Protein Science (1992), *1*, 667-677. Cambridge University Press. Printed in the USA.
Copyright © 1992 The Protein Society

Protein classification artificial neural system

CATHY WU,[1,2] GEORGE WHITSON,[1] JERRY McLARTY,[2]
ADISORN ERMONGKONCHAI,[1] AND TZU-CHUNG CHANG[1]

[1] Department of Computer Science, The University of Texas at Tyler, Tyler, Texas 75701
[2] Department of Epidemiology/Biomathematics, The University of Texas Health Center at Tyler, Tyler, Texas 75710

(RECEIVED December 9, 1991; ACCEPTED January 10, 1992)

Abstract

A neural network classification method is developed as an alternative approach to the large database search/organization problem. The system, termed Protein Classification Artificial Neural System (ProCANS), has been implemented on a Cray supercomputer for rapid superfamily classification of unknown proteins based on the information content of the neural interconnections. The system employs an *n*-gram hashing function that is similar to the *k*-tuple method for sequence encoding. A collection of modular back-propagation networks is used to store the large amount of sequence patterns. The system has been trained and tested with the first 2,148 of the 8,309 entries of the annotated Protein Identification Resource protein sequence database (release 29). The entries included the electron transfer proteins and the six enzyme groups (oxidoreductases, transferases, hydrolases, lyases, isomerases, and ligases), with a total of 620 superfamilies. After a total training time of seven Cray central processing unit (CPU) hours, the system has reached a predictive accuracy of 90%. The classification is fast (i.e., 0.1 Cray CPU second per sequence), as it only involves a forward-feeding through the networks. The classification time on a full-scale system embedded with all known superfamilies is estimated to be within 1 CPU second. Although the training time will grow linearly with the number of entries, the classification time is expected to remain low even if there is a 10–100-fold increase of sequence entries. The neural database, which consists of a set of weight matrices of the networks, together with the ProCANS software, can be ported to other computers and made available to the genome community. The rapid and accurate superfamily classification would be valuable to the organization of protein sequence databases and to the gene recognition in large sequencing projects.

Keywords: database search; neural networks; protein classification; sequence analysis; superfamily

The continuing rapid growth of the molecular sequencing data has generated a pressing need for advanced computational tools to analyze and manage the data. An ideal computer tool should allow the interpretation of genomic information from the sequences and permit easy organization of the information into a database to facilitate information retrieval. Currently, a database search for sequence similarities represents the most direct computational approach to the analysis of genomic information (Doolittle, 1990). There exist good algorithms and mature software for this task. Sequence comparison algorithms based on dynamic programming (Needleman & Wunsch, 1970) have emerged as the most sensitive methods but have a high computational cost of order N^2 with respect to sequence length N. The FastA program (Pearson & Lipman, 1988) identifies related proteins rapidly using a lookup table to locate sequence identities (Lipman & Pearson, 1985). The QuickSearch method (Devereux, 1988) provides an even faster but less sensitive search against the database that is represented with a sparse hash table. A BLAST approach (Altschul et al., 1990), which directly approximates alignments that optimize a measure of local similarity, also permits fast sequence comparisons. In contrast to the above methods that are designed for pairwise comparisons, a profile analysis method (Gribskov et al., 1987) provides search against information from protein families instead of individual proteins using dynamic programming alignment. Even with the rapid advancement of new search methods, the database search is becoming computationally intensive and increasingly more forbidding due to the accelerating growth of sequencing data. It is desirable to develop methods with a search time that is not constrained by the database size.

Equally important to the development of new database search tools is the organization of second generation databases (Pabo, 1987) according to biological principles from which related information can be readily extracted.

Reprint requests to: Cathy Wu, Department of Computer Science, The University of Texas at Tyler, Tyler, Texas 75701.

The most notable example of a second generation database is the PIR (Protein Identification Resource) protein sequence database, which is organized with the superfamily concept (Sidman et al., 1988). A second generation database, however, is much more difficult and time-consuming to organize than a raw sequence database. The time necessary to annotate an entry and place it into the PIR database according to the superfamily is about four times that needed for preparation of a raw entry (Barker et al., 1990). This is because in order to place a new sequence entry, its degree of similarity to all other entries in the database needs to be determined by using a database search.

The neural network technique has its origins in efforts to produce a computer model of the information processing that takes place in the nervous system (Rumelhart & McClelland, 1986). One can simply view a neural network as a massively parallel computational device, composed of a large number of simple computational units (neurons). The neurons communicate through a rich set of interconnections with variable strengths (weights), in which the learned information is stored. Artificial neural networks with back-propagation (Rumelhart & McClelland, 1986) currently represent the most popular learning paradigm and have been successfully used to perform a variety of input-output mapping tasks for recognition, generalization, and classification (Dayhoff, 1990). In fact, neural networks can approximate linear and nonlinear discriminant analysis with much stronger capability of class separation (Gallinari et al., 1988; Asoh & Otsu, 1990; Webb & Lowe, 1990).

As a technique for computational analysis, neural network technology is very well suited for the analysis of molecular sequencing data. The perception learning algorithm developed by Rosenblatt in the late 1950s was adapted to sequence pattern analysis by Stormo et al. (1982) in an attempt to distinguish ribosomal binding sites from nonbinding sites. More recently, back-propagation networks have been used to predict protein secondary structure (Qian & Sejnowski, 1988; Holley & Karplus, 1989; Kneller et al., 1990) and tertiary structure (Bohr et al., 1990), to distinguish protein-encoding regions from noncoding sequences (Lapedes et al., 1990), to detect DNA-binding sites (O'Neill, 1991), and to predict bacterial promoter sequences (Demeler & Zhou, 1991). We have been applying back-propagation networks for protein classification as an approach to solve the large database search/organization problem (Wu & Whitson, 1991; Wu et al., 1990, 1991a,b). This paper describes the present state of the Protein Classification Artificial Neural System (ProCANS), which is scaled-up from a pilot protein classification system (Wu et al., 1990) and was termed Neural Network Protein DataBase (NNPDB) system previously (Wu et al., 1991a,b).

Results

Training and prediction set

The modular network architecture permits an incremental development of the ProCANS system, i.e., individual modules can be trained and optimized one at a time. The current system has four database modules trained and analyzed for seven protein functional groups, consisting of 620 superfamilies and 2,148 entries of the annotated PIR protein sequence database (Table 1). These include the electron transfer proteins and the six enzyme groups (oxidoreductases, transferases, hydrolases, lyases, isomerases, and ligases). The number of superfamilies for each network module is an optimum value of between 100 and 200. If the number is too small (e.g., 28 for the electron transfer proteins), then the network gives too many false positives (Wu & Whitson, 1991). If the number is too large (e.g., 305 for electron transfer proteins, oxidoreduc-

Table 1. *PIR protein entries used to train and test ProCANS*[a]

Database module	Protein functional group	Number of superfamilies: total (begin-end)	Number of entries: total (train + test)
EO	Electron transfer proteins	28 (1-28)	385 (266 + 119)
	Oxidoreductases	120 (29-148)	368 (291 + 77)
	Subtotal	148 (1-148)	753 (557 + 196)
TR	Transferases	157 (149-305)	499 (383 + 116)
HY	Hydrolases	178 (306-483)	584 (455 + 129)
LI	Lyases	66 (484-549)	196 (156 + 40)
	Isomerases	23 (550-572)	47 (41 + 6)
	Ligases	48 (573-620)	69 (64 + 5)
	Subtotal	137 (484-620)	312 (261 + 51)
Total		620 (1-620)	2,148 (1,656 + 492)

[a] The total numbers of superfamilies and entries for each database module are underlined.

tase, and transferases), then the network takes a long time to train (unpubl.).

Among the 2,148 entries, 1,656 were used for training, and the remaining 492 were used for prediction (Table 1). The prediction set was compiled by using every third entry from superfamilies with more than two entries. During the training phase, each database module was trained with its own training entries (i.e., 557, 383, 455, and 261 entries, respectively, for the four database modules, named EO, TR, HY, and LI). During the prediction phase, every database module was tested with all 492 prediction entries concatenated from entries of individual database modules.

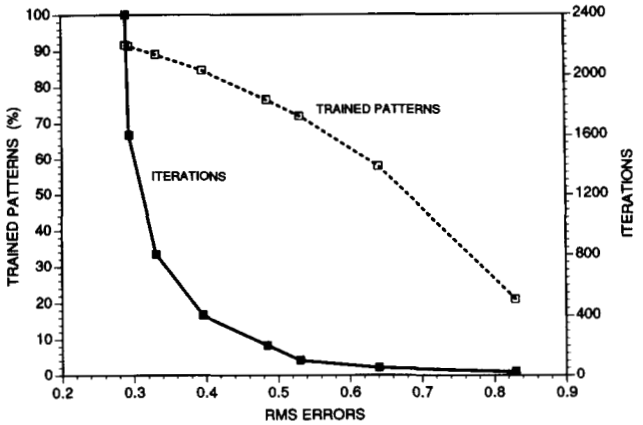

Fig. 1. The training curves of the neural network illustrated by the TR database module with the a2 encoding method.

Network parameters

Network configuration

The number of input units (size of input layer) is the number of input patterns created from the selected *n*-gram encoding method. The number of output units is the number of superfamilies of the database module, with each unit representing one superfamily. The optimum number of hidden units is between 100 and 300, which is a number close to the output size. Large hidden layer sizes resulted in poor convergence speed and predictive accuracy probably due to the overlearning of unnecessary details.

Training parameters

The learning rate (η) and the momentum term (α) are important to the network learning in speeding convergence and avoiding local minima (Weiss & Kulikowski, 1991). Their optimum values are the learning rate of between 0.2 and 0.8 with a momentum term of between 0.2 and 0.3. A high momentum term of greater than 0.6 resulted in very slow convergence. For the initial weight matrix, random weights ranging from −0.5 to 0.5 were used. The results presented below are based on the networks with 200 hidden units, learning factor of 0.8, momentum term of 0.3, and bias term of −1.0.

Stopping parameters

The training can be terminated when it is converged to within a certain tolerance or when a fixed number of training epochs (iterations) has been reached. The tolerance is a user-defined value of root mean square (RMS) error. Training curves are plotted to identify reasonable tolerance and epoch parameters (Fig. 1). The percentage of trained patterns (i.e., the percentage of known patterns that are correctly classified after training) is directly related to the RMS error. The RMS error decreased quickly from 0.8 to 0.4 with trained patterns increased from 21 to 85% during the first 400 iterations. The network converged to an RMS error of 0.33 with 89% trained patterns after 800 iterations. The degree of learning appeared to reach a plateau in the 90% range. The further training of another 800 and 1,600 iterations only decreased the RMS error slightly to 0.294 and 0.289 with 91.4 and 91.6% trained patterns. Although the convergence speed is highly dependent on the encoding methods chosen (shown below), similar training curves were observed for all network modules. Thus, the network modules were trained from only 800 iterations and then compared for their convergence speed and predictive accuracy.

Effects of encoding schema

It has been demonstrated (Wu et al., 1991a) that the encoding method is the most important factor that affects system performance. Encoding methods are designated by a two-character code: the first character is a letter designating the alphabet set; the second character is a digit representing the order of the *n*-gram. Three observations were made: (1) among the four alphabet sets, the amino acid and exchange group alphabets gave a much higher predictive accuracy (83–89%) than did the hydrophobicity alphabets (58–76%), with a2 (bigrams of amino acid) and e3 (trigrams of exchange group) being the best two encoding methods; (2) important *n*-gram patterns can be concatenated into one long input vector to improve prediction results, such as the concatenation of the a1 (monograms of amino acid), a2, e1 (monograms of exchange group), e2 (bigrams of exchange group), and e3 patterns in cat1 encoding (Wu et al., 1991a); and (3) prediction results (i.e., classification scores) from best performing encoding modules can be combined with an averaging function to improve the sensitivity and specificity of the system (i.e., more patterns identified with fewer false positives) (Wu & Whitson, 1991).

The input vectors used in previous studies (Wu & Whitson, 1991; Wu et al., 1991a) were essentially "count" vectors that represent the *n*-gram counts. However, the order information in the sequence string is not preserved. The

present study evaluates the addition of a second vector, the "position" vector that notes the positions of *n*-gram patterns on the sequence. The value of each neuron in the position vectors is the average position of each *n*-gram on the sequence string, scaled to fall between 0 and 1. With the two types of vectors, each *n*-gram pattern can be represented in three ways: count vector only, position vector only, and both vectors. Preliminary studies showed that for most encoding methods, best prediction accuracy was obtained with both vectors, followed by count vector only, then position vector only (unpubl.).

This study used 10 encoding modules, with 5 encoding methods represented by count vector only, and the same 5 methods represented by both vectors. The encoding methods chosen involve the concatenation of various amino acid and exchange group patterns. These are a2, ae12 (a1, a2, e1, and e2 concatenated), a2e2 (a2 and e2), a2e3 (a2 and e3), and ae123 (a1, a2, e1, e2 and e3). The 10 encoding modules thus are a2, ae12, a2e2, a2e3, and ae123, which use count vectors, and ba2, bae12, ba2e2, ba2e3, and bae123, which use both vectors. Tables 2 and 3 summarize the training and prediction results, respectively, for the various encoding modules after training for 800 iterations.

Table 2. *Training summaries of the four database modules after 800 iterations*

Database module	Encoding method	Network configuration: input × hidden × output	Number of connections	Cray CPU time (s)	RMS error	Trained patterns: number trained/total (%)
EO	a2	400 × 200 × 148	109,600	7,766	0.4384	450/557 (80.79)
	a2e2	436 × 200 × 148	116,800	8,325	0.4257	458/557 (82.23)
	ae12	462 × 200 × 148	122,000	8,668	0.1897	537/557 (96.41)
	a2e3	616 × 200 × 148	152,800	10,732	0.3742	479/557 (86.00)
	ae123	678 × 200 × 148	165,200	11,614	0.2472	523/557 (93.90)
	ba2	800 × 200 × 148	189,600	13,305	0.3416	492/557 (88.33)
	ba2e2	872 × 200 × 148	204,000	14,340	0.2202	530/557 (95.15)
	bae12	924 × 200 × 148	214,400	15,225	0.2243	529/557 (94.77)
	ba2e3	1,232 × 200 × 148	276,000	19,260	0.2543	521/557 (93.54)
	bae123	1,356 × 200 × 148	300,800	21,021	0.1896	537/557 (96.41)
TR	a2	400 × 200 × 157	111,400	5,546	0.3313	341/383 (89.03)
	a2e2	436 × 200 × 157	118,600	5,902	0.3429	338/383 (88.25)
	ae12	462 × 200 × 157	123,800	6,134	0.2019	368/383 (96.08)
	a2e3	616 × 200 × 157	154,600	7,571	0.3926	324/383 (84.60)
	ae123	678 × 200 × 157	167,000	8,187	0.2607	357/383 (93.21)
	ba2	800 × 200 × 157	191,400	9,491	0.2980	349/383 (91.12)
	ba2e2	872 × 200 × 157	205,800	10,000	0.2286	363/383 (94.78)
	bae12	924 × 200 × 157	216,200	10,508	0.2229	364/383 (95.04)
	ba2e3	1,232 × 200 × 157	277,800	13,404	0.2892	351/383 (91.64)
	bae123	1,356 × 200 × 157	302,600	14,624	0.2846	352/383 (91.91)
HY	a2	400 × 200 × 178	115,600	6,710	0.3722	392/455 (86.15)
	a2e2	436 × 200 × 178	122,800	7,096	0.3510	399/455 (87.69)
	ae12	462 × 200 × 178	128,000	7,372	0.2255	432/455 (94.95)
	a2e3	616 × 200 × 178	158,800	9,092	0.4546	361/455 (79.34)
	ae123	678 × 200 × 178	171,200	9,754	0.2695	404/455 (88.79)
	ba2	800 × 200 × 178	195,600	11,154	0.2966	415/455 (91.21)
	ba2e2	872 × 200 × 178	210,000	12,013	0.2653	423/455 (92.97)
	bae12	924 × 200 × 178	220,400	12,629	0.1817	440/455 (96.70)
	ba2e3	1,232 × 200 × 178	282,000	16,041	0.3146	410/455 (90.11)
	bae123	1,356 × 200 × 178	306,000	17,515	0.2045	436/455 (95.82)
LI	a2	400 × 200 × 137	107,400	3,666	0.4108	217/261 (83.14)
	a2e2	436 × 200 × 137	114,600	3,918	0.3817	223/261 (85.44)
	ae12	462 × 200 × 137	119,800	4,101	0.2770	241/261 (92.34)
	a2e3	616 × 200 × 137	150,600	5,161	0.3663	226/261 (86.59)
	ae123	678 × 200 × 137	163,000	5,429	0.2478	245/261 (93.81)
	ba2	800 × 200 × 137	187,400	6,186	0.2628	243/261 (93.10)
	ba2e2	872 × 200 × 137	201,800	6,664	0.2770	241/261 (92.34)
	bae12	924 × 200 × 137	212,200	7,009	0.2700	241/261 (92.34)
	ba2e3	1,232 × 200 × 137	273,800	9,384	0.3096	236/261 (90.42)
	bae123	1,356 × 200 × 137	298,600	9,909	0.1959	251/261 (96.17)

Table 3. *Prediction summaries of the system with four database modules after 800 iterations*[a]

Encoding method	CPU time (s)	Accuracy at 0.01			Accuracy at 0.3			Accuracy at 0.9		
		TC (%)	IC (%)	UN (%)	TC (%)	IC (%)	UN (%)	TC (%)	IC (%)	UN (%)
a2	43	80.28	15.45	3.86	74.59	1.02	24.39	62.40	0.20	37.40
a2e2	46	79.67	12.80	7.32	73.17	1.42	25.41	61.18	0.20	38.62
ae12	49	90.04	9.35	0.61	79.27	1.83	18.90	68.29	0.00	31.71
a2e3	59	79.88	14.63	5.28	73.17	0.41	26.42	62.40	0.00	37.60
ae123	63	87.40	10.16	2.44	79.88	0.81	19.31	69.31	0.00	30.69
ba2	71	83.94	8.94	6.91	75.20	0.81	23.98	61.99	0.61	37.40
ba2e2	77	86.38	7.52	6.10	78.66	0.81	20.53	67.68	0.41	31.91
bae12	81	87.60	11.59	0.81	78.05	1.02	20.93	66.87	0.61	32.52
ba2e3	106	85.57	9.15	5.28	77.64	1.22	21.14	66.26	0.41	33.33
bae123	123	87.20	9.55	3.25	78.05	0.81	21.14	67.07	0.41	32.52
Avg.		90.04	7.93	1.83	80.69	1.02	18.29	67.28	0.00	32.72
ae12[b]		84.15	15.24	0.61	78.66	2.44	18.90	68.29	0.00	31.71
ae123[b]		83.94	13.62	2.44	79.47	1.22	19.31	69.31	0.00	30.69
Avg.[b]		85.57	12.60	1.83	80.49	1.22	18.29	67.28	0.00	32.72

[a] The predictive accuracy is shown at three threshold values, 0.01, 0.3, and 0.9, and expressed with three terms: TC (the percentage of total correct patterns), IC (the percentage of incorrect patterns), and UN (the percentage of unidentified patterns). The total number of correct patterns is the total of correct first-fit, second-fit, and third-fit patterns. The average (Avg.) encoding method used the combined outputs of the ae12 and ae123 encoding methods.

[b] Only correct first-fit patterns are counted as correct patterns.

Convergence speed

The choice of encoding methods greatly affects the convergence speed of the network. After 800 iterations, the encoding modules converged to a tolerance level of between 0.18 and 0.45 RMS error, with the percentage of trained patterns ranging from 79 to 97% (Table 2). Because the RMS error is directly related to the percentage of trained patterns (Fig. 1), the latter is used as an indicator of the convergence speed. To compare the convergence speed of various encoding methods, the percentage of trained patterns of each method was calculated by adding the total number of trained patterns from all four database modules and dividing by the number of training patterns (i.e., 1,656). The resulting percentages of trained patterns for the 10 encoding methods in descending order were 95.3, 95.2, 95.1, 94.0, 92.3, 91.7, 90.5, 85.6, 84.5, and 83.9, respectively, for ae12, bae123, bae12, ba2e2, ae123, ba2e3, ba2, a2e2, a2, and a2e3. It appears that (1) encoding methods that incorporated a1 and e1 *n*-gram patterns (i.e., ae12 and ae123) converged faster, and (2) in general, methods that used both counting and position vectors converged faster than those that used only count vectors.

Predictive accuracy

The weight files generated after 800 iterations were used for prediction. The predictive accuracies were measured at threshold values ranging from 0.01 to 0.9. The threshold is a cut-off level for the classification scores, above which the superfamily identification is made. At lower threshold values, more superfamilies were identified, which resulted in a higher sensitivity (more true positives), but a lower specificity (more false positives). Figure 2 shows the effect of threshold on the sensitivity and specificity. At a low threshold of 0.01, more than 90% of the prediction patterns were correctly classified, with 9% false positives. On the other hand, at a high threshold of 0.9, although only 68% of the patterns were correctly classified, the false positive was completely eliminated. When both sensitivity and specificity are considered, the 10 encoding methods ranked from best to worst are ae12, ae123, ba2e2, bae12, bae123, ba2e3, ba2, a2, a2e2, and a2e3 (Table 3). Generally, a low degree of training (especially below 90% training) results in a low pre-

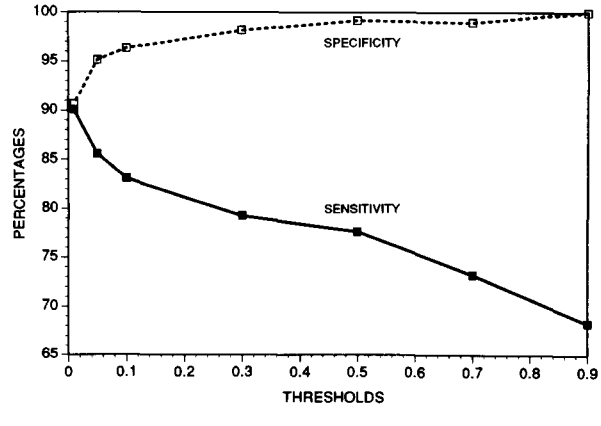

Fig. 2. The effects of threshold on the sensitivity and specificity of ProCANS classification illustrated by the ae12 encoding method. The sensitivity is the percentage of total correct patterns; the specificity is 1 – the percentage of total incorrect patterns.

dictive accuracy, as found in a2, a2e2, and a2e3. The performance of these encoding methods is improved with the addition of position vectors (i.e., ba2, ba2e2, and ba2e3) because methods that use both counting and position vectors generally converge faster. On the other hand, a high degree of training does not necessarily give better prediction. For instance, ae123, which is 92.3% trained, predicts better than bae123, which is 95.2% trained. Overall, ae12 and ae123 are considered the best encoding modules due to their high predictive accuracy, fast convergence rate, and small network size. The small network size makes them faster to train and classify, as shown below. The output values of ae12 and ae123 encoding modules can be combined (i.e., averaged) to slightly increase the sensitivity and specificity (Table 3). The predictive accuracy discussed above is based on the criteria defined in the Materials and methods section, with the number of correct patterns being the total of correct first-fit, second-fit, and third-fit patterns. The results obtained by using only first-fit as correct patterns are similar at higher thresholds of 0.3 and 0.9, but the sensitivity is lowered from 90 to 85% at a low threshold of 0.01 (Table 3).

CPU time for training and classification

The central processing unit (CPU) time for both training and classification is directly proportional to the number of neural interconnections and the number of patterns, the training time is also directly related to the number of iterations. In the present network model, all neurons are fully connected in a forward fashion (i.e., feed-forward network). Thus, the total number of neural connections of the network is the total number of connections between the input and hidden layers and the connections between the hidden and output layers. This can be expressed by:

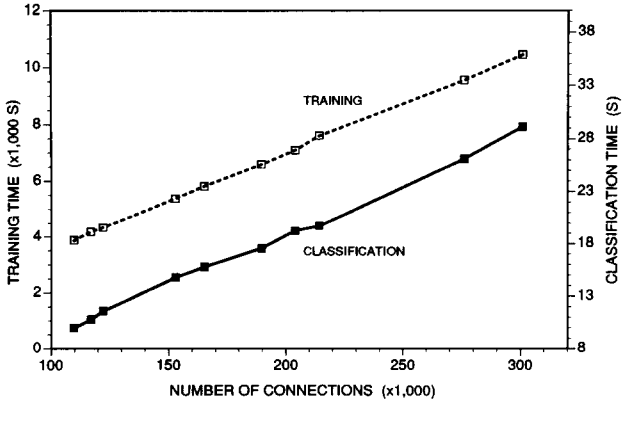

Fig. 3. The relationships between the number of neural interconnections and the training and classification time illustrated by the EO database module. The training time shown is for 557 patterns with 800 iterations; the classification time shown is for 492 patterns.

total connections = (number of input units × number of hidden units) + (number of hidden units × number of output units). Figure 3 shows the linear relationship between the number of neural connections and the CPU time for both training and classification.

The total training time of the current system, which consists of four database modules and 1,656 training patterns, is 7.3 and 9.7 Cray CPU hours, respectively, for the ae12 and ae123 encoding methods. Although it takes a long time to train a neural network, the classification is speedy because it involves only a forward-feeding through the network. The classification time of the system is the total classification time of all four database modules. It takes 49 and 63 CPU seconds, respectively, to classify all 492 prediction patterns using the ae12 and ae123 encoding methods (Table 3). This averages to approximately 0.1 CPU second per pattern.

Discussion

ProCANS is an alternative to other database search methods with two advantages. First, it is a classification system that can directly aid database organization. Secondly, the classification is fast and is not constrained by the database size. The current system is capable of classifying the superfamily of an unknown electron transfer protein or enzyme within 0.1 Cray CPU second with a more than 90% accuracy. As with other search methods, the major task for superfamily identification is to distinguish true positives from false positives. With the present system, 100% specificity (i.e., no false positives) can be achieved at a high threshold of 0.9, with a 68% sensitivity. Therefore, one can use ProCANS to screen a large number of unknown protein sequences and give true identifications to more than two-thirds of the query sequences quickly. The rapid and accurate superfamily identifications provided by ProCANS would be particularly valuable to the organization of protein sequence databases (e.g., the preliminary PIR database with 12,837 unclassified entries and the unverified PIR database with 12,354 unannotated entries) and to the gene recognition in large sequencing projects.

The classification time on a full-scale system embedded with all known superfamilies is estimated to be about four times that of the present system (well within one Cray CPU second), because the full system is about four times larger. (The current system has 620 superfamilies trained with four database modules, whereas the complete system will have 2,589 superfamilies trained by 14 database modules.) Unlike other search methods, in which search time depends strongly on database size, the classification time of ProCANS is expected to remain low even if there is a 10–100-fold increase of sequence entries. This is because (1) although the number of sequences in the database increases the number of superfamilies is likely to remain small, as most sequences will be classi-

fied into one of the known superfamilies (Doolittle, 1989), and (2) the classification time is directly proportional to the total number of neural connections, not the number of superfamilies. (As an example, given an ae12 network configuration of 462 × 200 × 100, if the number of superfamilies doubles, then the classification time would increase only 18% instead of 100%.) Training the network, however, is directly related to the number of training patterns, so the time needed to train each module will increase considerably. Therefore, we will continue to use the multiprocessor Cray for training the system.

ProCANS has two useful products: a neural database, which consists of a set of weight matrices that embed superfamily information in the neural interconnections after iterative trainings, and a system software that utilizes the neural database for rapid protein classification. The encoding schema used in preprocessing makes the neural database essentially a neural network presentation of the lookup (hash) tables built from the protein sequence database. The neural database and the system software can be ported to other computer platforms easily. We have implemented a back-propagation neural network system on an intel iPSC hypercube, termed HANS (hypercube artificial neural system) (Whitson et al., 1990). The weight files trained on the Cray have been ported to the hypercube and used to classify proteins successfully.

The predictive accuracy of the current system is about 90%, which is slightly lower than that of the FastA database search method (Pearson & Lipman, 1988). (A systematic comparison of ProCANS with other search methods including FastA and BLAST [Altschul et al., 1990] is being conducted and will be published elsewhere.) The accuracy is expected to increase with an enhanced encoding method and/or neural network design and with increased database entries for training. With the modular architecture, it is easy to incorporate other encoding schemes without affecting the present system. The accumulation of more sequence entries in the PIR database should improve accuracy because more patterns would be available for network training, and it was observed that only one single closely related training pattern is required for the unknown entry to be classified with a high classification score (unpubl.). The network should have enough capacity to hold the information embedded in the training patterns even with a 10–100-fold increase in the sequencing data. A typical ae12 network module in the present system has approximately 120,000 neural interconnections (i.e., weights). With an average of 300–500 training patterns used for each network, the present weight matrices are quite sparse. In fact, it has been suggested that a back-propagation network can average 10 patterns per weight (Weiss & Kulikowski, 1991).

The ProCANS system can be updated easily to reflect the growth of the database by retraining the neural networks. The system can also be adapted for other protein classification schemes. For example, the new proposed scheme for PIR database organization according to alignment groups, rather than superfamilies (Barker et al., 1990), can be readily implemented with the ProCANS design. The same design concept has been extended to the classification of nucleic acid sequences. A preliminary result showed a 92% accuracy for RNA classification by using a pentagram encoding for ribonucleotides (unpubl.).

Materials and methods

ProCANS design

ProCANS was designed to embed superfamily information from the PIR database and used as an associative memory to classify proteins based on the underlying information contents. The system employs two design principles (Wu et al., 1991b): (1) sequence encoding schema (used in the preprocessor) to extract information from the sequence string without knowing what kind of features in the underlying training data set are recognized by the network, and (2) modular network architecture (used in the neural network classifier) to allow the scaling-up of back-propagation networks for the processing of large and complex molecular databases.

Sequence encoding schema

The sequence encoding schema has three major components: (1) sequence interpretation, to interpret each sequence string with several different biological meanings for maximal information retrieval; (2) *n*-gram extraction, to extract patterns from sequence strings; and (3) pattern transformation, to convert the *n*-gram patterns to real-valued input vectors.

Sequence interpretation

Each amino acid residue of the sequence string can be attached with various biological meanings and represented by different alphabet sets. The alphabet sets of ProCANS include amino acids (20 letters), exchange group (6 letters), structural group (3 letters), and hydrophobicity group (2 letters). The six exchange groups are derived from the PAM (percent accepted mutations) matrix, which represents the conservative replacements found in homologous sequences through evolution (Dayhoff, 1972). The two hydrophobicity groups, hydrophobic and hydrophilic, and the three structural groups, ambivalent, external, and internal, are adopted from Karlin et al. (1989).

The n-gram extraction

In other studies that use neural networks (e.g., Qian & Sejnowski, 1988; Lapedes et al., 1990), each amino acid (or nucleotide) residue of the sequence string is represented by an indicator vector of 20 (or 4) input units. The length of the input vector, thus, is $20 \times n$ (or $4 \times n$), where n is the total number of residues in the string. The

major advantage of such representation is the preservation of the sequence of residues along the sequence string. This representation, however, is not suitable for detecting sequence similarities where long and varied-length sequences are to be compared. The key element of the sequence encoding schema presented here is a hashing function, called the *n*-gram extraction method, that was originally used by Cherkassky and Vassilas (1989) for associative database retrieval. The *n*-gram extraction method extracts various patterns of *n* consecutive residues from a sequence string and gives the number of occurrences of all possible letter pairs, triplets, etc. The total number of possible patterns from each *n*-gram extraction is m^n, where *m* is the number of different letters in an alphabet. The concept of the *n*-gram method is similar to that of the *k*-tuple method. The latter has been successfully applied to sequence analyses, such as the comparison of *k*-tuple locations for database search (Lipman & Pearson, 1985) and the statistical analysis of *k*-tuple patterns/frequencies for sequence discrimination and analysis (Karlin et al., 1989; Claverie et al., 1990). The hashing function has several advantages: (1) it is sequence length invariant; (2) it is residue insertion and deletion invariant; (3) it allows classification based on localized regions of similarity; and (4) it does not depend upon the a priori recognition of certain specific patterns.

Pattern transformation

The *n*-gram patterns are transformed into real-valued input vectors, with each unit of the vector (input neuron) representing an *n*-gram. The value of each neuron is the product of the *n*-gram count and weight, scaled to fall between 0 and 1. The count is the number of times the *n*-gram appears in the sequence string. The weight of each amino acid is the measure of its frequency of occurrence in nature.

Modular network architecture

It has been known that back-propagation networks do not scale up very well from small systems to large ones. A collection of small networks, however, has been demonstrated to be an effective alternative to large back-propagation networks (Kimoto et al., 1990; Lendaris & Harb, 1990). The network architecture used involves two levels of modularization, the database modularization and the encoding modularization (Wu et al., 1991b).

Database modularization

The training set derived from the PIR database is broken down to multiple sets according to functional groups, with each set trained by a separate network module, called the database module (Fig. 4). The protein functional groups in the PIR database include the electron transfer proteins, the six enzyme groups, enzyme inhibitors, growth factors, hormones, toxins, immunoglobulin-related proteins, heme-carrier proteins, chromosomal proteins, ribosomal proteins, fibrous proteins, contractile system proteins, lipid-associated proteins, organelle proteins, membrane proteins, bacterial proteins, bacteriophage and plasmid proteins, and viral proteins.

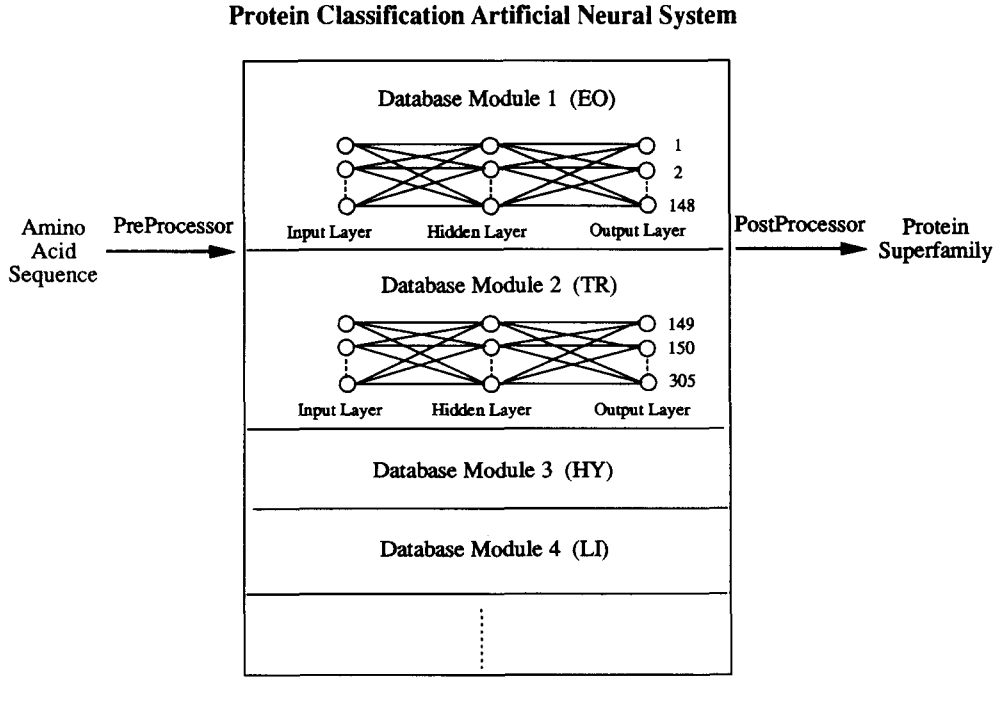

Fig. 4. The database modularization of ProCANS.

Encoding modularization

The sequence encoding performed by the preprocessor results in a large number of input neurons. The large input vector will be decomposed into multiple vectors, each corresponding to one or a few n-gram extractions and trained by a separate encoding module (Fig. 5).

ProCANS implementation

The ProCANS system software has three components: a preprocessor program to create from input sequence files the training and prediction patterns, a neural network program to classify input patterns, and a postprocessor program to summarize classification results and report superfamily identifications (Wu et al., 1991a). All three programs have been implemented on the CRAY Y-MP 8/864 supercomputer of the UTCHPC (University of Texas, Center for High Performance Computing).

The neural network program, termed CANS (Cray Artificial Neural System), is a set of programs and subroutines intended to be a tool for developing neural network applications. A CANS interface program serves as an automated neural network generator using user-defined parameters. The main driver makes calls to appropriate drivers for different learning algorithms, including back-propagation, competitive learning (Rumelhart & Zisper, 1986), Hopfield, and Kohonen (Wasserman, 1989). The user-input parameters include network configurations (number of neurons for each layer), network parameters (learning factor, momentum term, bias term, error threshold, number of iterations, and maximum runtime), and weight matrix option (initialization with random weights or normally distributed random weights, reading from a weight file, or prediction with trained weights). The back-propagation program module contains, among other routines, a back-propagation module for training, a classification module for prediction, and utility programs for weight initialization and saving.

Neural network model

The neural network used in this research is a three-layered, feed-forward network that employs a back-propagation learning algorithm. The three layers are: one input layer to provide input information, one output layer to collect output, and one hidden layer to capture information in nonlinear parameters. The functions used in the neural network model are discussed below (Rumelhart & McClelland, 1986).

Feed-forward calculation

Feed-forward calculation (change of state function) is used to determine the output of each neuron (Equations 1 and 2). The net input to each neuron is given by

$$net_i = \sum_j O_j W_{ij}, \tag{1}$$

where O_j represents the output from the neuron j in the preceding layer and W_{ij} represents the connection weight

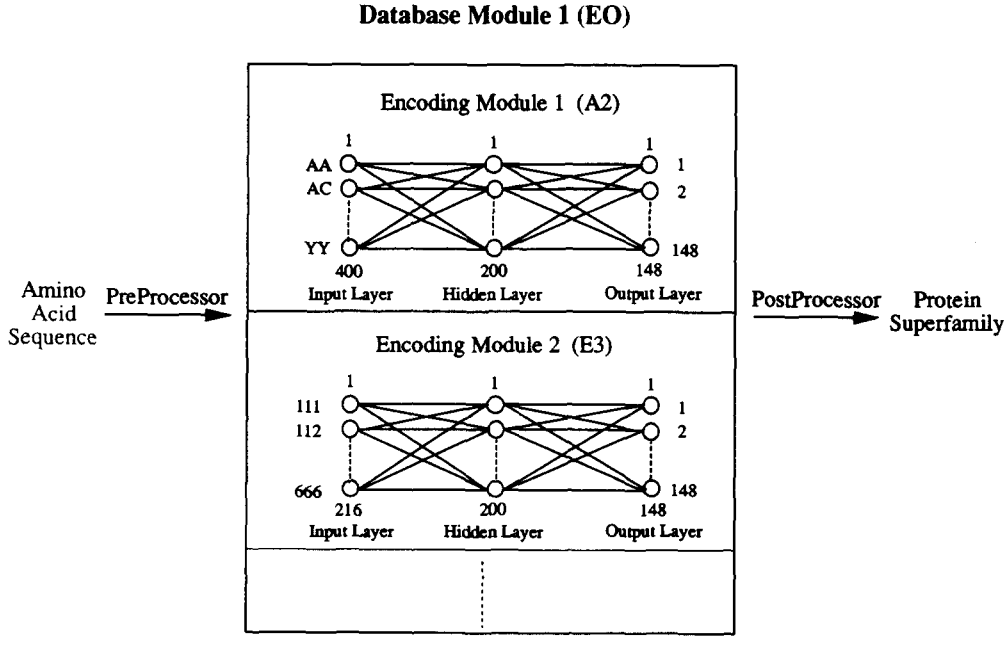

Fig. 5. The encoding modularization of ProCANS.

between neurons i and j. The output from each neuron is given by

$$O_i = f(net_i) = 1/[1 + e^{(-net_i+\theta)}], \quad (2)$$

where θ is a bias term, and f is a nonlinear activation (squashing) function.

Back-propagation learning

Back-propagation learning applies the generalized delta rule to recursively calculate the error signals and adjust the weights (Equations 3–7). The error signal at the output layer is given by

$$\delta_i = f'(net_i)(T_i - O_i), \quad (3)$$

where T_i is the target value, and $f'(net_i)$ is the first derivative of the activation (squashing) function, which can be described by

$$f'(net_i) = O_i(1 - O_i). \quad (4)$$

The error signal at the hidden layer is given by

$$\delta_j = f'(net_j) \sum_j W_{ij}\delta_i. \quad (5)$$

The error signals are then used to modify the weights with the following:

$$\Delta W_{ij(t+1)} = \eta O_j \delta_i + \alpha \Delta W_{ij(t)} \quad (6)$$

$$W_{ij} = W_{ij} + \Delta W_{ij}, \quad (7)$$

where η is the learning rate and α is the momentum term.

Learning control

The error function that back-propagation minimizes is a sum square error function, which is a function of weights; and back-propagation is really just a gradient descent applied to this function (Equation 8)

$$E = \sum_p (T_p - O_p)^2, \quad (8)$$

where p represents training patterns.

The weights are iteratively modified for every pattern being read in until the system is converged to the tolerance or until a fixed upper limit on the epochs is reached. The tolerance is a user-defined value of RMS error, which is the square root of the sum square error (E) calculated from all patterns across the entire training file.

ProCANS evaluation mechanism

The main criterion used to evaluate the performance of ProCANS is the classification accuracy for the prediction patterns. During the prediction phase, each pattern is classified on all four database modules. The values of the 620 output units represent the classification scores of the 620 superfamilies. The classification score ranges from 1.0 (perfect match) to 0.0 (no match). Superfamilies of the three highest scores are identified as the first-fit, second-fit, and third-fit. The predictive accuracy is expressed with three terms: the total number of correct patterns (true positives), the total number of incorrect patterns (false positives), and the total number of unidentified patterns (false negatives). A protein entry is considered to be accurately classified if one of its three best classifications matches the target value (the known superfamily number of the entry) with a classification score above a certain threshold. Thus, the total number of correct patterns is the total of correct first-fit, second-fit, and third-fit patterns. Three best fits are chosen because many proteins are composed of multiple domains that may be arbitrarily classified into different superfamilies, and even homologous domains may be placed in different superfamilies (Barker et al., 1990). The predictive accuracy is measured at threshold values ranging from 0.01 (low stringency) to 0.9 (high stringency) in terms of sensitivity and specificity. The sensitivity is the percentage of total correct patterns; the specificity is 1 − the percentage of total incorrect patterns.

Acknowledgments

This work was supported by the University Research and Development Grant of Cray Research, Inc.

References

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., & Lipman, D.J. (1990). Basic local alignment search tool. *J. Mol. Biol. 215*, 403–410.

Asoh, H. & Otsu, N. (1990). An approximation of nonlinear discriminant analysis by multilayer neural networks. *Proc. Int. Joint Conf. Neural Networks (June) III*, 211–216.

Barker, W.C., George, D.G., & Hunt, L.T. (1990). Protein sequence database. *Methods Enzymol. 183*, 31–49.

Bohr, H., Bohr, J., Brunak, S., Cotterill, R.M.J., Fredholm, H., Lautrup, B., & Peterson, S.B. (1990). A novel approach to prediction of the 3-dimensional structures of protein backbones by neural networks. *FEBS Lett. 261*, 43–46.

Cherkassky, V. & Vassilas, N. (1989). Performance of back propagation networks for associative database retrieval. *Proc. Int. Joint Conf. Neural Networks I*, 77–83.

Claverie, J.-M., Sauvaget, I., & Bougueleret, L. (1990). *K*-tuple frequency analysis: From intron/exon discrimination to T-cell epitope mapping. *Methods Enzymol. 183*, 237–252.

Dayhoff, J. (1990). *Neural Network Architectures, An Introduction*. Van Nostrand Reinhold, New York.

Dayhoff, M.O., Ed. (1972). *Atlas of Protein Sequence and Structure*, Vol. 5. National Biomedical Research Foundation, Washington, D.C.

Demeler, B. & Zhou, G. (1991). Neural network optimization for *E. coli* promoter prediction. *Nucleic Acids Res. 19*, 1593–1599.

Devereux, J. (1988). A rapid method for identifying sequences in large nucleotide sequence databases. Ph.D. Thesis, University of Wisconsin, Madison.

Doolittle, R.F. (1989). Redundancies in protein sequences. In *Predic-*

tion of Protein Structure and the Principles of Protein Conformation (Fasman, G.D., Ed.), pp. 599–624. Plenum Press, New York.

Doolittle, R.F. (1990). Searching through sequence databases. *Methods Enzymol. 183*, 99–110.

Gallinari, P., Thiria, S., & Soulie, F.F. (1988). Multilayer perceptrons and data analysis. *Proc. Int. Joint Conf. Neural Networks I*, 391–399.

Gribskov, M., McLachlan, A.D., & Eisenberg, D. (1987). Profile analysis: Detection of distantly related proteins. *Proc. Natl. Acad. Sci. USA 84*, 4355–4358.

Holley, L.H. & Karplus, M. (1989). Protein secondary structure prediction with a neural network. *Proc. Natl. Acad. Sci. USA 86*, 152–156.

Karlin, S., Ost, F., & Blaisdell, B.E. (1989). Patterns in DNA and amino acid sequences and their statistical significance. In *Mathematical Methods for DNA Sequences* (Waterman, M.S., Ed.), pp. 133–157. CRC Press, Inc., Boca Raton, Florida.

Kimoto, T., Asakawa, K., Yoda, M., & Takeoka, M. (1990). Stock market prediction system with modular neural networks. *Proc. Int. Joint Conf. Neural Networks (June) I*, 1–6.

Kneller, D.G., Cohen, F.E., & Langridge, R. (1990). Improvements in protein secondary structure prediction by an enhanced neural network. *J. Mol. Biol. 214*, 171–182.

Lapedes, A., Barnes, C., Burks, C., Farber, R., & Sirotkin, K. (1990). Application of neural networks and other machine learning algorithms to DNA sequence analysis. In *Computers and DNA, SFI Studies in the Sciences of Complexity*, Vol. VII (Bell, G. & Marr, T., Eds.), pp. 157–182. Addison-Wesley, Reading, Massachusetts.

Lendaris, G.G. & Harb, I.A. (1990). Improved generalization in ANNs via use of conceptual graphs: A character recognition task as an example case. *Proc. Int. Joint Conf. Neural Networks (June) I*, 551–556.

Lipman, D.J. & Pearson, W.R. (1985). Rapid and sensitive protein similarity searches. *Science 277*, 1435–1441.

Needleman, S.B. & Wunsch, C.D. (1970). A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol. 48*, 443–453.

O'Neill, M.C. (1991). Training back-propagation neural networks to define and detect DNA-binding sites. *Nucleic Acids Res. 19*, 313–318.

Pabo, C.O. (1987). New generation databases for molecular biology. *Nature 327*, 467.

Pearson, W.R. & Lipman, D.J. (1988). Improved tools for biological sequence comparisons. *Proc. Natl. Acad. Sci. USA 85*, 2444–2448.

Qian, N. & Sejnowski, T.J. (1988). Predicting the secondary structure of globular proteins using neural network models. *J. Mol. Biol. 202*, 865–884.

Rumelhart, D.E. & McClelland, J.L., Eds. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: *Foundations*. MIT Press, Cambridge, Massachusetts.

Rumelhart, D.E. & Zisper, D. (1986). Feature discovery by competitive learning. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: *Foundations* (Rumelhart, D. & McClelland, J., Eds.), pp. 151–193.38. MIT Press, Cambridge, Massachusetts.

Sidman, K.E., George, D.G., Barker, W.C., & Hunt, L.T. (1988). The protein identification resource (PIR). *Nucleic Acids Res. 16*, 1869–1871.

Stormo, G.D., Schneider, T.D., Gold, L., & Ehrenfeucht, A. (1982). Use of the 'Perceptron' algorithm to distinguish translation initiation sites in *E. coli. Nucleic Acids Res. 10*, 2997–3011.

Wasserman, P.D. (1989). *Neural Computing: Theory and Practice*. Van Nostrand Reinhold, New York.

Webb, A.R. & Lowe, D. (1990). The optimized internal representation of multilayered classifier networks performs nonlinear discriminant analysis. *Neural Networks 3*, 367–375.

Weiss, S.M. & Kulikowski, C.A. (1991). *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. Morgan Kaufmann Publishers, San Mateo, California.

Whitson, G., Wu, C., Ermongkonchai, A., & Weber, J. (1990). A back-propagation system for hypercubes. *Symp. Applied Computing*, 71–77.

Wu, C.H., Ermongkonchai, A., & Chang, T.C. (1991a). Protein classification using a neural network protein database (NNPDB) system. *Proc. Anal. Neural Net Appl. Conf.*, 29–41.

Wu, C.H., McLarty, J.W., & Whitson, G.M. (1991b). Neural networks for molecular sequence database management. *Proc ACM 19th Comp. Sci. Conf.*, 588–594.

Wu, C.H. & Whitson, G.M. (1991). Neural network database systems for genetic sequence classification. *Proc. 8th Int. Conf. Math. Comp. Model.* (in press).

Wu, C.H., Whitson, G.M., & Montllor, G.J. (1990). PROCANS: A protein classification system using a neural network. *Proc. Int. Joint Conf. Neural Networks (June) II*, 91–95.