-acid sequences (sequence and string are used as synonyms here), where similarity between symbols is established empirically to reflect observed mutability/stability of symbols. Because each pair of symbols can have a different similarity and no obvious regularity exists, similarities are stored in look-up tables, which have the form of a quadratic matrix. Among scoring matrices, the PAM (point accepted mutations) [5] and BLOSUM (block substitution matrix) [6] families are the most often used.

It is intuitively clear that distance and similarity are somehow related, but quantifying the relationship is not always straightforward. For additive measures, if the similarity of each symbol with itself is the same, it can be established by a simple equation [7]:

$$\langle \boldsymbol{s}_1 | \boldsymbol{s}_2 \rangle + d(\boldsymbol{s}_1, \boldsymbol{s}_2) = \frac{M}{2} \cdot (|\boldsymbol{s}_1| + |\boldsymbol{s}_2|). \tag{1}$$

The notation $\langle \boldsymbol{s}_1 | \boldsymbol{s}_2 \rangle$ is used to symbolize the optimal alignment score of strings $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, $|\boldsymbol{s}|$ the string length, and M is an arbitrary constant. The distance $d(\boldsymbol{s}_1, \boldsymbol{s}_2)$ is defined as the minimal cost for transforming one string into the other. The cost is the sum of the symbol replacement costs $c(\alpha_i, \beta_i)$, α_i and β_i being the symbols comprising $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, and the insertion/deletion costs, each of the operations costing some $h > 0$ per symbol. These values are related to the values comprising the string similarity:

$$\begin{aligned} p(\alpha, \beta) &= M - c(\alpha, \beta) \quad \text{and} \\ g &= \frac{M}{2} - h. \end{aligned} \tag{2}$$

where $p(\alpha, \beta)$ is the similarity score for the symbols α and β and g is the value of space in the alignment (usually a negative one). For the condition (2) on the distance to be satisfied, $c(\alpha, \alpha)$ must be always zero, that is, $p(\alpha, \alpha)$ must be constant for all α. Unfortunately, the above mentioned PAM and BLOSUM scoring matrices do not satisfy this condition.

Especially for searching large databases of protein and DNA sequences, sophisticated, fast heuristics like FASTA [8] and BLAST [9] have been developed. They have been fine-tuned not only for speed, but also for finding biologically meaningful results. For example, from an evolutionary point of view, a compact gap of 10 symbols in an aligned string is not twice as probable as a gap of 20 symbols at the same position. Thus, non-additive scoring schemes are applied. This is another reason why the above described method for computing the distance from similarity is not applicable.

A simple method for computing "distance" from similarity score for proteins was applied in [10]. For computing the score normalized scoring matrices with values scaled to $[0, 1]$ were used, and for spaces a fixed value of 0.1 were applied. The scores for all pairs of proteins from the data set were computed and ordered into a new matrix $\boldsymbol{S}$. The element $\boldsymbol{S}[i][j]$ was the similarity score for the i-th and j-th protein from the data set. This matrix was subsequently also scaled to $[0, 1]$. The distance between i-th and j-th protein was then computed as

$$\boldsymbol{D}[i][j] = 1 - \boldsymbol{S}[i][j]. \tag{3}$$

This approach has several disadvantages: First, the computational and storage overheads are obvious. In most applications pair-wise similarity scores of all data are not needed. Also, this method is not applicable for on-line algorithms, with data sets of unknown and maybe even infinite sizes. But more than that, it is not clear, if the above function is a distance at all. It is easy to see that simple scaling of the $\boldsymbol{S}$ matrix can lead to a situation where the requirement (2) for distance is violated. Such a case appears when the diagonal elements – standing for self-similarities of strings – are not all equal. A workaround, like attempt to scale the matrix row- or column-wise, so that the diagonal elements are all ones, would cause a violation of the symmetry relationship (3). Element $\boldsymbol{S}[i][j]$ would generally be scaled differently than $\boldsymbol{S}[j][i]$ so the two would not be equal any more. And finally, the triangle inequality – requirement (4) – is not guaranteed to be satisfied.

3 Distance-Based Neural Networks

As presented in the introduction, distance-based neural networks rely on distance for choosing the nearest neuron. However, in the training phase the distance is also involved, at least implicitly. All distance based neural networks known to the author try to place or adapt neurons in such a way that they serve as good prototypes of the input data for which they are sensitive. If the Euclidean distance is used, the natural prototype is the arithmetic mean. It can be expressed in terms of distance as the element having the smallest sum of squared distances over the data set. This property is helpful if operations like addition and division are not defined for the data, but a distance measure is provided.

Another kind of prototype is the median. For scalar data, the median is the middle element of the ordered set members. Like mean, it can be expressed through distance. It is easy to see that the median has the smallest sum of distances over the data set. Thus the concept of median can be generalized on arbitrary data for which a distance measure is defined.

Based on distance measures for strings and relying on the above mentioned properties of mean and median, neural networks for strings have been defined. Self organizing map and learning vector quantization – have been defined both in the batch [11] and on-line [12] form. As is often the case, the on-line version has been shown to be significantly faster than the batch form.

Finding the mean or the median of a set of strings is not quite easy. The first algorithms [13] performed an extensive search through many artificially generated strings in order to find the one with the smallest sum of (squared) distances. A later and much faster algorithm [12] used a by-product of computing the distance – the edit transcript – for finding the prototype string. The edit transcript is a list of operations needed to transform one string into another. It is not actually needed for computing the distance, but it can be directly deduced by backtracing through the so-called dynamic programming table, which is an intermediate result when computing Levenshtein or related distance. Starting from an approximation of the prototype, the edit operations needed for transforming it into every of the set strings were computed. Then, at each position, the edit operation appearing most frequently in all transcripts was applied, as long as it minimized the sum of (squared) distances. This is only a best effort approach and it can get stuck in a local optimum. Moreover, since strings are discrete structures, the mean or the median are not always unambiguous. For example, for the set of two strings, { `A`, `B` }, both `A` and `B` are equally good mean and median.

4 Similarity-Based Neural Networks

As mentioned in section 2, for some applications similarity is a more natural measure than distance. Taking the above discussed inconveniences that can appear when using distance, one could attempt to circumvent it altogether. If one were able to define a prototype based on similarity, a neural network could be described by an algorithm almost identical to the distance-based networks: one only needs to search for the neuron with the highest similarity instead of the one with the minimum distance. Of course, if the distance is defined in a way that properly reflects the reality, similarity can be defined as the negative distance and the two paradigms converge.

Computing the prototype based on similarity can indeed be performed, at least for strings. Recall that the similarity between two strings is computed from their optimal alignment by summing the similarity scores of aligned symbols. Three or more strings can also be aligned, in which case one speaks of a multiple alignment. Having computed the multiple alignment (let the question how to compute it efficiently be put aside for the moment), the prototype string $\bar{\boldsymbol{s}}$ can be defined as the string maximizing the sum of similarities with all strings from the set:

$$\bar{\boldsymbol{s}} : \sum_i \langle \bar{\boldsymbol{s}} | \boldsymbol{s}_i \rangle \geq \sum_i \langle \boldsymbol{s}_k | \boldsymbol{s}_i \rangle \quad \forall \boldsymbol{s}_k \neq \bar{\boldsymbol{s}}. \tag{4}$$

Not quite identical, but in practice also often used prototype is the “consensus string”, defined as the string having at each position the most common symbol appearing at that position in all aligned strings.

5 Implementation Considerations

Many of the ideas presented here rely on approximate string matching algorithms. Efficient implementation of a neural network based on string similarity should take care of the following issues: computing the similarity, finding the most similar string in the given set and computing multiple alignment.

5.1 Computing the Similarity

Similarity is computed from aligned strings, and alignment can be found using dynamic programming. The simplest algorithms have quadratic time and space complexity. Both can be reduced under certain circumstances, but also increased if more complex scoring schemes are used (for literature cf. [4,7,14]).

An adaptation of the original algorithm, which includes a divide-and-conquer strategy [15], computes the alignment in linear space for the price of roughly doubling the computation time. If the alignment itself is not needed, but only the similarity score, the computation time can remain the same.

5.2 Finding the Most Similar String

Such a case appears in the search for the neuron most similar to the input string. A simple approach would be to go through all neurons and compute their similarity to the string, retaining the most similar. Alignments between neurons and the string are never needed. This algorithm can further be refined: not only alignment, but even exact similarity need not be computed for all neurons.

The idea is roughly the following: suppose one expects the most similar neuron to have the similarity score above a certain threshold. This expectation can be based on experience from previous iterations or on knowledge about the strings. Then one can go through neurons, start computing the similarity for each and break off the computation as soon as it becomes obvious that the score would be below the threshold. If the threshold was correct, the algorithm finds the most similar neuron by computing the similarity only on a fraction of them. Else, the threshold is reduced and the whole process repeated.

The above idea is implemented by computing only the similarity matrix elements in a limited band around its diagonal, thus reducing the complexity to $O(kn)$ [16], k being the band width and n the string length. The observed speed-up is about an order of magnitude.

5.3 Computing Multiple Alignments

Multiple alignment is considered the “holy grail” in computational molecular biology [4]. Exact computation can be performed by dynamic programming, but with time complexity exponential in the number of sequences. Therefore a number of approximate algorithms have been developed. In this work, the “star alignment” [17] is used, not only for its speed and simplicity, but also because it’s close relatedness to the problem of finding a prototype string.

In star alignment, the multiple alignment is produced by starting from one string – the “star center” – and aligning it successively with other strings. Spaces inserted into the center in course of an alignment are retained for further alignments and simultaneously inserted into all previously aligned strings. That way, the previous alignments with the center are preserved.

The choice of the star center is not critical. At the beginning, one can use the string from the set that maximizes equation (4), and the current prototype in all consecutive iterations. Measurements presented below show that iterative search for the prototype is not very sensitive to the initialization.

6 Experimental Results

To show that computing the string prototype from a set of similar strings is quite robust to initialization, a series of experiments has been performed. In one, the prototype string was computed from a set of strings, all derived from an original random string by adding noise (insertions, deletions and replacements of random symbols), and starting from one of these strings. In repeated runs, keeping the noise level at 50%, the computed prototype converged to the original string in 97% of cases. Even increasing the noise level to 75% still resulted in 80% of prototypes converging to the original string.

In another type of experiment, everything was left the same, except that the algorithm started from a completely random string. Even then, a 89% convergence was observed in case of 50% noise and 62% in case of 75% noise.

Small-scale experiments on similarity-based self-organizing maps were performed on a set of 1750 words generated artificially by adding noise to 7 English words (Figure 1). Care was taken that no original words appear in the set. Even at the noise level of 75% and with random initialization, the resulting 8×12 map converged to a state with the original 7 words placed at distinct closed regions.

In a real-world test, a self-organizing map was tested on a set of 68 sequences from seven protein families. The 12×10 map was initialized by random sequences ordered along their Sammon projection [18]. The map was computed using BLOSUM62 similarity matrix and additive scoring. The map (Figure 2) showed all families distributed on it, although one of them (transducin) much weaker. On the class boundaries, unidentified “mutations” appeared.

Another real-world example is shown in figure 3. The mapping is produced from 320 hemoglobine alpha and beta chain sequences of different species, as used in [19].

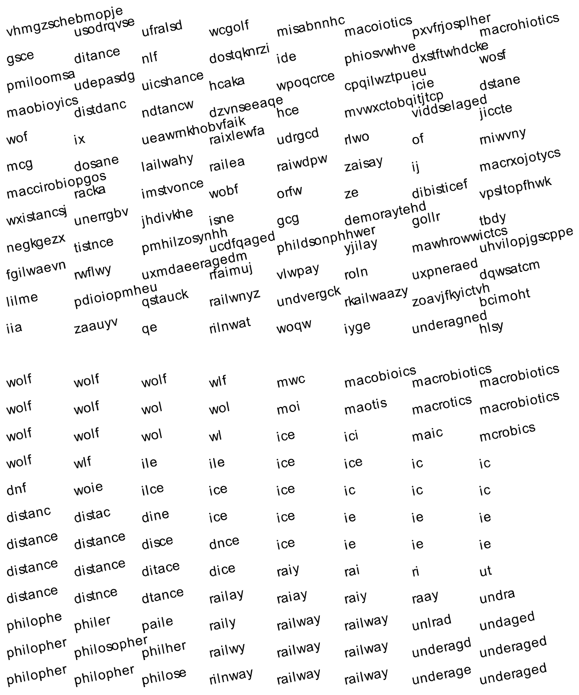

Fig. 1. A simple example of similarity string SOM. The mapping is produced from the set of English words corrupted by 75% noise. **Above:** Initial map with corrupted words randomly distributed over it. **Below:** The converged map after 300 iterations. The algorithm has extracted the original (prototypic) words from the noisy set and ordered them into distinct regions on the map. Artificial "transition" words appear on regions borders.

Table 1. LVQ classification of seven protein family samples, using 15 prototypes, on average two per class. $\boldsymbol{\mu}_{ij}$ denotes the j-th prototype of the i-th class and N_i is the number of elements from class i assigned to the prototype in the corresponding column. In this set, six sequences are incorrectly classified.

	$\boldsymbol{\mu}_{11}$	$\boldsymbol{\mu}_{12}$	$\boldsymbol{\mu}_{13}$	$\boldsymbol{\mu}_{21}$	$\boldsymbol{\mu}_{22}$	$\boldsymbol{\mu}_{31}$	$\boldsymbol{\mu}_{41}$	$\boldsymbol{\mu}_{42}$	$\boldsymbol{\mu}_{51}$	$\boldsymbol{\mu}_{52}$	$\boldsymbol{\mu}_{61}$	$\boldsymbol{\mu}_{62}$	$\boldsymbol{\mu}_{63}$	$\boldsymbol{\mu}_{71}$	$\boldsymbol{\mu}_{72}$
N_1	3	3	4												
N_2				9	2										
N_3					2	6									
N_4							4	5		1					
N_5									5	5					
N_6						1					7	1	1		
N_7				1				1						4	4

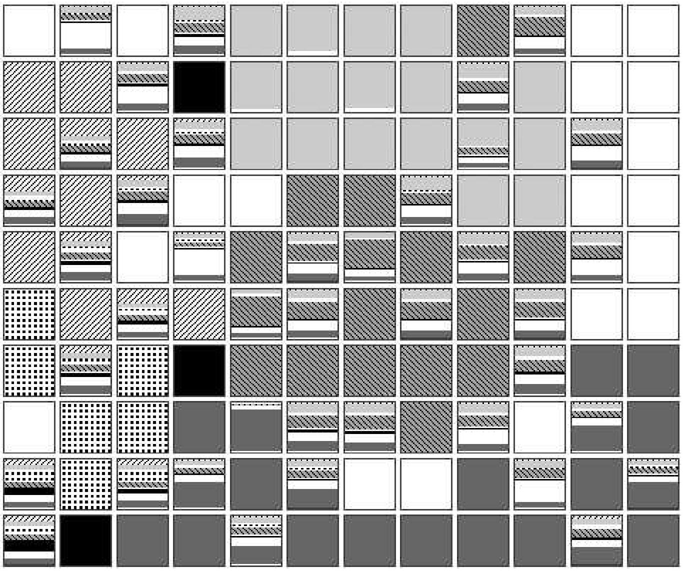

Fig. 2. Similarity string SOM of seven protein families. Each square represents a node in the map , and different families are represented by different fill patterns. The area filled by a pattern corresponds to the node similarity to the protein. All families are well expressed except transducin (the black squares).

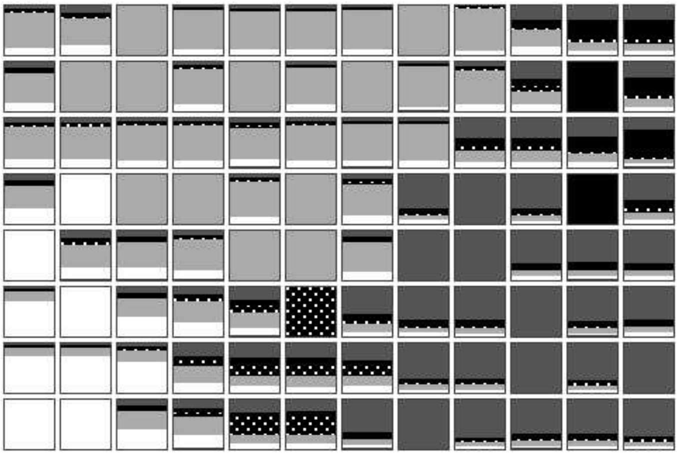

Fig. 3. Similarity string SOM of hemoglobine sequences. Two different chains, α (dark shades) and β (light shades), occupy distinct map areas. Moreover, weakly expressed subclusters can be recognized in each of the chains (white-dotted and black for α, and white for β).

Attempting to classify the seven protein families by LVQ, using 15 prototype strings led to the results presented in Table 1. Six sequences (about 9% of the data set) were incorrectly classified. On the simpler hemoglobine data, perfect classification is obtained by already two prototypes.

7 Conclusion and Outlook

This paper shows that already a simple similarity measure, combined with an algorithm for its local maximization, is sufficient to define a large class of neural networks. A distance measure is not necessary. Even if it is defined, like, for example, for symbol strings, it is not always a 'good' measure. In some applications, like molecular biology, a similarity measure is more natural than distance and is preferred in comparing protein sequences. It is shown that such data can be successfully processed by similarity-based neural networks. It can therefore be concluded that similarity-based neural networks are a promising tool for processing and analyzing non-metric data.

In the presented work, simple, additive similarity measures for strings were used. Further experiments, for example embedding BLAST search engine, are in preparation.

References

1. Kohonen, T.: Learning vector quantization. Neural Networks **1** (1988) 303
2. Levenshtein, L.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics–Doklady **10** (1966) 707–710
3. Kohonen, T.: Self-Organization and Associative Memory. Springer, Berlin Heidelberg (1988)
4. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press (1997)
5. Dayhoff, M., Schwartz, R., Orcutt, B.: A model of evolutionary change in proteins. In Dayhoff, M., ed.: Atlas of Protein Sequence and Structure. Volume 5., Washington, DC., Natl. Biomed. Res. Found. (1978) 345–352
6. Henikoff, S., Henikoff, J.G.: Amino acid substitution matrices from protein blocks. In: Proceedings of the National Academy of Sciences. Volume 89., Washington, DC (1992) 10915–10919
7. Setubal, J.C., Meidanis, J.: Intorduction to Computational Molecular Biology. PWS Publishing Company, Boston (1997)
8. Pearson, W.R., Lipman, D.J.: Improved tools for biological sequence comparison. In: Proceedings of the National Academy of Sciences of the U.S.A. Volume 85., Washington, DC, National Academy of Sciences of the U.S.A (1988) 2444–2448
9. Altschul, S.F., Gish, W., Miller, W., Meyers, E.W., Lipman, D.J.: Basic local alignment search tool. Journal of Molecular Biology **215** (1990) 403–410
10. Agrafiotis, D.K.: A new method for analyzing protein sequence relationships based on Sammon maps. Protein Science **6** (1997) 287–293
11. Kohonen, T., Somervuo, P.: Self-organizing maps of symbol strings. Neurocomputing **21** (1998) 19–30

12. Fischer, I., Zell, A.: String averages and self-organizing maps for strings. In Bothe, H., Rojas, R., eds.: Proceedings of the Neural Computation 2000, Canada/Switzerland, ICSC Academic Press (2000) 208–215
13. Kohonen, T.: Median strings. Pattern Recognition Letters **3** (1985) 309–313
14. Sankoff, D., Kruskal, J.B.: Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison. Addison-Wesley, Reading, MA (1983)
15. Hirshberg, D.: A linear space algorith for computing maximal common subsequences. Communications of the ACM **18** (1975) 341–343
16. Landau, G., Vishkin, U.: Introducing efficient parallelism into approximate string matxhing and a new serial algorithm. In: Proceedings of the ACM Symposium on the Theory of Computing. (1986) 220–230
17. Altschul, S.F., Lipman, D.J.: Trees, stars, and multiple biological sequence alignment. SIAM Journal of Applied Mathematics **49** (1989) 197–209
18. Sammon, Jr., J.: A nonlinear mapping for data structure analysis. IEEE Transactions on Computers **18** (1969) 401–409
19. Apostol, I., Szpankowski, W.: Indexing and mapping of proteins using a modified nonlinear Sammon projection. Journal of Computational Chemistry (1999)