UNIVERSITY of CALIFORNIA
SANTA CRUZ

PROTEIN STRUCTURE AND FUNCTION PREDICTION: TOOLS AND TECHNIQUES

A thesis submitted in partial satisfaction of the
requirements for the degree of

MASTERS OF SCIENCE

in

COMPUTER SCIENCE

by

Albion E. Baucom

September 2000

The dissertation of Albion E. Baucom is approved:

Professor David Haussler, Chair

Assistant Professor Lydia Gregoret

Professor Harry Noller

Dean of Graduate Studies

Contents

List of Figures

List of Tables

To my mother for always being there and believing in me, my father for instilling in me a love of science, and my organic chemistry professor, Harry Ungar, for helping me discover and explore my interest in computers.

Chapter 1

Introduction

Proteins are a fundamental component of life. Their structure, chemical features and catalytic activity play a crucial role in the processes of all living organisms. Cells use them in such things as synthesizing important compounds in biochemical pathways, as messengers to trigger metabolic events, to help regulate cell division, and to assist with otherwise energetically unfavorable chemical reactions. It is easy to see why understanding how they achieve their functional form is an interesting and important topic to study.

Proteins are fundamentally polymers composed of linear chains of a simpler building block, the amino acid. In most organisms proteins are composed of a combination of 20 different amino acids. Each amino acid has a common set of atoms and a unique side-chain. Amino acids in a protein's linear sequence, or primary structure, are linked to one another by peptide bonds. The lengths of amino acid sequences can vary from as short as 50 amino acids to as long as 2000. By virtue of the number of amino acids that can be in each position of the linear sequence of a protein, the number of combinations possible grows quickly. Most proteins have much longer sequences and therefore even larger number of possible combinations. Because the 20 common amino acids have unique chemical and structural features, such as charge, size, and possible molecular conformations, there can be many different interactions between amino acid residues in the polypeptide chain. Hydrogen bonding is a common interaction and drives the formation of structural elements such as α-helices, where amino acid residues hydrogen bond to other residues adjacent in sequence, and β-sheets which form from hydrogen bonding between adjacent peptide strands. Cysteine, an amino acid that possess a sulfur atom can also form disulfide bonds with other cysteins. These particular structural arrangements constitute what is known as a protein's *secondary* structure.

A still higher order arrangement of the secondary structure elements in three–dimensional space is known as *tertiary* structure and is what describes a protein's 'fold'. Interactions of a protein with its surrounding environment cause hydrophobic and hydrophilic effects to bring secondary structure elements together. Globular proteins contain a hydrophobic core of amino acids that are shielded from polar water molecules surrounding the structure. Conversely, most proteins are more likely to have hydrophilic amino acid side-chains on the exterior than in buried regions that can hydrogen bond with water molecules that are in contact with the surface.

Finally separate protein molecules can interact with one another to form larger multi-chain molecules. Hemoglobin is a classic example of this type of interaction. In this molecule two identical, independent, α-chains associate with two identical β-chains to form a large globular protein. This is the highest order of protein interaction and is referred to as *quaternary* structure.

It is obvious from the number of possible permutations of amino acids in a primary sequence, and the number of interactions amino acids can have amongst themselves and their environment, that there is an almost limitless number of potential three–dimensional conformations. Answering the question "What is a protein's structure given its primary amino acid sequence?" is therefore a daunting task.

1.1 Protein Structure Determination

Understanding a protein's structure can give the researcher many clues about it's role in an organism. The overall fold can immediately indicate evolutionary ties to other cellular proteins as well as other organisms. The overall shape, compactness, and intra–molecular interactions may indicate its cellular function. Specific amino acids found in solvent accessible positions, or in binding pockets, may suggest what other molecules the protein interacts with or what kinds of chemical reactions it is able to catalyze. Currently, the only way to accurately determine a protein's structure to atomic resolution is through experimental procedure. The two most common techniques for solving a proteins structure are x-ray crystallography and nuclear magnetic resonance (NMR). Unfortunately both of these methods of structure determination have limitations that prevent them from solving structures rapidly, if at all. With the help of more powerful computers, and new experimental techniques, the process of structure determination has become easier, but is still far behind the speed at which new proteins are discovered.

X-ray crystallography relies on the ability of the researcher to grow a crystal of high enough purity and three-dimensional regularity to allow valuable information to be gathered about its structure. In this technique photons in the x-ray energy spectrum are directed at a crystal which are in turn influenced the by the presence of electrons in the structure. The x-rays are diffracted, and a pattern of their scattering is captured. Using sophisticated physical and mathematical analysis the density of electrons can be inferred. A model of the structure can then be fit to the electron density to help determine the three-dimensional arrangement of individual atoms. If protein crystals can be obtained, solving a structure can take several months. If the protein's structure has been previously solved and a mutant structure is being examined, or a homologous proteins structure has already been determined, this process can be shortened signifigantly. If not, it can takes many months or years to obtain a high resolution structure that accurately positions each atom in the model, particularly if the quality of the crystals is low. Unfortunately some proteins will not crystallize at all which precludes gathering any information about a particular protein using this technique. This is the case with most membrane bound proteins that only assume their native structure when portions of their poly–peptide chain are embedded in a layer of hydrophobic lipids. Co-crystallizing a protein with lipids, or a suitable alternative environment, has proven very difficult and the structure of very few of these types of proteins have been determined.

NMR is another common technique that does not require the crystallization of the protein of interest. By using a strong magnet and altering it's field in a regular manner, the chemical shift of protons in a molecule can be observed. Hydrogens bonded to different combinations of atoms have different chemical shift signatures and allow the researcher to deduce positions of particular protons and create a list of physical constraints in the model. NMR has limits in that the protein must be relatively soluble in an aqueous environment at high concentration. Enough of theses physical constraints must be observed to accurately place portions of the molecule and give the researcher a clear idea of the three–dimensional structure.

Clearly having fast, reliable, alternative techniques to those discussed would greatly speed the research process. With the recent explosion of new sequence data coming from the human genome project, a quicker means of accurately predicting the three dimensional structure of a given amino acid sequence is more valuable than ever. The structural information of many proteins, when compiled together, can give a much larger view of their role in a cellular environment. Ultimately this information can be used to guide research in the medical field to combat a whole range of disease including cancer, Alzehimers, and the Human Immnodeficiency Virus.

1.2 Protein Structure Prediction

With the advent of computers, the ability to analyze large amounts of data became possible. This, in turn, became an immediate means of processing information discovered about protein structure. Using the Protein Data Bank (PDB) [31], a repository of three-dimensional protein structure information, it is possible to perform statistical and structural studies of such things as amino acid composition, protein conformation and function. Out of this, many new protein structure prediction techniques have emerged. The way in which in the information is utilized to make the prediction is diverse, as is the accuracy. Several techniques stand out as promising new ways to estimate what shape an amino acid sequence will assume when it folds.

The most ambitious, and often most difficult technique is *ab initio* prediction. This type of prediction uses only limited statistical information gleaned from the PDB about preferred conformational states and residue-residue contact potentials, but relies more heavily on what physical chemists know about the interactions between atoms. A common approach in this field is to allow an protein to sample many different conformations while applying an energy function to evaluate the plausibility of each particular fold. Many different conformations are observed in hopes of finding an energy minimum. The two biggest difficulties with this particular approach are finding a suitable energy function which accurately describes atomic interactions, and the ability to effectively sample the myriad of conformations a protein can assume. In most cases energy functions are simplifications and cannot account for complex interactions with the environment such as solvent molecules. In addition, trying all the conformations of even small proteins is impossible because of the shear number of permutations of amino acid orientations.

Another approach which limits the fold 'space' is *fold recognition*, sometimes referred to as *threading*. This technique superimposes, or 'threads', an amino acid sequence onto a series of proteins of known structure in a fold library. Each superposition of sequence and structure is then evaluated with a potential energy function. The energy is used to score how well that sequence can assume that shape given biological constraints such as bond distances and angles. Typically this is done only with the peptide backbone since orienting each amino acid side chain is a very difficult task. This particular method has become more valuable as more protein structures are solved and the size of the fold library is increased. The two biggest drawbacks to this method are 1) a sequence may assume a fold that has never been seen before and therefore may not find an appropriate match in the fold library and 2) the quality of the alignment should the correspondence in structure be accidentally positioned out of phase. The first problem may only be overcome by increasing the number of solved structures of unique fold through experimental techniques and is the most serious limitation of this technique. The second problem may be overcome by close analysis of the alignment and cross-validation of structural elements from supporting biochemical evidence. Still this technique does not give high-resolution solutions to the folding problem but rather provides low-resolution models that can help direct further experimental discovery. A third class of protein prediction relies on statistical analysis of protein structures solved by NMR and x-ray crystallography and deposited in the PDB. By quantifying proteins in the context of their fold and function, the construction of statistical data sets is possible. These in turn can be applied to the prediction of a novel amino acid sequence's fold or function comparing it to features previously observed in the statistical analysis. Neural networks are one such applications of this technique. Neural networks are provided a non-biased data set of protein features extracted from the PDB and allowed to 'learn' relationships between the data. A target output is given to the network and it is allowed to modify internal weights to achieve that target given the input. This is done repeatedly until no further adjustments to the weights improve the performance. The neural network is then evaluated on it's ability to correctly identify, or classify, a protein sequence it has never seen before. Using an iterative approach, the input data and features selected can be refined to give the best predictive results. Another method is the Hidden Markov Model (HMM). HMM's can classify sequences by looking at the composition of amino acids as an alphabet and determining what, for instance, is the probability of it belonging to a particular family of proteins. Training an HMM entails assigning probabilities to each position in the sequence. The probabilities specify the likelihood that an amino acid will transition from one type to the next in the sequence. Generally, the training data are multiple alignments of protein sequences with at least one of known structure. A sequence can then be evaluated with the HMM to determine its likelihood of belonging to that protein family and ultimately its fold and function.

With both of these techniques, as with any statistical analyses, it is important to have as large a data set as possible. If the input data set is not sufficiently large it may not contain all combinations of features and lack the ability to generalize on never before seen examples. Neural networks are particularly sensitive to the particular features it is presented with. If information provided to the network is uninformative it is, in essence, noise. The network may then look for and assign patterns in the noise which weakens it's predictive abilities. HMM's on the other hand require many sequences of related proteins in order to accurately assign transition state probabilities. Proteins with few homologs, and thus few known protein family members, limits the ability of an HMM to arrive at meaningful transition state likelihoods.

It is clear that solving the protein folding problem is not a trivial task. Each method described attempts to solve it using a different approach in hopes of finding a model system or scheme that can accurately describe a protein's structure. This is most commonly done using what is known about atomic interactions, related proteins of known structure, or statistical analyses of general protein features.

1.3 Summary of Chapters

The chapters that follow in this thesis approach the protein folding problem, as well as protein function prediction, using variations of several of the aforementioned techniques. Chapter 2, "Predicting Protein Function from Structure Using Neural Networks" describes the application of pattern recognition using neural networks in the area of protein function prediction. Using a non–biased data set of 36 proteases, and 878 non–proteases, a neural network was trained to accurately predict a novel protein's inclusion in the family of digestive catalysts. 10 features describing biochemical composition and structural properties were selected as input. The neural network was evaluated on its ability to distinguish proteases from non-proteases. The resulting network was able to correctly classify 86% of proteases as true examples, and 87% of non-proteases as false examples. These results strongly suggest a relationship between this protein families structural features and functional abilities. It is now believed that this technique may be applied to other family's of proteins using a selection of features that best describe a unique set of properties. This work was done with Eric Stawiski, Scott Lohr and Lydia Gregoret and published in the *Proceedings of the National Academy of Sciences* [30].

Chapter 3, "DINAMO: An Interactive Protein Alignment and Model Building Tool", introduces a Java based application that allows a researcher to evaluate protein alignments based on evolutionary conservation and structural features. DINAMO provides a means with which to build three-dimensional models of proteins based on an alignment. The sequences of interest may be viewed in a linear fashion using an editor which gives the user feedback on the quality of the current alignment. If one of the sequences has a known structure, that may be displayed using molecular graphics, and the sequence of another protein may be superimposed on the structure to evaluate the fit both from a conservation point of view as well as structurally. Together, the molecular graphics portion and the sequence editor give the researcher information with which to judge the quality of the output of structural alignment programs such as DALI [4], and conservation based alignment programs such as BLAST [9]. Originally started as a class project with Marc Hansen, this work was realized by the work of Jesse Bentz with the help of Lydia Gregoret and published in both the proceedings of the *2nd Annual Pacific Symposium on Biocomputing* [19], and the journal *Bioinformatics* [26].

Chapter 4, "β-sheet Structure Prediction Using Neural Networks", presents the results of an attempt to classify and predict anti-parallel β-sheet structure. Using a non-redundant data set of 339 β-sheet containing proteins from the PDB, a neural network was initially trained to predict whether a window of 10 amino acids on two β-strands was a true β-sheet contact or not. The window of residues was selected on the basis of side chain recognition between amino acids adjacent on the peptide backbone, and opposite one another in the β-sheet. Later the network was trained to classify the same 10 amino acid window as belonging to 1 of 12 different types of β-sheet structure, including a false pairing. The experiments were conducted with the hopes of obtaining a set of rules that could be used to discriminate true and false β-sheet pairings using predicted β-strands. Features such as amino acid hydrophobicity, volume, charge, β-sheet propensity, and identity were used as input. Unfortunately the neural network was never able to achieve better than 63% accuracy in discriminating true and false examples using realistically available data as input. Furthermore the network only achieved 62% accuracy in correctly classifying a given contact as belonging to one of 12 different defined types of pairings. The work was done with Melissa Cline, Lydia Gregoret, and David Haussler and presented as posters at the *10th annual Protein Society Meeting* in August 1996, and at the *2nd Meeting on the Critical Assessment of Techniques for Protein Structure Prediction* in December of 1996.

Chapter 2

Predicting Protein Function from Structure Using Neural Networks

2.1 Introduction

A protein's biological role in an organism cannot always be fully understood through biochemical analysis, and this is one obvious reason why structures are determined by NMR or x-ray crystallography. Even after the structure of a protein is solved its mechanism of catalysis is not always obvious. In some cases even its function is not apparent. With the recent rapid growth of sequence information coming from several different genome sequencing projects, it has become more im-

[1]This work as been previously published as Stawiski E.W. Baucom A.E Lohr S.C. Gregoret L.M. Predicting protein function from structure: unique structural features of proteases. *Proceedings of the National Academy of Sciences of the United States of America* 97(8):3954–3958

portant than ever to be able to quickly determine the function of gene products. This emerging field is becoming known as 'structural genomics' and aims at solving a representative set of all protein structures. This particular endeavor could greatly profit from a novel means of assigning function to a protein based on its structural features which it shares with a known structure, while still having low sequence similarity to another protein.

Once a protein's structure is solved it may be described using many different features such as its amino acid composition, shape, mass, and solvent accessible surface area, to name a few. These properties can be assigned without a need to explicitly describe the primary structure of the protein. Thus it may be possible that proteins share common physical attributes without having much sequence similarity. It has been previously shown that proteins can perform very similar catalysis, and have very different amino acid composition, and vice versa [17]. It follows that two very different amino acid sequences could arrive at a similar function by arranging themselves in such a way as to perform a similar biological role.

There are many more properties that can be used to describe a protein once its structure has been determined. These might include the presence of exposed hydrophobic regions, the presence of ligands such as metal ions, and structural elements such as loops, α-helices and β-sheets. Individually, these may not be important enough to accurately describe a family of proteins, but together they may form a signature for a particular type of functionality. Recognition of these structural patterns could then be applied to proteins that have been solved by x-ray or NMR, and have known function. This information could in turn be used as a predictive technique to identify new members of that particular protein family whose structure has been determined, but posses novel folds or catalytic mechanism.

Here I describe a neural network that was trained on 10 features to learn the relationship between structure and function and applied as a method for protein structure classification. Using a feed-forward, error back-propagation neural network, features chosen for their ability to uniquely describe protein structure, we have devised a means of accurately distinguishing the class of proteins known as proteases from non-proteases. The ability of the network to generalize on protease structure was evaluated by selecting several different training sets of true examples and quantifying what information is needed to make accurate predictions. The relative importance of each input, its 'sensitivity' with respect to training, was also observed in order to better understand what constituted the major components of the input signal and what may have been noise. Furthermore the networks ability to identify remote structural homologs was compared with that of other current methods in order to evaluate its value as a predictive technique.

We have found that a neural network can be trained to accurately recognize the difference between a protease and non-proteases. We also observed a close correspondence between the features the network paid the most attention to during training, and an accompanying statistical analysis of structural elements in proteases. Our neural network was also found to be far better at recognizing distantly related members of a protease class than methods such as PSI-Blast [10] which are based on iterative sequence database search techniques. Our method has been applied to other classes of proteins such as DNA and RNA binding proteins, as well as kinases with encouraging preliminary results that suggest this method may be effectively applied to other families of proteins.

2.2 Proteases

As a model system in which to apply function prediction from structural features we chose the protease family of proteins. Proteases represent a diverse group of proteins that are involved in the degradation of other polypeptide chains. They are found in a variety of organisms and serve many roles such as breaking down proteins in the digestive tract of animals and plant tissue as well as assisting in blood clotting and in various stages of growth and development. Members of this class of protein have been easy to isolate in large quantities from a spectrum of different organisms. This has facilitated intensive studies of their structure and function [18]. Due to their degradative specificity protease's have also been widely utilized by biochemists to characterize other proteins and therefore their enzymatic action is well understood. Many high resolution structures of proteolytic enzymes have been solved by both x-ray crystallography and NMR. Because of the wealth of biochemical and structural information, and the broad spectrum of members represented, proteases were an ideal class of protein to choose to characterize structurally. The set of known structures also enjoys a variety of distinct structural and mechanistic subclasses that further divide the members into families. There exists aspartic proteases that have well studied reaction mechanism that uses the amino acid aspartate during catalysis, metalloproteases that use a metal ions to promote peptide bond hydrolysis, and carboxypeptidases that specifically remove amino acids residues from the carboxy-terminal end of polypeptides, to name several. Our study included the following families of proteases, in addition to the 3 mentioned above: aminopeptidases, aspartic proteases, cysteine proteases, metzincins, subtilisin-like serine proteases, and trypsin like serine proteases.

A previous study conducted by one of us previously has also found that in a set of 72 representative protein structures [16], proteases have several unique structure features that distinguish them from other proteins [14]. These features include a higher number of α-carbon contacts and lower surface to molecular weight ratios as compared to non-protease. Together these findings suggest that proteases have unusual amino acid packing properties that could be used to distinguishing them from other types of proteins.

2.3 Neural Network Fundamentals

Neural networks are an application of statistical pattern recognition. Pattern recognition attempts to find order in data that seems otherwise disorderly. By viewing information statistically, artificial neural networks provide a means with which to find this order amongst noise. Most importantly, where a distinct signal in the data can be observed, neural network can learn whether data presented to it belongs to that which it has already observed. In this manner a neural network can generate a function that has the ability to classify information. This can in turn be applied as a predictive method able of making educated guesses at the origin of novel data.

Originally based on real living neurological systems, neural networks can be trained to recognize complex relationships in data presented to them. It has been shown that neural networks are particularly powerful at finding patterns in the presence of uninformative inputs, more commonly referred to as noise, and can even improve their ability to generalize using it [13].

Artificial neural networks are collections of coefficients and functions that, in their biological counterpart, correspond to synapses and neuron cell bodies. Synapses possess an adjustable value that represents their strength. Each individual cell body in a neural network has a collection of individual synapses associated with it that modify the value of the inputs. Cell bodies do the work of processing inputs and, depending on the strength of the signal, generating a corresponding output signal.

In this simplified model, learning is accomplished by repeatedly presenting the network with example input data and a desired outcome, or target value, in what is called a training step. Following the presentation of an example, the network is allowed to adjust the strength of its synapses, weights in an artificial neural network, in an attempt to minimize the error between the desired target value and it's output. Finally there is a testing step where the neural network is evaluated on its ability to generalize by monitoring its output when presented with examples never seen before.

This iterative process is continued until the network is unable to make adjustments that improve its ability to map input values to target outputs. What remains is a set of internal weights that are optimized to achieve the target output given an example from its training set while remaining able to generalize on examples never seen before. The network may then be applied to inputs other than those in the training or test set and allowed to generate an output.

The basic idea of an artificial neural network was introduced by Waren McCulloch and Walter Pitts in the 1940's using simplified models of true neurological systems [48]. Soon after, in the 1950's and 60's, the learning capabilities of neural networks were more extensively studied, first by Frank Rosenblatt using a system known as the *perceptron* [12], and later by Bernard Widrow and Marcian Hoff using what they labeled the Adaptive Linear Neuron or ADALINE [35]. Unfortunately it became clear that the systems implemented by these neural network pioneers were unable to learn anything more than linearly separable classes of problems. It was not until David Rumelhart, Geoffery Hinton, and Ronald Williams discovered back-propagation in 1986 [22], a means of systematically changing adaptive weights in complex network topologies, that neural networks were able to efficiently, and more importantly effectively, learn non-linear relationships in data presented to them. The ability to update the weights in response to error created many different neural network architectures that could be trained on complex input data and ushered in many new possibilites for pattern recognition.

2.3.1 Neural Network Architecture

Network Topologies

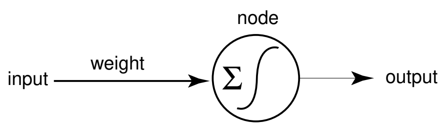


Figure 2.1: A diagram of single layer neural network

Weights are the simplest component of a neural network. They connect individual nodes together to form a network. A node has one weight per input. An adjustable value is attached to each weight and represents the strength of that connection. The value associated with a weight may be positive, or negative, depending on whether that connection should impose an excitory or inhibitory response. An example of a simple, single layer, neural network is shown in figure 2.1

A node in a neural network is a simple calculator that emits an output based on the totality of it's weighted inputs. A node first computes the sum of the product of its weights and inputs. This sum is then presented to an activation function, that

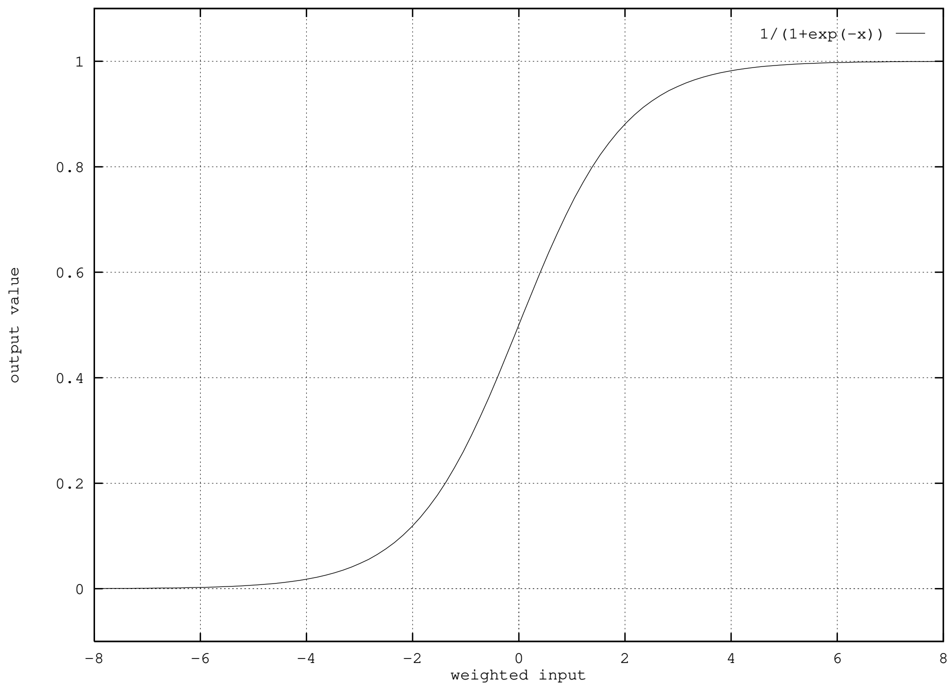


Figure 2.2: An example of a logistic activation function

determines that node's output. The activation function acts as a simple threshold; only if the incoming signal is larger than the a certain value does the node "fire". If a step function were used as an activation function, then the node would emit a 1 if the input was greater than 0, and 0 otherwise. A more common activation function is a sigmoidal logistic function as shown in figure 2.2.

Instead of a boolean output, this function can emit an output scaled anywhere between 0 and 1. This function creates more flexibility in learning since different 'shades' of output are possible. This type of output can be more informative when interpreting a neural network's response to particular input. When a network uses sigmoidal activation functions, the target values are often scaled to 0.1 for false, and 0.9 for true. Since the activation reaches 0 or 1 asymptotically as the input value approaches $-\infty$ or $+\infty$, respectively, scaling the target values can shorten the training times by preventing the neural network from attempting to train until it reaches these extremes. Correspondingly the decision of whether an example is true or false is generally made by saying all outputs greater than or equal to 0.5 are true, and all values less than 0.5 are false.

A bias input, a special kind of input that always receives an input of 1, may also be incorporated into a neural network. Since the bias constantly receives a stimulus, the value of the weight attached to this input is always included in the sum of the inputs. This provides nodes connected to the bias with a weight that can always be adjusted to help achieve a desired output.

Training is performed by presenting a *vector*, a set of inputs based on the features of a particular example, to the network along with a target output value. An example of an input vector could be could be percentages of amino acids in β-sheets and α-helices in a protein. The input is propagated through the network by multiplying the inputs by weights associated with each node. An activation function is then used to normalize the stimulus/weight product into a desired output range. An error based on the difference between the output value and target values is computed and the weights are changed according to each particular node's contribution to the output. This is generally repeated many times with both true and false examples. Each cycle of training that includes presentation of all true and false training and test examples is known as an *epoch*. In this way the examples presented to the network are mapped to their target values by minimizing error between the input and output. Ultimately the network is allowed to settle on a set of weights that produce a desired output when it encounters an example of a particular class.

Neural networks can have a single output, or many outputs depending on their application. A set of nodes that all receive the same input is called a *layer*. If a single output is desired then the final layer, or output layer, of the network is a single node, otherwise the output layer may have several nodes that will generate individual output values. A common application of a multiple output architecture is the competitive neural network. This network paradigm is can be used as a "winner-takes-all" approach where the decision of class membership is made from the node with the largest output. Only that node is allowed to update its weights thereby encouraging a network to assign it with a particular class membership decision given a training example. A transformation known as a *softmax* may be used so the outputs of all nodes sum to 1. This provides more information on the certainty with which a particular output node makes it's decision on class membership.

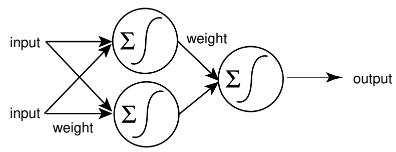


Figure 2.3: A diagram of a two layer neural network.

A neural network can also have more than a single layer of nodes. By connecting the output of a layer of nodes to the inputs of another layer one can create a multi-layer network as shown in figure 2.3.

Learning Algorithms

An artificial neural network 'learns' by making appropriate adjustments to it's weight structure. The change made to weights is based on the difference, called the error, between the networks output when presented with a training example and the target value associated with that example. The most common method of changing the weights in response to the error is known as *error back-propagation*. The back-propagation algorithm was introduced as early as 1974 by Paul Werbos in his Ph.D. thesis at Harvard University [37], and later popularized by David Rumelhart, Geoffery Hinton, and Ronald Williams in 1986 [22]. The development of error back-propagation represented a large step in neural network theory since it was a computationally efficient and easily implemented means of training a neural network how to map it's inputs to particular outputs.

In error back-propagation the values of the weights are adjusted by passing the error from the output backwards through the network from one layer to the previous. The change to a weight is proportional to the amount the node, to which the weight is connected, contributed to the overall error of the network. This backwards trace concludes at the weights of the nodes directly connected to the inputs. Typically, the change in weights is calculated by taking the partial derivative of the error with respect to a nodes output. This specifically defines the amount it contributed to the difference between the output and target values. The outcome of learning is a mapping between inputs and outputs. In the case of the logistic activation

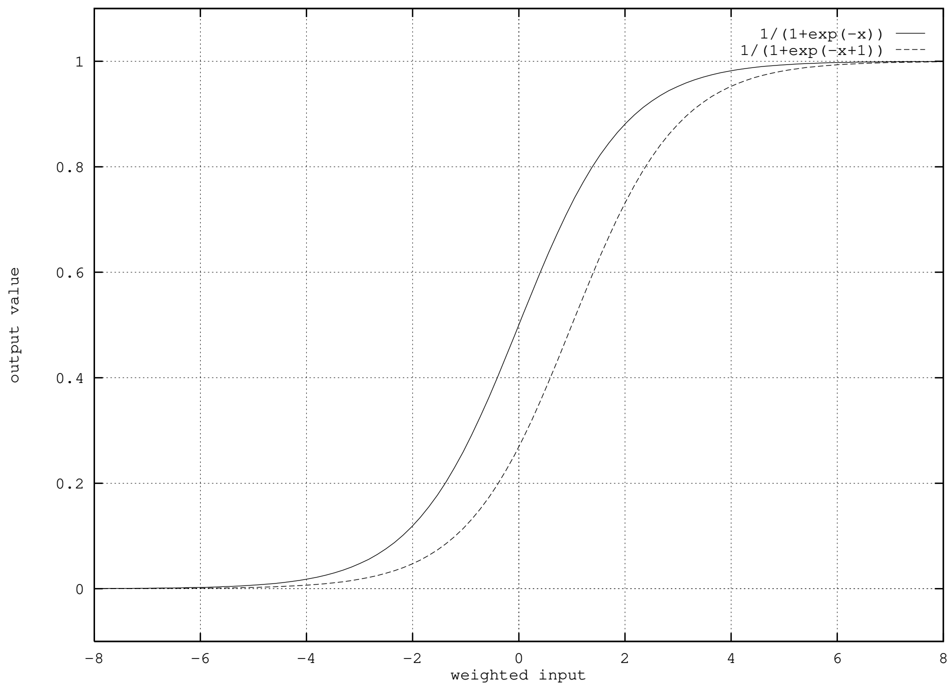


Figure 2.4: An example of a logistic activation function with and without a bias term

function the sum of the products of weights and their inputs determines the slope of the activation function, and the bias controls the offset from the origin, as shown in figure 2.4. By controlling these two parameters, the activation function can be configured to achieve the correct response given a particular input. In this sense, the function is ‘taught’ how to respond to inputs.

Single layer neural networks have limited learning capabilities. It can easily be shown that a network having a single layer of nodes between the inputs and outputs cannot learn any arbitrary boolean function [42]. For instance there exists no set of weights in a single layer net that can satisfy the problem of separating trues and false classes of the XOR function shown in figure 2.5. This can be easily demonstrated by trying to draw a straight line that divides trues from falses. Multi-layer networks are capable of learning higher-order information relationships than a single layer network because they can approximate non-linear functions.

x	y	XOR
0	0	false
0	1	true
1	0	true
1	1	false

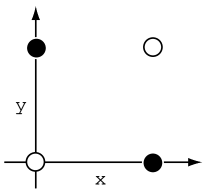


Figure 2.5: The boolean function XOR demonstrating how trues (white dots) cannont be separated from the falses (black dots) using a single line.

Layers in a multi-layer network not directly connected to the input data are able to create internal representations of the data that correspond to more complex dependencies in the information. In fact it has be shown that just a 2-layer network is capable of approximating almost any function [29] [28].

2.3.2 Example Training

To demonstrate how a neural network would be trained to approximate a function we can observe an actual training. We will use a simple *majority function* as our input. The majority function is represented as a string of bits, values that can be either 0 or 1, where the function is true whenever more than half of the values in the string are set to 1, and false otherwise. For this particular demonstration we can select the number of bits in this function to be 9 with the last bit randomly set to 1 or 0 to observe training in the presence of an uninformative input, or noise. The target value of an example will be 0.9 if it is true, and 0.1 otherwise.

This information is presented to the neural network as input vectors. An input vector represents one combination of the bits such as (0.9 1 1 1 1 1 0 0 0 1). The first number in this format is the target value, and the remaining fields are the bits of the input. In this case 6 out of the 8 informative inputs are set to 1, a majority, so the function evaluates to true and the target value is true (in this example training the 9th input is not considered in the majority since it is noise and does not contribute to the decision of whether a vector is a true or false example). It is common practice to place 2/3 of examples in the training set, and the remaining examples in the test set. For a 9 bit majority there are 9^2, or 81 examples so 54 examples are randomly chosen and placed in the training set and the remaining 27 in the test set. The division of the examples is performed randomly to prevent biasing the input.

For this example, a single layer network with a bias term was employed. The weights in the neural network were initialized with small random values. The network was then allowed to see all of the training examples. After each example the output of the network was compared to the target value, and the weights updated accordingly. Following examination of all the training examples the network was tested on the examples in the test set to see how it performs on data it had not been allowed to train on. This continued until either the test error stops improving, or the user decides the error was sufficiently minimal.

Figure 2.6 shows the training and test error following 260 epochs of training. The final values of the weights is also shown. It can be seen that by adding together a majority of the inputs modified by the values of the corresponding weights you overcome the value set by the bias term. Thus in this case it takes 5 or more inputs

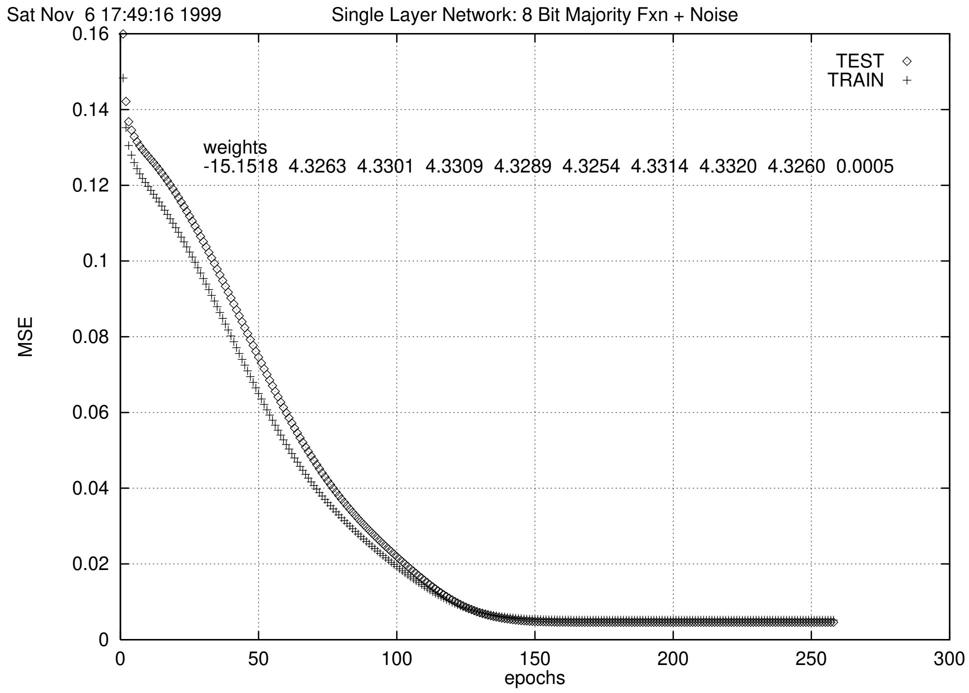


Figure 2.6: The training and test mean squared error (MSE) and final weight values of a single layer network when presented with data from a majority function

set to 1 before the incoming signal is positive and the network will emit a value close to 1 corresponding to a true example. Notice that the weight associated with the "noisy" bit is small relative to the other weights. The neural network was able to recognize this input as uninformative in obtaining the desired output and therefore diminished it's influence on any outcome.

The input sensitivity, though in this particular example it is obvious from the magnitude of the weights, can also be evaluated after each epoch over the course of training. Figure 2.7 presents the sensitivity of output with respect to each input. It is clear from this plot that the sensitivity of the output with respect to an input is low in the beginning of training while the network is beginning to learn, maximal

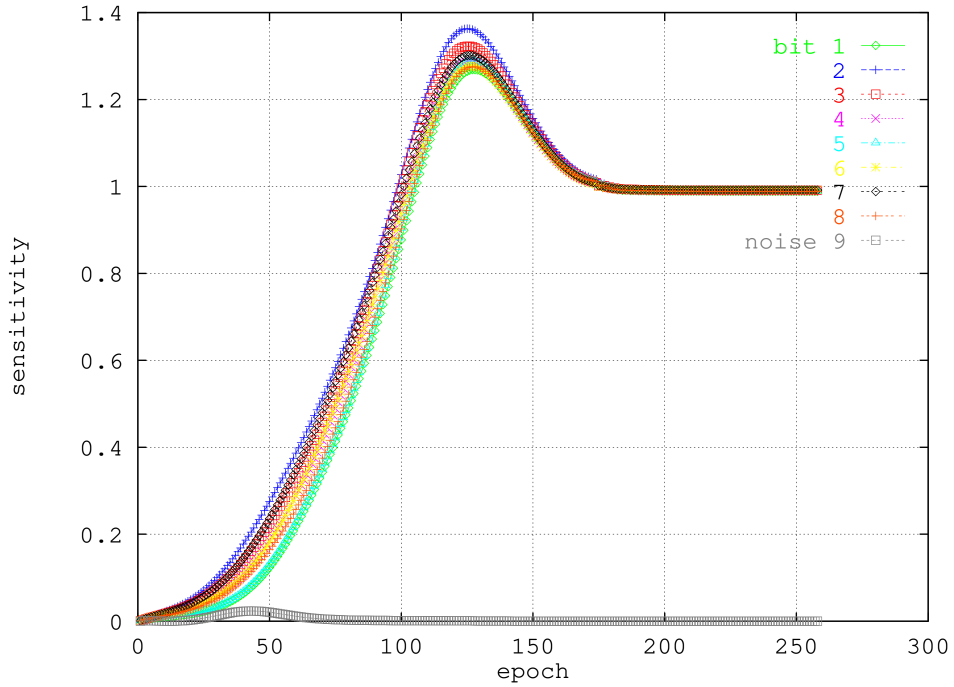


Figure 2.7: The sensitivity of each input of single layer network when presented with data from a majority function. Note the low sensitivity of the last input which is noise.

when it begins to reach an error minimum, and levels off when there are no more significant changes made to the weights and active learning has ceased.

For multiple layer networks, the magnitude of the weights are harder to interpret as a measure of the sensitivity of a particular input, since architectures having hidden nodes can create internal representations of the data. In this case weights may assume values that do not directly correspond to the inputs influence on the output, but rather contribute to perturbations in the output through connections made to other hidden nodes or output nodes. Thus, input sensitivity is a more valuable metric when used in the context of networks containing more than one layer.

It must be noted that this example is idealized. It is seldom that input data is as "clean" as that presented from this majority function. Obtaining data that is meaningful is almost always the most difficult part of neural network training and ultimately has the greatest influence on the networks ability to learn.

2.4 Feature Extraction and Input Preprocessing

An important aspect of neural network training is the selection of input features. This step is commonly referred to as feature-extraction and involves selecting relevant information to present to the net. A neural network can only tolerate so much uninformative data in the input before it loses it's ability to generalize and attempts to find patterns in the noise. Thus it is essential that characteristics of the data to be learned be rationally selected as most likely to contain a recognizable pattern. Feature extraction is usually the most difficult part of neural network prediction since the selection of meaningful inputs is seldom obvious.

A pre-processing step is also necessary to normalize the data and eliminate unintentional biases between different types of features. For instance, if features such as solvent accessibility, and amino acid side chain volume were presented together to a neural network, the fact that one is measured in square angstoms, and one in cubic angstroms must be taken into account. If this were neglected the neural network would treat these as if they were equal metrics and compare the importance of their values directly during training.

2.5 Applications of Neural Networks

Neural networks are used in a variety of biological applications ranging from gene finding, to protein secondary structure and solvent accessibility prediction. A popular suite of programs based on neural networks is located at the PredictProtein Server created by Burkhart Rost, Volker Eyrich, and Christian Sander [41]. Included are neural networks for predicting protein solvent accessibility, secondary structure, and trans-membrane spanning regions using an amino acid sequence as input. The secondary structure predictor has greater than 70% overall accuracy.

2.6 Methods

2.6.1 Data Set Construction

The dataset of non-protease examples used for neural network training and testing was selected from the Protein Data Bank [31] starting from Hobohm's and Sander's "PDB Select" list of proteins with a cut-off of 25% sequence identity [3]. The resolution of all structures in this list were better than 2.5 Å to ensure structures of sufficient quality. All proteins in the dataset were also required to be biologically active in a monomeric form. The set of protease examples met the same criteria as the false set, except that the sequence identity cutoff was set at 35% to include more examples necessary for a sufficiently large true training set. Initially there were 36 structures of proteases, and 154 structures of non-proteases. In all of our

153l	16pk	1a0p	1a17	1a26	1a34	1a6g	1a8e	1a9s	1ad6
1ads	1ah7	1ak0	1ak1	1ako	1akz	1alu	1am7	1amm	1amx
1anf	1aoh	1aol	1aqb	1arv	1ash	1asw	1at0	1atg	1aua
1auk	1ax8	1axn	1azo	1bc5	1bd8	1bg0	1bgc	1bgf	1bjk
1bkb	1bol	1bp1	1brt	1bv1	1bxw	1byb	1byq	1c25	1c3d
1ceo	1cex	1cfb	1cfr	1chd	1ckn	1cnv	1cpo	1csh	1csn
1dad	1dhr	1dhs	1dhy	1edg	1fdr	1fmt	1fts	1g3p	1gky
1gpl	1grj	1gso	1ha1	1hxn	1idk	1ihp	1inp	1ips	1ixh
1juk	1lba	1lcl	1lit	1lki	1maz	1mml	1mpg	1mrj	1mrp
1msk	1mup	1nar	1nif	1nkr	1np4	1npk	1ois	1opr	1oyc
1pda	1pgs	1phc	1phm	1pmi	1pne	1poc	1pot	1pta	1pty
1pud	1qnf	1qtq	1ra9	1rcf	1rec	1rhs	1rss	1rsy	1sbp
1tca	1tde	1tfr	1thv	1tib	1tml	1uae	1uxy	1v39	1vhh
1vid	1vjs	1wab	1zin	2abk	2baa	2cba	2cyp	2dri	2end
2gar	2hft	2i1b	2liv	2pia	2plc	2pth	2sns	2thi	3nll
3seb	3sil	4xis	6cel						

Table 2.1: PDB identifiers of proteins used in the non-protease data set.

final experiments the set of false examples was enlarged to 878 members. The PDB identifiers of proteins included in the false set can be found in table 2.1.

The calculation of surface area was done by the program CALC-SURFACE [32] which utilizes the Lee and Richards algorithm [15]. A probe size approximating a water molecule at a radius of 1.4 Åwas used except when calculating the surface roughness which used radii of 1.25, 1.5, 1,75 and 2.0 Å. Programs written by Eric Stawiski were used to calculate each protein's radius of gyration, contact order [8], and number of inter-residue Ca-Ca contacts. α-carbons's were considered to be in contact if they were within 5.5 Å of one another. All secondary structure assignment was performed by the Definition of Secondary Structure of Proteins (DSSP) program [5].

For all experiments, roughly 2/3 of the total false examples were placed in the training set, and the remaining 1/3 in the test set, which corresponded to 521 and 249 falses respectively. The individual examples in each of these sets were fixed to their particular sets to ensure that there was no more than 45% homology between any two false examples in the training and test set.

All inputs were normalized to a value between 0 and 1 to remove any artificial biases due to relative differences in metrics or ranges of values. The highest and lowest values were determined for each particular feature for both true and false examples. Each feature for a particular example, F, was normalized by subtracting the lowest value l, from the raw feature value f,

$$F = \frac{f - l}{|h - l|} \tag{2.1}$$

and dividing by the difference between the highest value h, and lowest values, l, thereby giving a value scaled between 0 and 1.

2.6.2 Neural Network Algorithm

The architecture chosen for protein function prediction was a feed-forward, error back- propagation, 2 layer neural network. There were 2 nodes in the first layer, and a single output node. Each node in the network used a sigmoidal logistic activation function. The weights were adjusted according to a sum-of-squares error and a gradient descent learning rule. On-line learning was utilized where adjustments to the weights to minimize the error were made after each example was presented to the network.

Forward Propagation

In our feed-forward network inputs from a training example are first forward propagated though the net by computing for each node a sum of the product of it's weights and inputs, a_j

$$a_j = \sum_{k=1}^{n} w_i z_i \tag{2.2}$$

where a_i is an input either from actual data for nodes in the first layer, or from nodes in first layer in the case of the output node. The weight w_i connects the ith input to the jth node. The sum runs over all n inputs that are connected to that node. The resulting weighted sum of inputs and weights, z_j is then presented to a sigmoidal logistic activation function of the form

$$z_j = g(a_j) = \frac{1}{1 + e^{-a_j}} \tag{2.3}$$

where e is the base of the natural logarithm.

All weights in the network were initialized to small, random numbers prior to training. This technique helps to lower the likelihood of trapping the network in a local minimum of the error surface by preventing unwanted artificial symmetry in the network topology. This also speeds training by discouraging the generation of values that approach the horizontal asymptote of the activation function.

Error Back-propagation

The learning rule for our neural network was based on gradient descent minimization of the difference between the network's output and the target value for an example using error back-propagation to adjust the weights. The error contributed by each node in the network was first calculated by finding the partial derivative of that nodes output with respect to the overall network error. The change in a weight's value to minimize the error was then calculated using the derivative of the error of the node to which the weight was connected and the input that particular weight received. The change in the weight value was attenuated by an adaptive learning rate to dampen harmonics which might slow or prevent convergence on a minimum error value. For the explanation of the learning rule the nodes to the inputs will be referred to as being in the ith layer, and the output node as jth layer.

The error E for the neural network was calculated as one half the squared difference between the target output and actual network output

$$E^n = \frac{1}{2}\sum_{i=1}^{n}(o_n - t_n)^2 \tag{2.4}$$

where o^n is the actual outcome of the network when presented with the nth example, and t^n is the target value for that example.

To calculate the contribution of a weight's error with respect to the network's output the chain rule of differentiation for functions with more than one variable was applied. Starting with the derivative of the network's error, and back-propagating it through the network recursively, the error introduced by each individual node was calculated. This error is commonly referred to as δ_j for the jth node.

Using the chain rule of partial derivatives, the derivative of the error ∂E with respect to a weight in the network, ∂w_{ij},

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} \tag{2.5}$$

is in general the product of the partial derivative of the error ∂E, with respect to the nodes output to which the weight is connected, ∂a_j, and the partial derivative of the nodes output with respect to the weight being examined, ∂w_{ij}. Starting with the output node, the partial derivative of the error ∂E, with respect to the weights connected to the output node, ∂w_{ij}, was calculated as the product of the derivative of the error ∂E, with respect to the nodes output ∂a_j, where

$$\delta_j \equiv \frac{\partial E}{\partial a_j} \tag{2.6}$$

and the derivative of the nodes output with respect to the weight being examined. For the output nodes (2.6) becomes the product of the derivative of the error function

$$E' = o_n - t_n \tag{2.7}$$

and the derivative of the activation function

$$g'(a_j) = g(a_j)(1 - g(a_j)) \tag{2.8}$$

The derivative of the output node with respect to the error (2.6) is thus calculated by substituting (2.3) into (2.8) and multiplying by (2.7)

$$\delta_j = \frac{\partial E}{\partial a_j} = z_j(1 - z_j)(o_n - t_n) \tag{2.9}$$

Finally, using (2.2), we can now find the derivative of the error with respect to the weight in question,

$$\frac{\partial a_j}{\partial w_{ij}} = z_i \tag{2.10}$$

where z_i is input from the output of the ith hidden node.

The derivative of the error ∂E of hidden node weights, with respect to a weight in the hidden layer, is analogous to that for the output node. In the case of hidden nodes the derivative of the error is again the product of the derivative of the error ∂E with respect to the nodes output, and the derivative of the nodes output with respect to the weight in question, ∂w_i. Here, though, the derivative of the error at the output node δ_j was substituted for the derivative of the error function (2.7) in (2.9) and multiplied by the weight connecting the hidden node to the output node. Thus back-propogating δ_j to the hidden nodes the derivative of the hidden layer nodes with respect to the error is

$$\delta_i = z_i(1 - z_i)w_{ij}\delta_j \tag{2.11}$$

Since the network topology used for these experiments only involved two weights in the hidden layer, one from each of the hidden layer nodes to the output node, (2.11) is specific to this implementaion. In general when there exists more than two layers and a node in the hidden layer is connected to more than one node (2.11) becomes a sum of these connections and weights between them. Thus in general the derivative of a hidden node is

$$\delta_i = z_i(1 - z_i) \sum_{j=1}^{n} w_{ij}\delta_j \tag{2.12}$$

where the sum runs over all connections and weights j that connect the ith layer to the next in the network.

The change in each weight is calculated as the negative product of the learning rate η, the error derivative of the node that it belongs to, δ_n, and the input value it received during the process of forward propagation. For the weights in the hidden layer connecting the ith to the jth layer this becomes

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta \delta_j z_i \tag{2.13}$$

For the hidden layer weights connected to the inputs directly, (2.13) becomes

$$\Delta w_i = -\eta \delta_i x_i \tag{2.14}$$

where x_i was actual training data for a particular example connected to the weight being updated.

Input Sensitivity

To quantify the importance of each input variable presented to the neural network, a technique known as input sensitivity was used. After forward propagating an input vector through the network and finding the networks output, an estimate of a particular input's contribution to the error can be determined by taking the partial derivative of the output with respect to an input. For this the Jacobian matrix was evaluated after each example in the training set. In a Jacobian matrix each row represents an output, and each column an individual input. For our purposes, the matrix had a one row for the single output node of the network and 10 columns for each of our 10 inputs. It can be found that the change in the output o_l is proportional to the sum of the product of the partial derivatives of the output with respect to each input and its change in value.

$$\Delta o_l \simeq \sum_i \frac{\partial o_l}{\partial x_i} \Delta x_i \tag{2.15}$$

In order to determine the change in the output with respect to a particular input, the Jacobian matrix J is evaluated in a fashion analogous to that of error-propagation whereby the chain rule of derivation is used to find the partial derivative of the output, o_l, with respect to an input, x_i

$$J_{li} \equiv \frac{\partial o_l}{\partial x_i} \tag{2.16}$$

where J_{li} is the partial derivative of the lth output with respect to ith input.

Starting with the output node, as the partial derivative of the output o_l, with respect to its weighted sum of inputs, a_k,

$$\frac{\partial o_l}{\partial a_k} = g'(a_k) \tag{2.17}$$

is the derivative of the activation function of the output node. Remembering that the input is propagated through two layers in the neural network, and applying the chain rule, the partial derivative of the output with respect to the weighted sum of inputs for a hidden layer node is

$$\frac{\partial o_l}{\partial a_j} = g'(a_j) \sum_k w_{jk} \frac{\partial o_k}{\partial a_k} \tag{2.18}$$

This runs over all nodes in the kth layer to which nodes in the ith layer send connections.

Finally, the partial derivative of the weighted sum of inputs with respect to the input variables is the sum of the product of weights connecting the input to hidden layer nodes w_{ij} and the partial derivative of the output with respect to each hidden nodes weights sum of inputs, a_j

$$J_{li} = \sum_i w_{ij} \frac{\partial o_l}{\partial a_j} \tag{2.19}$$

Putting it all together gives us the input sensitivity back-propagation algorithm for out two layer network with a single output

$$J_{li} = \sum_j w_{ij} g'(a_j) \sum_k w_{jk} \frac{\partial o_l}{\partial a_k} \tag{2.20}$$

starting with (2.17) for the output node.

Implementation of the Neural Network

The program AquaNet was used to do all neural network training and testing. AquaNet is an object oriented program written in ANSI C++ for a UNIX environment. The program consists of five main classes and a utility class used for parsing

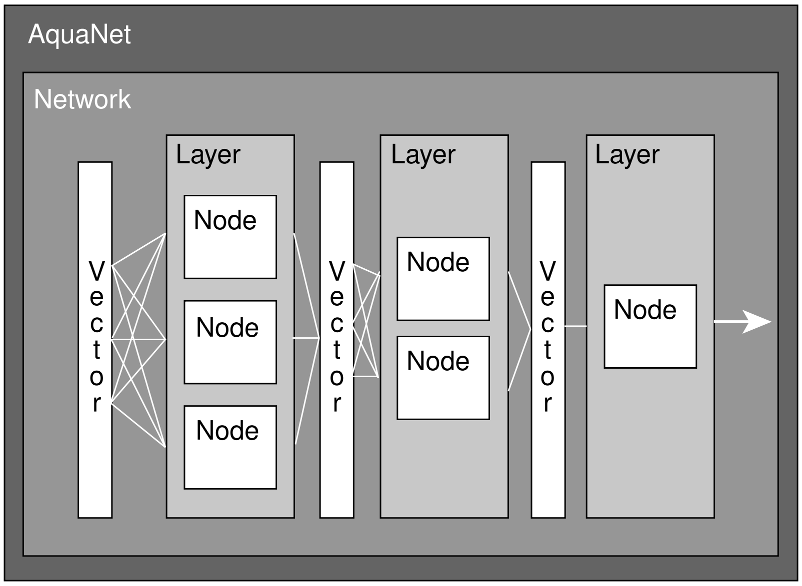


Figure 2.8: A diagram of the hierarchy of classes in the AquaNet program

input data files as shown in figure 2.8. Each class is necessary to implement a flexible program structure that can be configured many different ways. In particular, this implementation allows the user to specify, among other things, how many inputs the net will be receiving, the number of layers, how many nodes are in each layer, the maximum epochs to train for, and whether or not to enable a bias term for each node. The parameters are provided to the neural network via a simple text based script file that is easily modified by the user. An additional C program called Dep was created to read in a weight set after training and testing was completed, recreate a specific neural network architecture, and generate output when presented with input examples. This program provides an implementation of a particular network once it has been suitably trained and tested and can be used in all future applications of a particular weight set.

The simplest class in AquaNet is the Vector. A Vector stores a variable sized array of floating point values. In addition to an array, a Vector overloads all of the simple arithmetic operators to allow two Vectors that are the same size to be added, subtracted, multiplied and copied from one to another. The equality operator is also overridden to allow direct comparison of two Vectors, as is the output stream operator which generates formatted output of the stored values. A single value may also be added, subtracted or multiplied by all of the values in the Vector's array. Each value stored may be individually retrieved or changed, and the sum of all values may also be calculated. Vectors are used both as a way to manage inputs and outputs to and from AquaNet, as well as a structure to store and manipulate weights in each layer of the neural network.

Nodes in AquaNet represent the next simplest class. Nodes use Vectors to store weights, and to manage input values. A Node may have a user defined number of weights associated with it. The Nodes in a neural network only use a logistic activation of the form $\frac{1}{(1+e^{-a})}$, but could easily be modified to use another. A Node also controls whether to use a bias term or not depending on how it is initialized. A Node can report the sum of the product of its weights and inputs, and output.

A Layer in this framework manages a collection of Nodes. Data received by a Layer is distributed to all of the nodes that it contains, and the output generated by its nodes is returned. A Layer is responsible for computing the error derivative from its output during back-propagation, and updating each nodes weights in response.

The sensitivity of a particular input is also computed by each Layer based on the change in its weights. Additionally, Layers also have methods for changing the learning rate of its nodes, and storing and retrieving the values of all the weights connected to each of its nodes at particular stages during training.

To assemble a neural network, the Network class manages a series of Layers and constructs a particular neural network architecture depending on user definable parameters. A Network is responsible for presenting the first Layer with all training and test inputs during a training epoch, computing the mean squared error of the current output, as well as storing error for both the current and previous epoch. A snapshot of the networks weights at a particular epoch is also be stored by a Network in order to retrieve the weights at the point of lowest test error.

The Aquanet class represents the highest order component that manages one or more Networks. With the use of a startup script, Aquanet can read in user specified parameters, such as the number of inputs in the training and test set, the number of layers to construct the network with, the number of nodes in each layer, the training and test set file names, maximum epochs to train for, and the name of a file to save the final weights to. Using a utility class, an InputParser, Aquanet can read in the training and test data specified by the user, build the network as described in the script file, and pass the input to the Network classes to begin training. The current training and test error are retrieved from each of its Networks and displayed after each epoch by the Aquanet class.

The Dep program is an accessory C program that implements a neural network given a weight set. Weight sets generated by Aquanet include information on the architecture of the network used to arrived at their values and includes the number of inputs, the number of layers and the number of nodes in each layer. Dep can read the weight set file, and recreate a Network object capable of generating output at the state the Network was in during it's best test error using Aquanet.

2.6.3 Training Regimes

Two different training and testing experiments were conducted on the protease input data; a take-one-out experiment, and a protein class experiment. These were designed to test the neural networks ability to accurately distinguish proteases from non-proteases under different input conditions. Additionally, these experiments could provide clues as to the information content of particular examples in the training and test sets.

The take-one-out experiment involved the removal of one of each of the true examples from the training set and placing it in the test set. The neural network was then trained on 35 of the 36 examples and tested on the single true example. This gave the neural network a maximal number of true examples to train on. Under these conditions the network was tested on its ability to correctly identify a protease example that belongs to a class of proteases that the network has previously trained on.

To increase the difficulty of prediction and test the neural network's true ability, the class training regime was devised. The protease class experiment consisted of removing 1 of the 8 protease families from the training set, placing it in the test set, and training on the remaining 7 families. The neural network was then evaluated on its ability to classify proteins in the test set that had little resemblance in fold or specific catalytic mechanism to any other example in the training set.

For both of these training regimes, the network was run five times on each training and test set to determine an average training/test error. Since neural networks contain a certain amount of randomness with respect to their weight initialization, performing the experiments five times allowed us to judge whether the results of a particular run were a good indicator of the neural networks learning ability, or an artifact of a poor selection of starting weights.

2.6.4 Results Visualization

The results generated by the neural network consisted of 5 runs each of 36 different examples for the take-one-out experiments, and 5 runs for each of 8 different family of protease in the class experiments. Additionally each training run consisted of a true error, false error, and the sensitivity of each of 10 different input features. For the take-one-out experiment, this amounted to a total of 2 * 36 * (2 + 10) or 2160 data points, and 5*8*(2 + 10) or 480 data points for the class experiments.

In order to easily evaluate the results of the training examples, a Java program

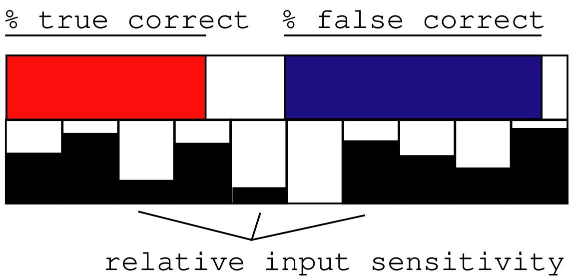


Figure 2.9: A legend for an individual panel in a NetPlot image. The input sensitivity panels are from left to right, %loop, %helix, %sheet, avg. loop length, molecular weight per residue, avg. contacts, surface area per molecular weight contact order, radius of gyration, and a Zn boolean

named NetPlot was created. NetPlot uses the principles of small multiples to make a large amount of data easily accessible to the researcher. Using a formatted text file containing the output of a series of neural network trainings, NetPlot creates a 2-D plot of individual graphs representing each individual training run as shown in figure 2.9.

Each graph contains the number of true examples classified as true, the number of false examples classified as false, and the sensitivity of each input feature. The true and false example performance are color coded for easy comparison to that of other runs. A row in this graph represents a particular training set, and the columns represent each of five different runs that were made for each training set. For the take-one-out experiment their were 36 rows each representing the network performance after each successive proteases structure that was removed from the training set and placed in the test set. For the class experiment each row represented the performance of the network when a different family of proteases was removed from the training set and placed in the test set.

2.7 Results

Using the structural characteristics of proteases as described above we trained a 2 layer neural network to discriminate proteases from non-proteases in two separate experiments. In the first take-one-out experiment where the 36 true examples where partitioned into the training and test sets 35 to 1, respectively, the neural network was able to correctly identify the true test example in 31 out of 36 cases (86%). The falses were correctly identified 87% of the time in these particular training runs. The results for all 36 true examples are shown in figure 2.10 as a NetPlot image for 5 separate trainings.

For the class experiment, 7 of the 8 different protease families were placed in the training set, and remaining class in the test set. The network was in turn trained 8 different times and scored on its ability to correctly identify all members of the family placed in the test set. The performance of the neural network under these training conditions is slightly lower than in the take-one-out experiments correctly identifying the true examples 81% of the time.

The results of each individual class are shown in table 2.2 and the corresponding training and test error as well as input sensitivities are shown in figures 2.11 and 2.12. A summary of the results are shown as a NetPlot image in figure 2.13.

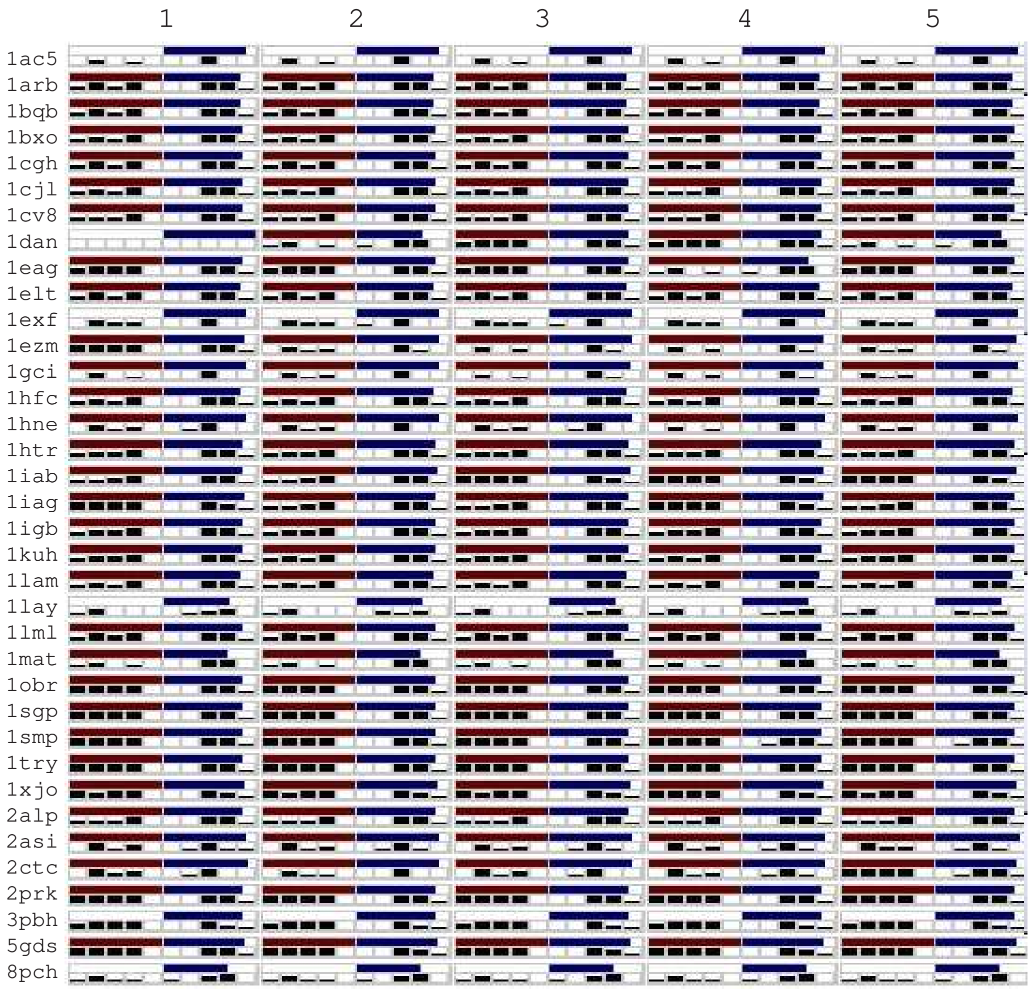


Figure 2.10: A NetPlot image showing the results of all 36 protease examples when trained 5 different times. The PDB identifier to the left of each row is the protease that was removed from the training data for that run and placed in the test set

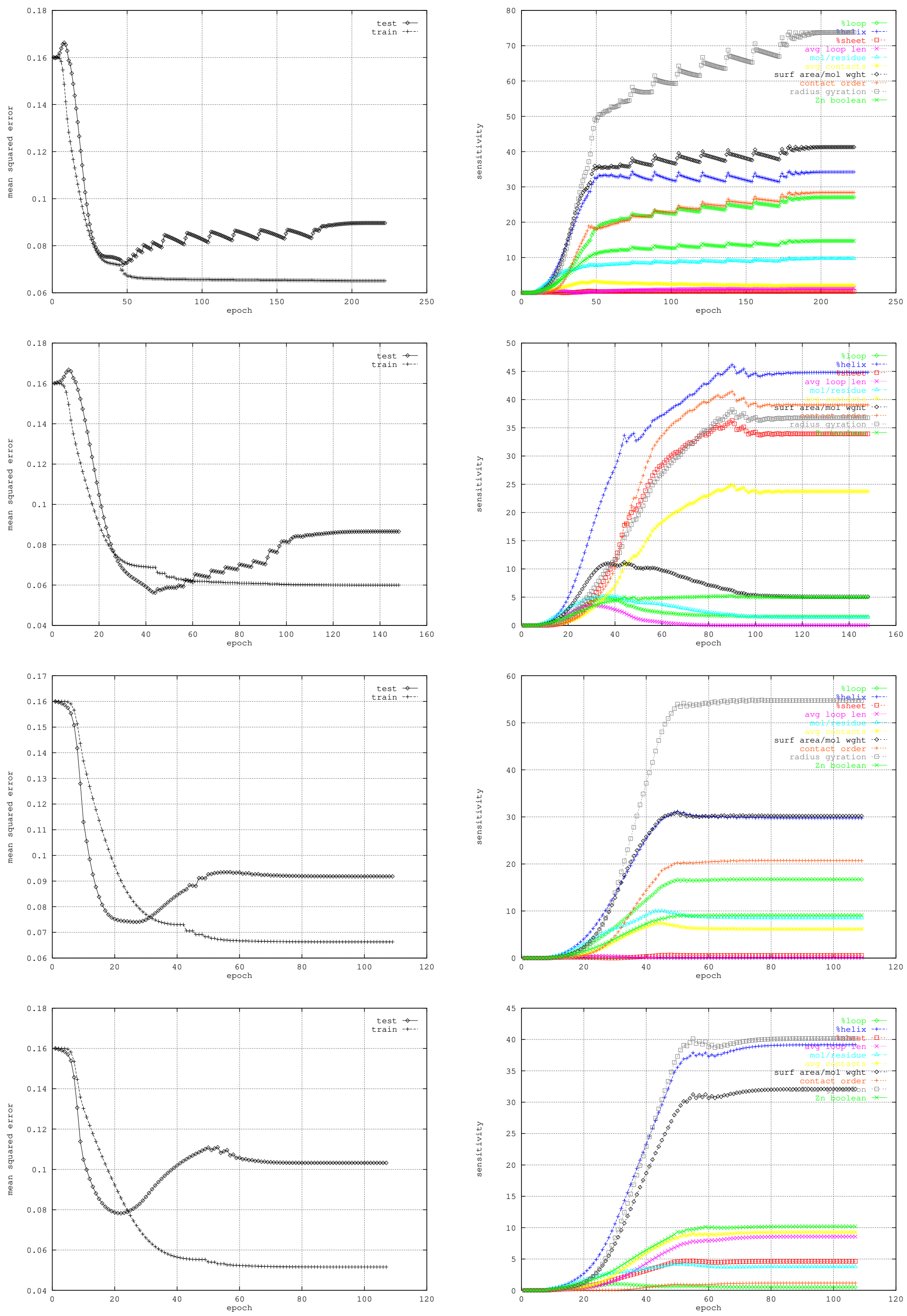


Figure 2.11: Training and test results and input sensitivity for the class experiments. The classes are, from top to bottom, aminopeptidases, aspartic proteases, carboxypeptidases, cysteine proteases

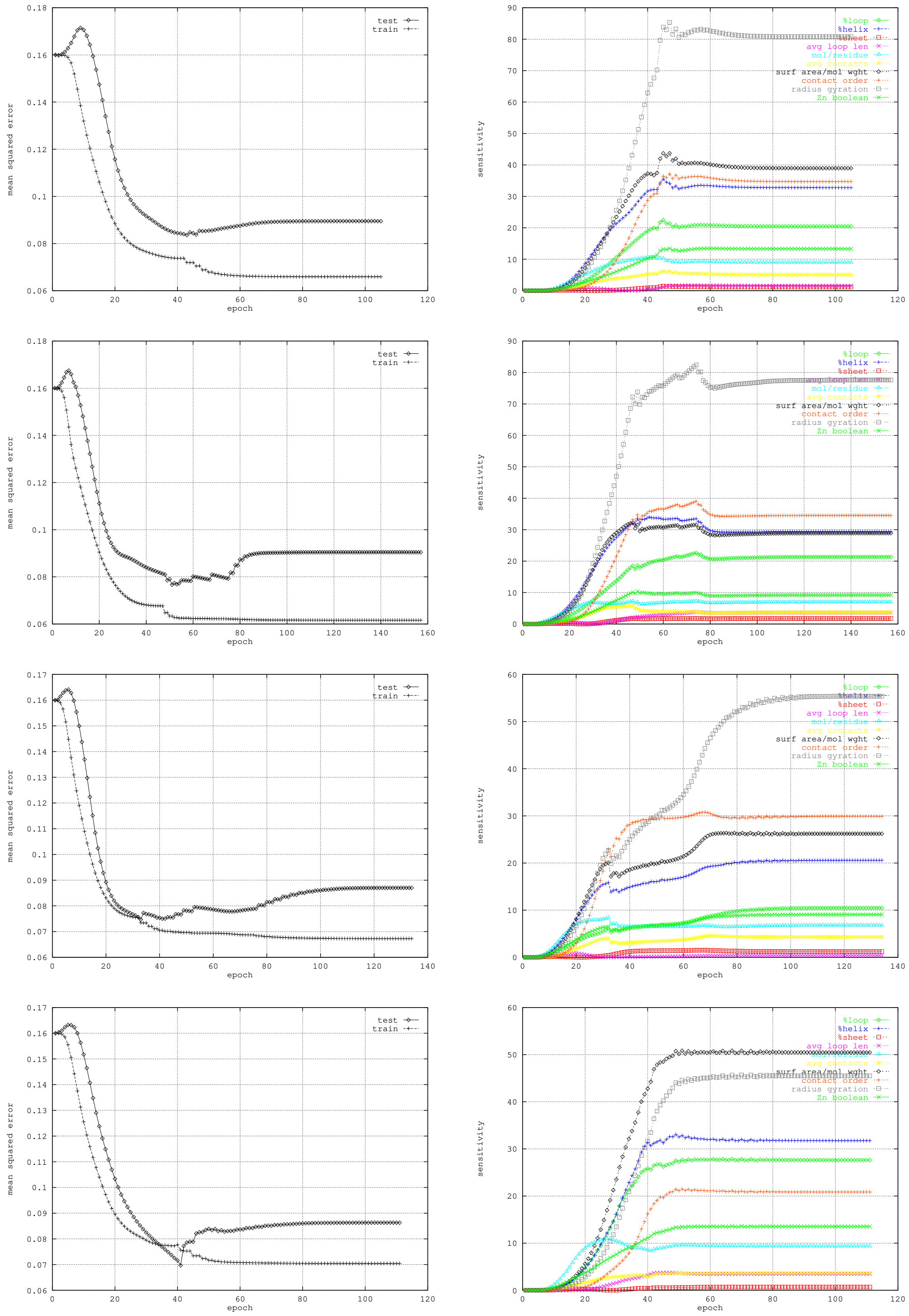


Figure 2.12: Training and test results and input sensitivity for the class experiments. The classes are, from top to bottom, thermolsyin-like metallo proteases, metzincans, subtilisin-like serine proteases, and trypsin-like proteases

Protease class	*Proteases* Correct	Incorrect	*Nonproteases* Correct	Incorrect
aminopeptidases	3	0	211	47
Aspartic Proteases	4	9	212	46
Carboxypeptidases	4	0	207	51
Cysteine proteases	0	9	209	29
Metzincins	7	2	207	51
Thermolysin-like metalloproteaes	5	0	208	50
Subtilisin-like serine proteases	8	0	207	51
Trypsin-like serine proteases	19	0	209	49

Table 2.2: The results of the class experiment where one class was removed from the training set and placed in the test set. The network was then tested on it's ability to identify members of a class it had never seen before

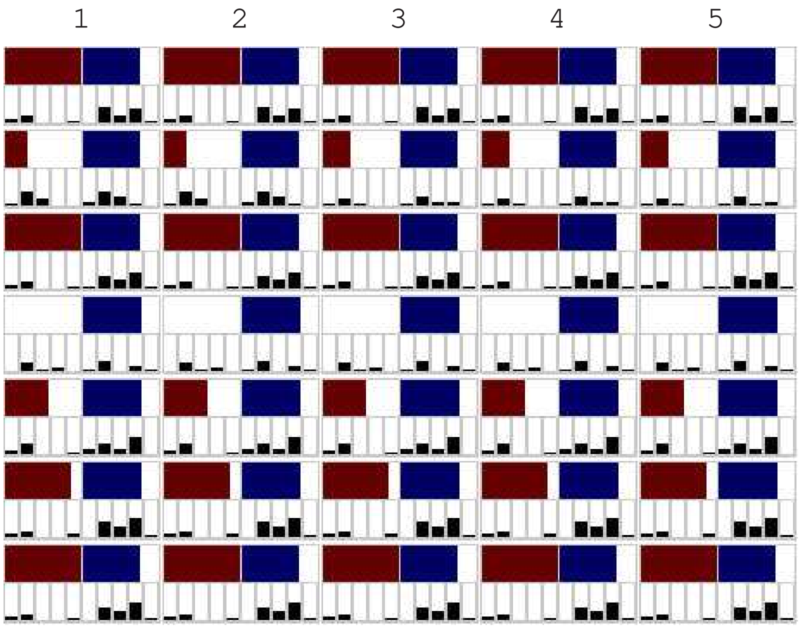


Figure 2.13: A NetPlot image showing the training and test error, and relative input sensitivites for 7 classes of protease. From top to bottom they are aminopeptidases, aspartic proteases, carboxypeptidases, cysteine proteases, thermolsyin-like metallo proteases, subtilisin-like serine proteases, and trypsin-like proteases

2.8 Discussion

We have developed a new means with which to predict a protein's membership in a class of enzymes known as proteases. This was accomplished by training a neural network to recognize distinct patterns of global structural features taken of proteases taken from the PDB. The performance of the network on novel protein structures indicates that our data set construction and selection of features were informative enough to allow the network to predict the membership of structures never seen before with 86% accuracy. Although we can not say with certainty whether the features we selected to train the network on are directly related to function, they do provide enough information to allow us to predict with a high degree of confidence a protein belongs to this class of enzymes.

During training and testing, interesting results were observed that may indicate the ability of the network to detect structural features related to the protease family of proteins. In particular several of the false examples incorrectly identified as proteases belong to the O-glycosidase family. The output of the network after presentation of these proteins was considerably higher than the output of other false examples that were incorrectly identified as proteases. These particular enzymes may share structural features with the true proteases since their function is analogous in that they catalyze the cleavage of glycosidic bonds. These may have arisen from similar selective pressures during evolution since glycosidases may have a weak affinity for peptide bonds as a substrate. The network's ability to identify the relatedness of these structural features could suggest an alternate role for the network in examining evolutionary ties between protein structure and function.

Closer inspection of input sensitivity provides a better idea what features are having the biggest influence on the networks decision of protease membership. In particular surface area per molecular weight, average loop length and average number of contacts ranked high. This follows with the statistical analysis indicating compactness as distinguishing features of proteases which might help them protect against auto-degradative processes. Interestingly all true examples incorrectly identified as false showed low sensitivity to the surface area per molecular weight input. Closer inspection of the features of misclassified proteases might prove insightful. The zinc boolean input, although not ranked high overall, did significantly improve the performance of the network in correctly classifying metallo-proteases. A low overall sensitivity, coupled with an improvement to performance in a specific class of protease, strongly suggests the networks ability to use that input to correctly identify examples exhibiting that feature. This is an important consideration when using the relative magnitudes of the sensitivity to judge the importance of data.

Evaluating which inputs should remain and which should be culled is was not an easy process. After performing a series of experiments aimed at eliminating information that appeared not to contribute significantly to the networks learning, the overall performance only declined. Based from the reorganization of the sensitivities following the elimination of an input (data not shown), the usefulness of data with high sensitivity was presumably linked to inputs that had low sensitivity. The performance of the net, particularly with false examples, dropped considerably after the remove of just one of the 10 inputs with lowest sensitivity. This could suggest one of two things: 1) a higher-order inner-dependence of inputs that is not measurable with sensitivity analysis 2) inputs that are not informative overall, but rather for correctly identifying a specific class of protease, as might be case with the zinc boolean input. A thorough inspection of all permutations of possible input sets was not done, though based on the results culling this might be the next step in a rigorous analysis of the importance of the data presented to the network.

This predictive technique has immediate use in the field of structural genomics. Using our neural network to predict the class of a protein's structure following x-ray or NMR studies could greatly speed the process of new fold and function discovery. As structure determination becomes faster, particularly due to the emergence of new automated procedures, our predictor could help provide a 'first-look' functional screen based on the charateristics of the protein. The classification of new protein structures could also assist in assigning functions to homologous proteins in a variety of other organisms.

Chapter 3

DINAMO: An Interactive Protein Alignment and Model Building Tool

3.1 Introduction

Proteins - polymers of amino acids residues, proteins can assume many different shapes depending on their composition, environment, and interactions with other proteins. It follows that obtaining a model of their three-dimensional structure is a very valuable part of understanding their role in an organism [44][46]. Information about a protein's fold, domain arrangement, and packing can provide biochemists

[2]This work has been previously published as Bentz J. Baucom A. Hansen M. Gregoret L. Dinamo: Interactive protein alignment and model building. *Bioinformatics* 15(4):309–316, 1999

with the information needed to guide experiments aimed at a better understanding of their biological function as well as the mechanics of folding. Protein sequence information generally preceeds a structure determination by x-ray crystallography or NMR. A high-resolution structure is not always easy to obtain due to the complexities of protein purification and crystallization. Structure determination is also a time-consuming process in comparison with sequence determination.

To date, however, many protein structures have been solved and deposited in the Protein Data Bank [31]. These structures and their sequences may act as templates in two common protein modeling techniques, *comparative modeling*, and *fold recognition*. Using a protein with known structure these two types of modeling techniques can use the three-dimensional structure as a scaffold with which to position the amino acid residues of the sequence of interest. The scaffold molecule is commonly referred to as the template, and the sequence being investigated, the target. Evaluation of the three-dimensional superposition together with an analysis of alignments of the sequences of known structure and the target sequence can provide criteria with which to judge the quality of the resulting model.

Comparative modeling depends on the availability of a sequence of known structure with high amino acid identity relative to the target sequence [1]. Using the template structure as a scaffold, the target sequence can be modeled for the best fit. It has been shown that proteins with 50% identity have very small α-carbon root mean squared deviations ($<$1Å) indicating that they have highly similar folds [2].

Even a structure having just 25%-30% identity to the target sequence can still provide the biochemist with a model of sufficient quality to direct focused experiments. In the absence of a homolog the method of fold recognition may be applied to model a protein's structure. This method relys on the observation that some proteins with very dissimilar sequences share a similar fold. Fold recognition compares the target sequence against a library of unique folds and evaluates the quality of a hypothetical alignment using one of many different scoring criteria [7][36]. In this case the structures having the highest probability of being the fold the target sequence would adopt are chosen as candidates for modeling.

Once a model has been built, closer examination of the structure is important in evaluating its plausibility. Evaluation is best done in the context of both the three-dimensional model, and either a sequence similarity comparison, or a structural alignment, depending on the technique employed. Refinement is almost always necessary following evaluation of the model to satisfy both biochemical constraints as well as structural evidence that may be known.

To help facilitate the process of model evaluation and refinement, a simple, interactive, cross-platform compatible sequence alignment editor and structure viewer named DINAMO (dee-nah-moe) has been developed. DINAMO couples a suite of sequence alignment analysis tools with a three-dimensional structure visualization program. This provides an environment with which to perform a closer inspection and improvements to a model following the application of either a comparative modeling, or fold recognition algorithm. Based on information returned by DINAMO, the user may judge the plausibility of the proposed model and make appropriate changes to improve its fit. DINAMO does not automate this process, but rather provides the user with an integrated environment designed to expedite the types of tasks necessary for model evaluation and refinement. It is DINAMO's ease of use, cross-platform compatibility, and completely interactive nature that set it apart from other model building tools that are currently available [47] [24][34].

3.2 DINAMO Internals

DINAMO was written in Java and designed for use either over the web or on a local host machine. Java provides cross platform functionality and allows the program to run on a variety of operating systems including Macintoshes, PCs and UNIX. DINAMO was designed as a modular application. The application is distributed as a package and consists of four major components: a multiple sequence alignment editor, molecular graphics package, analysis tools, and an analysis tool manager. The organization and inter-communication of the components is shown in figure 3.1. The editor is used primarily for viewing and manipulating multiple sequence alignments. The molecular graphics package provides a 3D environment with which to visualize the target sequence in the context of a template structure. Analysis tools return information about amino acid properties and calculate conservation among multiple

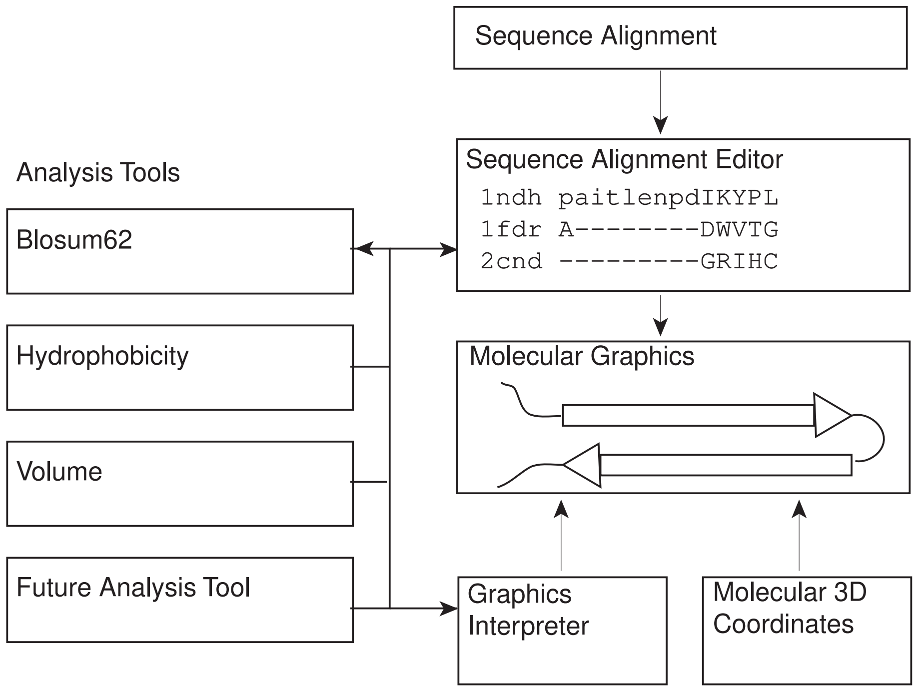


Figure 3.1: A diagram of the components and their connectivity within DINAMO

sequence alignments. The analysis tool manager allows the user to determine which tools are currently active and how their results ere displayed in the editor.

The editor may be used with or without a 3D-visualization package, and with or without analysis tools. Additionally the analysis tools are provided as plug-ins allowing the user to enable only those they find useful for the task at hand. Developers may also create new plug-ins to add to the suite of tools already packaged with DINAMO. Ideally, when new metrics and algorithms become available, new analysis tools will be created that will enhance the functionality of DINAMO.

3.2.1 Sequence Editor

The multiple sequence alignment editor represents the core of DINAMO. The editor allows the user to import sequences in the FASTA format, add insertions, deletions and to save changes following editing. The number of sequences imported by the editor is only limited by the host computer's resources, and allows the user to view the target sequence aligned against the template sequence as well as numerous other sequences of interest.

Amino acids in the editor are represented by tiles that indicate the identity of the residue. The background of tiles may be colored, the font may be colored, and a simple geometric shape may be drawn. These tile attributes allow three different results generated by the analysis tools to be mapped onto a residue simultaneously, possibly revealing new patterns in the information contained in a multiple sequence alignment. The user may customize which colors are used to represent the tile and font color attributes, and which attribute is mapped to a particular piece of information.

The attributes that are mapped to both the tiles and the 3D model may be static values that are fixed, or dynamic, which update with changes in the alignment. Static values may represent a particular amino acids charge, hydropathy, or volume. Dynamic values that are calculated in the context of a multiple sequence alignment such as residue conservation using a Blosum62 matrix [21], charge, or volume are continuously updated to reflect the current state of the alignment. When

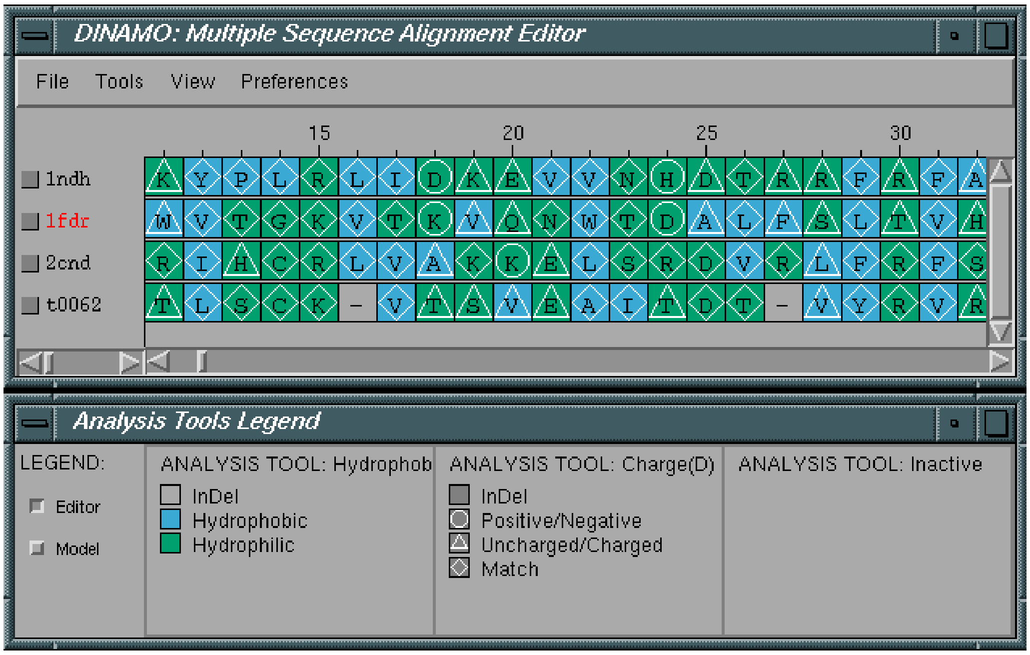


Figure 3.2: A snapshot of the sequence editor (top) and editor/graphics legend (bottom).

the alignment is changed, the attributes in the sequence editor and graphics that are mapped to these values are updated to reflect the change. Both static and dynamic attributes may also be mapped to the 3D-template structure using color. When a change is made to a sequence relative to the template structure the colors of the model are also updated to provide the user immediate feedback on the impact of the change. Coloring the model according to the sequence provides the user with a context in which to evaluate the plausibility of the alignment and help guide improvements to the model. DINAMO is limited to the display of a single 3D structure, but allows the user to select which sequence to model onto the template structure.

A floating legend displays which attributes are mapped to which analysis tool results for both the sequence alignment editor and the 3D graphics display. The editor and legend are shown in figure 3.2. Several additional pieces of information may also be displayed in the editor. Particular amino acid residues may be highlighted with one of four colors and labeled with their position in the sequence independent of number changes caused by insertions and deletions. A schematic representation of the secondary structure of the template may also be displayed above the alignment.

3.2.2 Molecular Graphics

The molecular graphics package of DINAMO provides the user with a 3D representation of the template structure that is directly coupled to the multiple sequence alignment editor. Using color, the model can display the current state of a sequence in the context of an alignment. The interface between the editor and graphics is dynamic, allowing the template structure coloring scheme to be syncronized with a particular sequence in the editor that a user has selected to map onto the structure. Any changes made to the alignment are immediately reflected in the coloring of the 3D model. The user may also mark specific residues in the editor and monitor their position in the 3D structure with the use of labels. This can be very useful in correctly positioning amino acids that are in important secondary structural elements or sites of reactivity.

The web-based version of DINAMO uses MDL Chime [39] for visualization of 3D structures. Using the LiveConnect interface in Netscape Navigator, and Chime's script enabled browser plug-in, DINAMO may issue commands to Chime that allow different PDB files to be viewed, and display attributes to be dynamically updated to reflect the current state of the sequence alignment. The standalone version of DINAMO currently uses RasMol [11] as a graphics package in a UNIX environment. When DINAMO is run as a standalone application it may spawn programs such as RasMol, and communicate with them via a standard input interface. Because Chime and RasMol have very similar scripting, the Chime interface was adapted to work with RasMol.

The molecular graphics interface was designed to be independent of the modeling program used in visualization. The interface allows for other programs to be used in conjunction with the editor by providing an interface able to translate DINAMO commands into any other graphics language.

Graphics Implementation

The graphics portion of DINAMO is an extensible collection of Java classes that facilitates the addition of new molecular visualization packages. The classes included with DINAMO demonstrate the ease with which subclasses can inherit the functionality needed to dynamically link with both the editor and analysis tools. A schematic of the Java class hierarchy for the graphics package is shown in figure 3.3.

To receive the results of analysis tools, a *PluginClient* class was created to act as an interface between plug-ins and a graphics implementation. All graphics implementations must subclass the PluginClient to gain access to PluginResult classes that contain the data generated by analysis tools. A *PluginResult* is an abstract class that defines the methods necessary for communication, and only implements a minimal set of its own methods. The responsibility of interpreting the results is left to the client class. The client class may be any other object that utilizes the results of an analysis tool. For the molecular graphics incorporated into DINAMO this was the *Graphics* class.

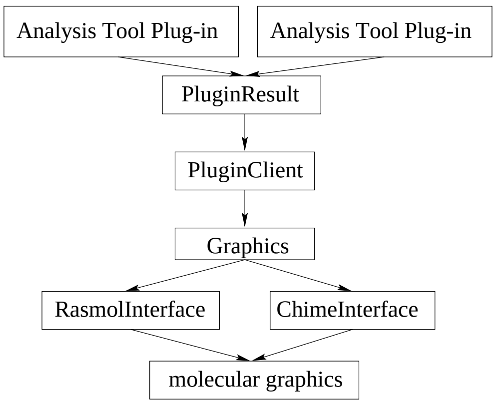


Figure 3.3: A diagram of the graphics class hierarchy in DINAMO

In order to interpret the results returned by an analysis tool the Graphics class was created to encapsulated a PluginResult class. The Graphics class inherits methods from PluginClient, and implements those methods that were defined to be abstract. The Graphics class is also an abstract class implementing only a very limited number of simple methods and declaring several that are essential to a program that will translate plug-in results into molecular graphics. The primary aim for this class is too define a more specific type of PluginClient but still remain abstract enough to allow extensibility.

Chime was an adaptation of RasMol to a browser plug-in and therefore shares a majority of its scripting capabilites with its parent application. Taking advantage of this, an abstract *Rasmol* Java class was created that extends the Graphics super-class. This class is primarily responsible for translating information stored in

PluginResult classes into the Rasmol/Chime scripting language. The translation is performed for a variety of operations including loading a model from the editor, coloring the model as a result of information returned by an analysis tool, and labeling when markers are set in the sequence editor. The Rasmol class handles all but the most detailed aspects of graphics communication, avoiding the specifics of graphics communication between Java and the Chime browser plug-in or Rasmol application.

To handle the information exchange between the Java classes that compose DINAMO and the external viewing devices such as the Chime plug-in and RasMol application, several interface classes were created. The *ChimeInterface* handles LiveConnect communication between the Java based Rasmol scripting class, and the scripting interface of the Chime plug-in in Netscape. The ChimeInterface also generates dynamic HTML to load the plug-in a web page and create a text label of the PDB model ID. The *RasmolInterface* class is responsible for launching RasMol and managing a standard input pipe through which scripting commands are passed from Java to the application. An additional class, the RasmolListener, has the ability to receive information back from Rasmol through a standard output pipe. This class continuously runs in a thread listening at the standard output pipe created by the RasmolInterface. Although not fully implemented, this could be used in future versions of DINAMO to pass information back to the editor, such as which residue a user has clicked on in the viewer in order to highlight those residues in the sequence.

A *NullInterface* class is also provided with DINAMO which acts as a shunt for information generated as a result of the Graphics class. This class is used when disabling the use of molecular graphics in DINAMO and using the multiple sequence editor alone. An additional class, the *GraphicsAttribute*, was created as a storage class used to pass information between methods in the graphics pipeline. The positions of insertions, deletions, and markers generated by the sequence editor are encapsulated in this class which provides access methods for retrieving the data when it is needed.

3.2.3 Analysis Tools

The analysis tool serves as the basis for improvement of models both at a sequence level as well as a three-dimensional model level. A list of current tools is shown in table 3.1. There are two classes of analysis tool, static and dynamic. Static analysis tools display information that is independent of the alignment for each amino acid residue. For instance, a sequence may be colored according to the charge of amino acids. In this context the coloring scheme is fixed, and therefore considered static. Dynamic tools color tiles in a sequence relative to a residue-by-residue comparison between a selected sequence, and the template sequence. When a change is made relative to their alignment, the coloring scheme changes to reflect the new state using one of the dynamic metrics. An independent analysis tool known as the *Pie Chart* may be used to display the percentages of residues in the currently selected sequence

Plugin	Static	Dynamic
Blosum62	-	X
Buried Sheet Propensity	X	-
Charge	X	X
Edge Sheet Propensity	X	-
Environmental Preference	X	-
Helix Propensity	X	-
Hydrophobicity	X	X
Volume	X	X

Table 3.1: The list of static and dynamic plugins available in DINAMO.

that are in each state as a result of the output of an analysis tool. When used in conjunction with a static analysis tool the Pie Chart can provide the user with information such as the overall percentages of charged and non-polar residues in a sequence. If used in conjunction with dynamic analysis tools the Pie Chart updates continuously to show the results of a change in a multiple sequence alignment on a global scale. This can be particularly useful when trying to evaluate whether a local change in the alignment is a global improvement based on changes in the percentages of residues in a given state. The Pie Chart and graphics window of DINAMO are shown in figure 3.4.

3.3 CASP

Every two years the Critical Assessment of Protein Prediction (CASP) [50]contest provides the protein structure prediction community with a host of protein sequences with structures that are in the process of being solved by NMR and x-ray crystal-

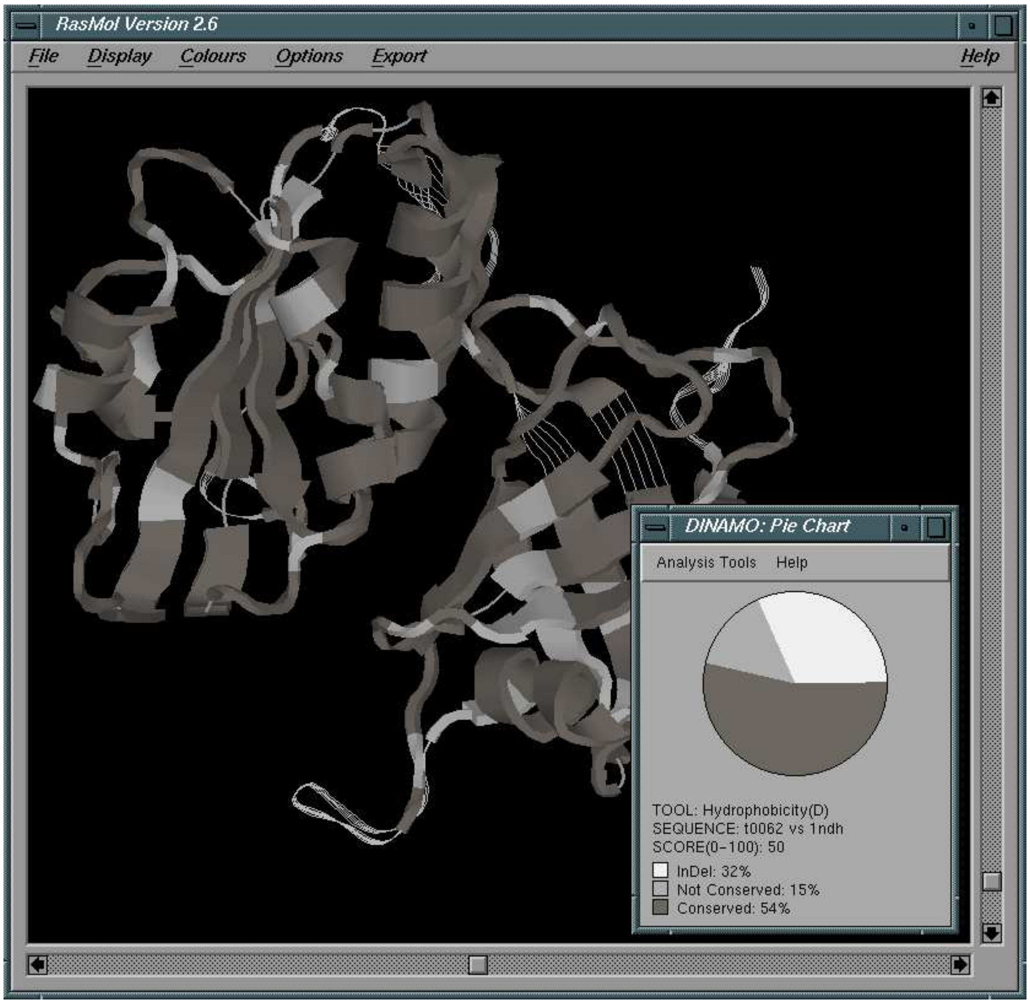


Figure 3.4: A snapshot of the DINAMO graphics window and Pie Chart (foreground)

lography. Participants are provided with the protein sequence, the organism from which it was isolated, and sometimes a short description of its function. They are then allowed to submit structural predictions using any method they choose. Following the contest, the proposed structures are compared against the solved structures and evaluated for their quality using several different criteria including root mean squared deviation (rmsd) of α-carbons atoms in the peptide backbone. The results are posted to allow participants to compare their techniques and are discussed when the meeting is held. To evaluate the usefulness of DINAMO in a realistic prediction setting, it was used to predict four structures for the 3rd CASP meeting.

Several programs were employed to discover both remote sequence and structural homologs and to create multiple sequence alignments. The general scheme of the approach is shown in figure 3.5. The CASP target sequences where first run through a BLAST sequence search, and the fold recognition program THREADER [23]. The results of these programs were compared to evaluate which was a better candidate for a plausible homolog to be used as a template structure. The sequence of the template was then submitted to the DALI [4] server to find other structural homologs. A multiple sequence alignment was then made between the target sequence, the template sequence, and any structural homlogs returned by DALI using the pro-

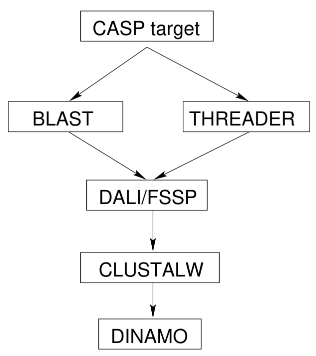


Figure 3.5: Flowchart showing the order of programs used to make a prediction for CASP using DINAMO

gram CLUSTALW [45]. This established a working template sequence/structure from which to make refinements using DINAMO.

The refinement process in DINAMO involved several different criteria to evaluate the quality of the alignment, and direct further improvements. Using the results of the analysis tools, the sequences were manually aligned to 1) maximize the number of non-polar residues in the core of the protein 2) maximize exposed hydrophilic residues 3) maximize hydrophobic residue-residue contacts 4) maximize favorable charged and polar residue interactions, and 5) minimize buried charged residues. Additionally, insertions and deletions were placed in loop regions whenever possible to avoid disrupting secondary structural elements.

3.4 Results

The multiple sequence alignment generate for target t0062 of CASP3 is shown in figure 3.6. The alignment includes the template structure, `1ndh`, structural homologs `1fdr` and `2cnd`, the initial alignment prior to refinement in DINAMO, and the final alignment that was submitted with the coordinates of the template sequence to CASP. The results based on the rmsd for the 3 targets that we submitted predictions for are shown in figures 3.7, 3.8, and 3.9, for targets t0055, t0062, and t0085 respectively. The results show our position among all other groups that also submitted predictions for the same target as well as the rmsd for each prediction on a per-residue basis.

3.5 Discussion

Our performance over the three targets predicted for CASP are encouraging. We scored high relative to other groups in all of our submissions. This is good, in light of the fact that many groups used proprietary algorithms and techniques for their predictions that are believed to be state of the art. Together these results show that valid predictions may be made with automated programs, but may still be improved manually using tools such as DINAMO.

DINAMO was designed to act primarily as a workspace in which to evaluate and refine an alignment. Since DINAMO does not provide any energy minimization, packing, or side chain placement algorithms, its accuracy is based solely on the quality of the sequence alignment used to model a structure. Thus if a sequence alignment of sufficient quality cannot be obtained, either manually, or through automated means, the use of DINAMO in building a plausible model is severely limited.

Several important improvements could be made to DINAMO enhance its functionality. Specifically the graphics portion of DINAMO is limited to the display of a single attribute at a time. Work has been done to improve this by attempting to map several different attributes simultaneously using varying atomic and dot sphere sizes, as well as labels to give the user visual cues on values displayed in the sequence

```
         1ndh paitlenpdIKYPLRLIDKEVVNhDTRRFRFALPspehILGLPVGQHIYLSARIdgnLVIRPYT
         1fdr A--------DWVTGKVTKVQNWTDALFSLTVHAP----VLPFTAGQFTKLGLEI---RVQRAYS
         2cnd ---------GRIHCRLVAKKELSRDVRLFRFSLPSPDQVLGLPIGKHIFVCATIEGKLCMRAYT
t0062(init.) ---------TTLSCKVTSVEAITDTVYRVRI---VPDAAFSFRAGQ--YLMVVMD-ERDKRPFS
t0062(final) ---------TTLSCK-VTSVEAITDT-VYRVRIV-PDAAFSFRAGQ--YLMVVMD-ERDKRPFS

         1ndh PVSSDDdkGFvDLVIKV----YFKdthp-kfpaGGKMSQYL-ESMKIGDTIEFR-GPNGLlVYQ
         1fdr YVNSPDN-PD-LEFYLV----TVP---------DGKLSPRL-AALKPGDEVQVVSEAAGFFVLD
         2cnd PTSMVDEIGH-FDLLVK----VYFKNEHpKFPNGGLMTQYL-DSLPVGSYIDVK-GPLGHVEYT
t0062(init.) MASTPDEKGFIELHIGA----SEINLYA-K--A--VMDRIL-KDHQI--VVDI---PHGEAWLR
t0062(final) MASTPDEKGFIELHIGASEINLYAK---------AVMDRILKDHQI---VVDI---PHGEAWLR

         1ndh -GKGkfairpdkksspviktVKSVGMIAGGTGITPMLQVI-RAIMkd-PDDHT-VCHLLFA-NQ
         1fdr EVPH----------------CETLWMLATGTAIGPYLSIL-RLGK---DLDRFKNLVLVHA-AR
         2cnd -GRGSFVIN----GKQR--NARRLAMICGGSGITPMYQII-QAVLRDqPEDHT-EMHLVYA-NR
t0062(init.) ----------DDEE-------RPMILIAGGTGFSYARSIL-LTALARNPN--R-DITIYWG-GR
t0062(final) -DDEE----------------RPMILIAGGTGFSYARSILLTALARN-PN-R--DITIYW-GGR

         1ndh TEKDILLRP-eLEEL-RNEHSARFKLWYTVDRAP---ea-w-dysQGFVN--E--EMIR-dhlp
         1fdr YAADLSYLP--LMQELEKRYEGKLRIQTVVSRET---AA-G-SLTGRIPALIESGELESTIGLP
         2cnd TEDDILLRD--ELDRwAAEYPDRLKVWYVIDQVKrpeEG-W-KYSVGFVT--E--AVLR-EHVP
t0062(init.) EEQHLYDLC-ELEAL-SLKHPG-LQVVPVVEQPE---AG-W-RGRTGTVL--T--AVLQ-DHGT
t0062(final) -EEQHLYDLCELEAL-SLKHP-GLQVVPVVEQ-P---EAGWRGRT-GTVL--T--AVLQ-DHGT

         1ndh PPEEePLVLMcgpPPM--I-Q--YA-CLPNL-ER-VGhp-k-E-R--------CFAF---
         1fdr MNKETSHVMLC--GNP--QMV--RD-TQQLLKET-RQM----T-KHLRRRPGHMTAEHYW
         2cnd EGGDDTLALA--CGPPpmI-Q--FA-ISPNL-EK-MKYD-M-AnS--------FVVF---
t0062(init.) LAEHDIYIAGRFEMAK--I-ARDLF-CSERN-AR-EDRL-F-G-D--------AFAFI--
t0062(final) LAEHD-IYIA-GRFEMAKI-----ARDLFCS-ERNAREDRLFG-D--------AFAFI--
```

Figure 3.6: The target t0062 alignment before (init.) and after (final) refinement in DINAMO. The structure 1ndh was used as the template and both 1fdr and 2cnd as structural homologs.

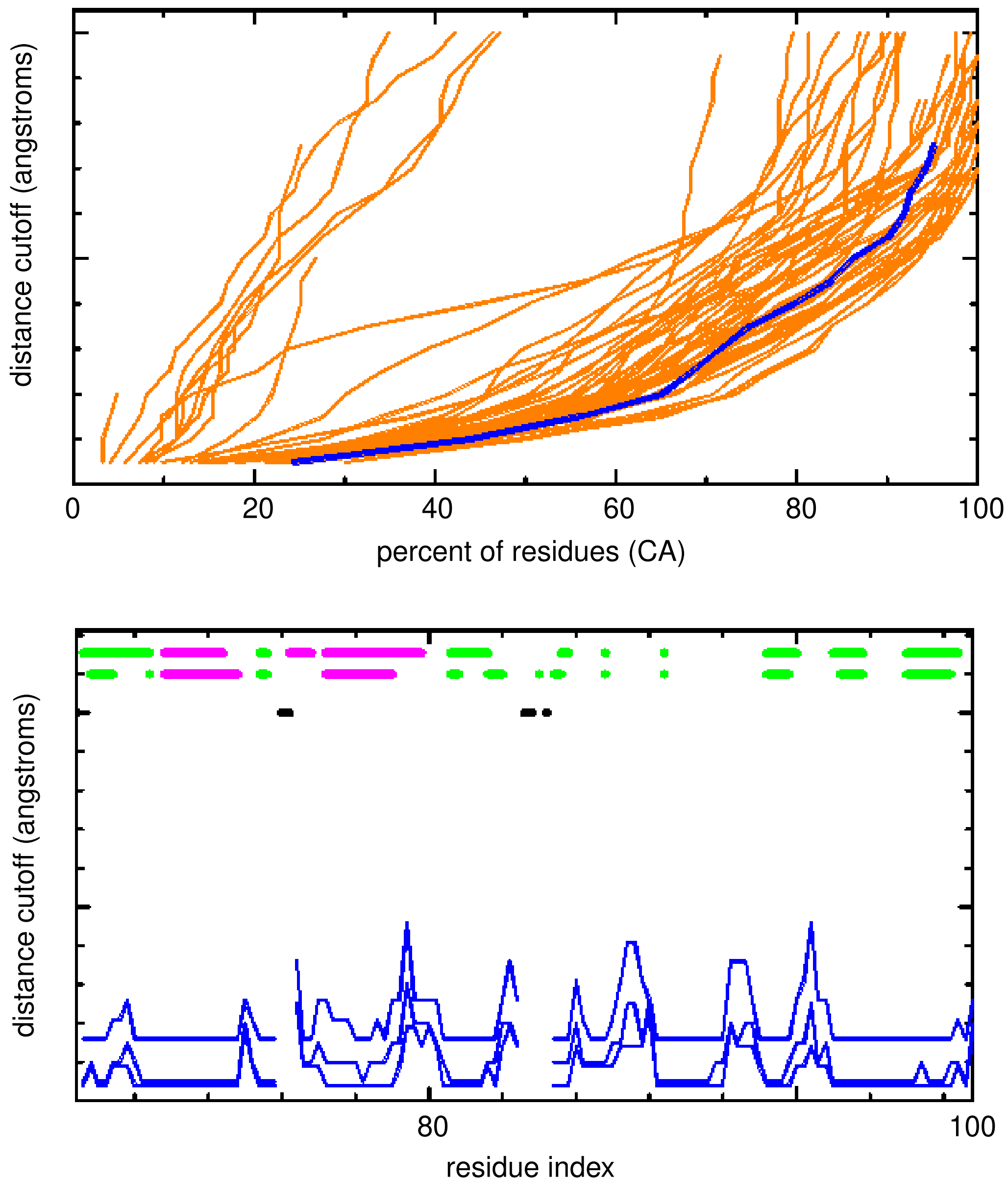


Figure 3.7: The results from the CASP3 contest showing our position relative to other groups for target t0055 (top) based on the root mean squared deviation of our model and the actual structure, and the rmsd on a per-residue basis (bottom)

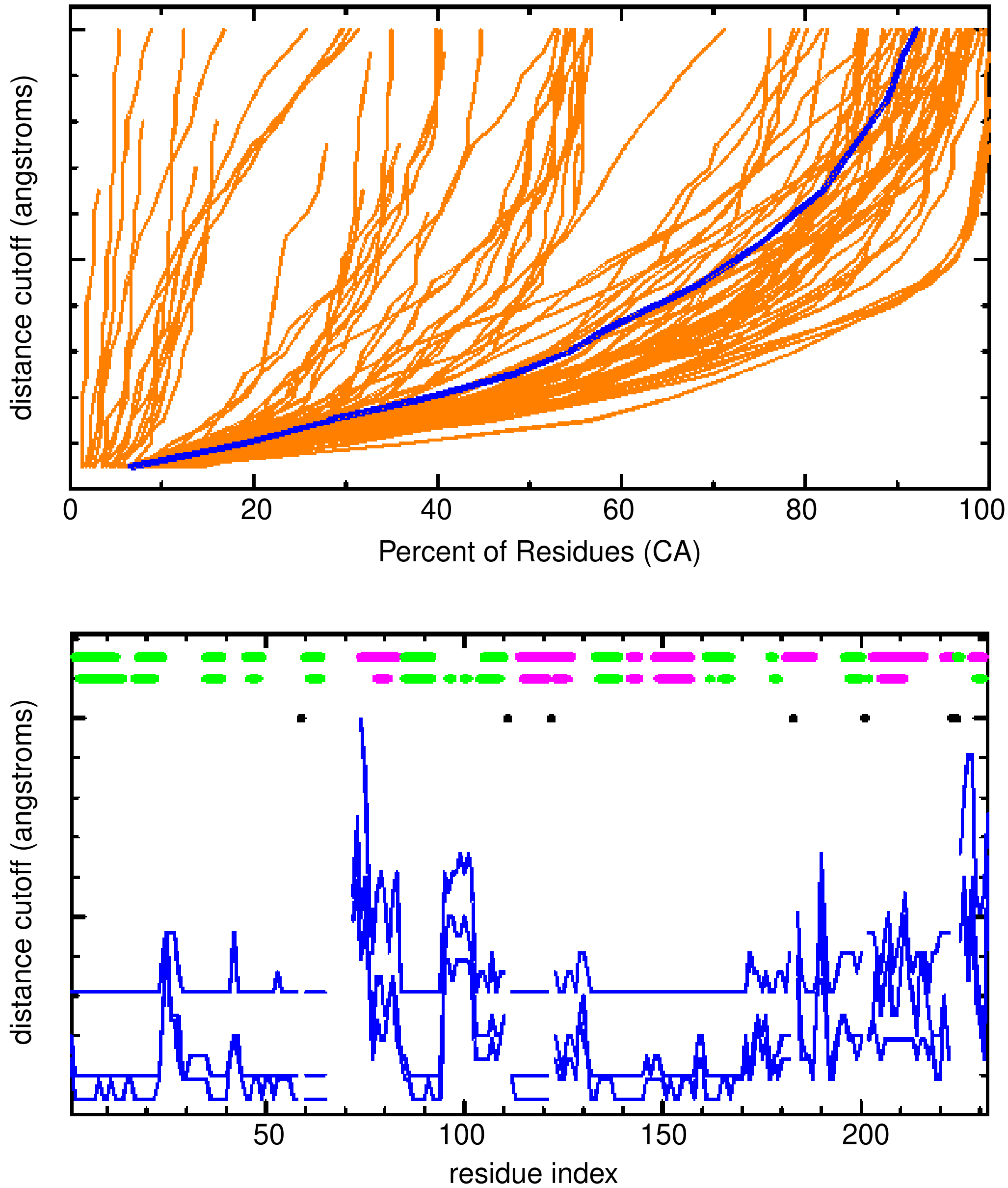
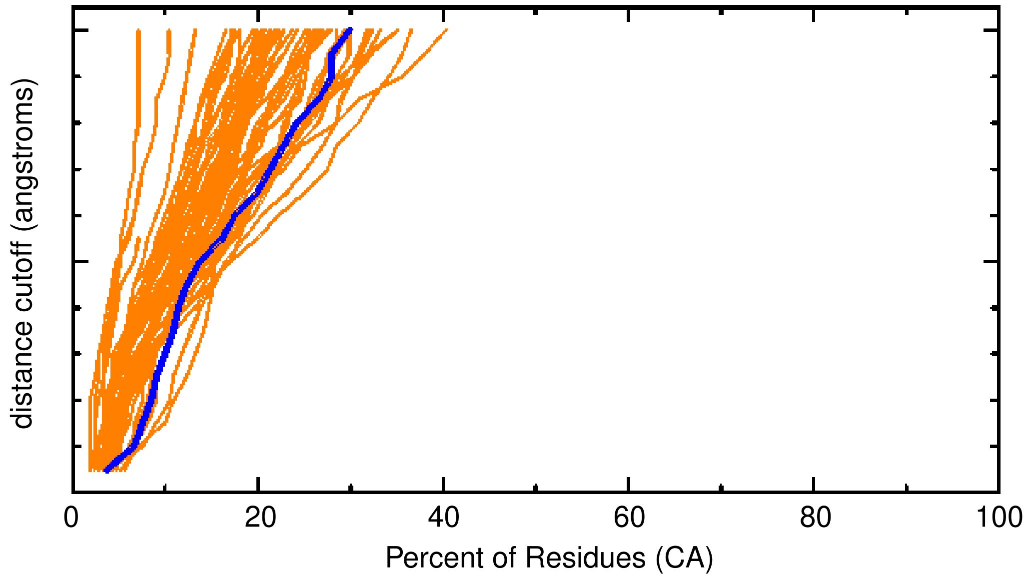


Figure 3.8: The results from the CASP3 contest showing our position relative to other groups for target t0062 (top) based on the root mean squared deviation of our model and the actual structure, and the rmsd on a per-residue basis (bottom)



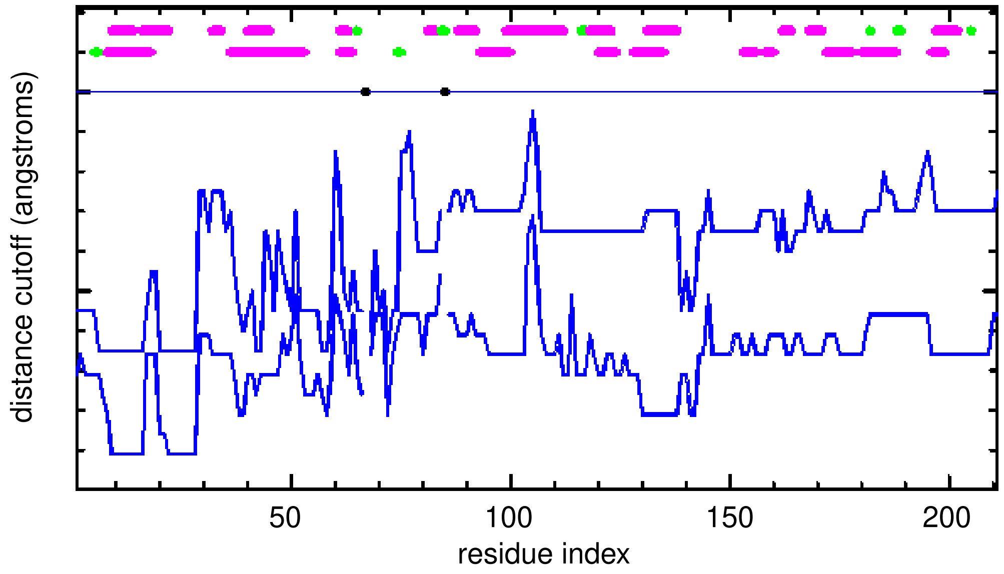


Figure 3.9: The results from the CASP3 contest showing our position relative to other groups for target t0085 (top) based on the root mean squared deviation of our model and the actual structure, and the rmsd on a per-residue basis (bottom)

editor.

Improvements in the standalone version of DINAMO, and discontinued support for the web version would help DINAMO's stability. The number of permutations possible with Netscape versions, Chime versions, Java virtual machine implementations, and operating systems has made development of the web version almost impossible. Accounting for these differences and improving DINAMO to work under varying conditions proved very difficult. A robust RasMol interface for both Macintosh and PC's could easily substitute for Chime, stabilize the program and help the majority those who regularly use one of these two operating systems.

Chapter 4

β-Sheet Structure Prediction Using Neural Networks

4.1 Introduction

The details of protein structure are multi-faceted. Composed of amino acids residues that can assume a variety of conformations by virtue of single bonds having rotational freedom, polypeptide chains have the potential to fold in nearly limitless ways. Interactions with other proteins, large and small molecules such RNA, and varying environmental conditions, contribute to the structural possibilities a protein may sample. Understanding these interactions in enough detail to accurately predict in advance the structure of a protein is thus a complicated task best taken with a reductionistic approached.

Fortunately many proteins observed in nature share common structural motifs that present a means with which to begin methods of prediction. Secondary structural elements created by both local and distant interactions between amino acid residues, namely α-helices and β-sheets, have been well studied in this context. Using the database of known proteins structures deposited in the Protein Data Bank, statistical analyses have been applied in an effort to better understand the nature of these interactions and arrive at set of rules that could be used to predict their formation in novel sequences. Several techniques have been developed such as PHD that have better than 76% accuracy in locating regions in sequences likely to assume these types of interactions [41]. Yet there are deficiencies in these techniques with regards to β-sheets that involve more distant amino acid residues coming into contact. α-helices develop through the interactions of adjacent amino acids in the polypeptide chain. Hydrogen bonding between polar groups found in the peptide backbone help stabilize these coiled structures. β-sheets form from the interaction of extended regions of polypeptide chains, called β-strands, hydrogen bonding to one another. The extension of the polypeptide chain is a local interaction caused by a series of amino acid residues assuming a regular backbone conformation. The association of individual β-strands to form a β-sheet, is on the other hand, frequently formed from interactions between sequentially distant amino acid residues. The large distances through which β-strands find one another frequently finds them being classified as tertiary structural elements.

The forces that drive β-sheet formation are poorly understood [25] and still harder to accurately predict [41]. It is not clear how distantly separated β-strands find one another in a natively folded protein, and amino acid recognition contributes to a proper alignment once the strands come together. A better understanding of this recognition and proper alignment, or β-strand register, is thus an interesting topic of research with immediate applications in the area of protein structure prediction.

I describe here an attempt to derive a set of rules governing β-sheet formation from β-strands. Using amino acid identities and properties compiled from a data set of protein structures determined by x-ray and deposited in the PDB, a single layer neural network was trained in effort to first discriminate true β-strand pairings from false ones. A second two layer competitive neural network was then devised to classify the interaction as one of several different specific structural classes as defined by the context in which the β-sheet was found.

4.2 β-Sheet Structure

Originally described by Linus Pauling [40], β-sheets are composed of extended regions of the polypeptide chain known as β-strands. The strands are held together primarily by hydrogen bonds formed between a donor amide proton of a peptide bond nitrogen, and carbonyl oxygen acceptor of the adjacent peptide backbone. Two or more β-strands may hydrogen bond in this fashion to form these characteristic "pleated" structures. The direction of the peptide backbone relative to each strand, running from the amino to carboxy terminus, can vary giving rise to two distinct hydrogen bonding patterns. If both strands run in the same direction the β-sheet is referred to as parallel, if they run in the opposite direction the sheet is referred to as anti-parallel. The hydrogen bonding pattern of these two forms easily distinguish them. A β-sheet may be composed of more than two strands and may have a heterogeneous mix of parallel and anti-parallel interactions.

Interactions between the atoms of neighboring amino acid residue side-chains is also an important contribution to both the stability and recognition of adjacent β-strands. When in an extended conformation, side-chains orient themselves in a regular periodic direction normal to the peptide backbone, and alternating between each face of the plane created by the β-sheet. In both parallel, and anti-parallel β-sheets the side chains are aligned proximally, bringing their atoms within distances conducive to non-covalent forces such as charge-charge attraction, hydrogen bonding and van der Waals interactions. It is these specific interactions which are thought to convey both recognition of sequentially distant strands and enforce their particular register once they have found one another [6][27].

Depending on the properties of the amino acid residues, and their location relative to the aqueous environment, β-stands may have polar and non-polar faces. Strands having predominantly polar residues are most frequently found at the edge of a β-sheet that can interface with the solvent, and those having predominantly hydrophobic residue content are found in the protein's core regions. If the face of a β-strand makes a regular switch between polar and non-polar residues we refer to it is as *bipolar*. Bipolar faces are created by alternating the composition of residues periodically so that hydrophobic side chains can point toward the core of a protein while adjacent residues on the opposite face of the sheet can be directed into the solvent.

Amino acids residues and their individual physical properties, together with hydrogen-bonding variants possible between β-strands and their location in a folded protein give β-sheets a broad spectrum of definable class diversity. Understanding their formation is thus best done in the context of these features and may provide clues to the mechanisms underlying observed β-strand pairings.

4.3 Methods

4.3.1 Data Set Construction

A non-redundant data set of proteins for the first true training and test set was initially constructed from 207 structures deposited in the PDB. These structures all contained at least one β-sheet, had better than 2.3 Å resolution, and a maximum of 30% homology between any two protein in the data set. Multimeric homodimers were processed to remove any redundant polypeptide chains. The program DSSP [5] was used to identify β-sheets in candidate structures as well as make assignment consistent among all members of the data set. The data set was later enlarged to 339 representative proteins after additional structures became available from the PDB. Criteria for protein selection were as described above for the first data set. This enlarged data set was used for all subsequent experiments. The PDB identifiers of the proteins in the enlarged data set are shown in table 4.1.

False examples were generated by randomizing pairings between true strands. After false pairings were created, different strand-strand registers were generated by sliding the β-strands relative to one another and capturing each new pairing possibility. All randomized pairings were then compared against all trues to prevent inadvertently creationing a true example.

4.3.2 Representation of Sequence Information

In order to present the neural network with meaningful training and test examples a representation of β-sheet structure was created. Based on prior experiments involving β-sheet structure analysis and neural networks, a window of 10 amino acids was chosen as the scope of an individual β-strand pairing example [20][43]. Designating two paired amino acids in a β-sheet i and j, the two amino acid residues adjacent on either side of each of the two paired residue were included in the window. The windowing scheme is illustrated in figure 4.1 with labels indicating particular positions relative to the residues i and j. If the identity of an amino acid was included in the training example, a bit vector representation was generated. Using a vector

1aak	1aan	1aba	1acf	1add	1ade	1amp	1aor	1apa	1arc
1asq	1ast	1atl	1atp	1bam	1bbp	1bcx	1bec	1bhp	1bia
1bmd	1bnd	1bnh	1bov	1bpb	1btn	1byd	1cbg	1cca	1cdc
1ceo	1cew	1chd	1chm	1clc	1cmc	1cpt	1csn	1ctf	1ctm
1ctu	1cxi	1cyx	1daa	1dea	1dfn	1dhr	1dpg	1dri	1dsb
1dts	1dup	1dyn	1eaf	1edt	1eny	1epb	1esl	1ezm	1fba
1ffe	1fiv	1fna	1frd	1frn	1fxd	1gal	1gar	1gca	1gcb
1gd1	1gky	1goh	1gox	1gp1	1gpc	1gpm	1gpr	1grj	1gse
1gtq	1han	1hip	1hmy	1hny	1hoe	1hpl	1hpt	1hqa	1htp
1hur	1hxn	1hyh	1hyl	1ino	1irk	1knb	1kpt	1lat	1lba
1lcp	1lfa	1lfi	1lla	1llo	1ltt	1mat	1mdc	1mdt	1mhl
1mla	1mml	1mmr	1mol	1mpp	1msa	1msc	1mup	1nar	1nba
1nci	1nfp	1npx	1nsd	1oac	1omp	1onc	1opb	1opr	1otf
1otg	1ova	1ovb	1oxa	1pbn	1pbp	1pda	1pea	1pfk	1pgx
1phh	1php	1pht	1pii	1plq	1pne	1pnf	1pnk	1poc	1pox
1prn	1psp	1ptr	1pya	1pyd	1qor	1rbp	1rcb	1rec	1reg
1rhd	1rib	1ris	1rnb	1rsy	1rtp	1rve	1sac	1sbp	1sca
1sce	1ses	1shb	1shg	1slt	1smn	1smp	1spa	1sts	1tca
1tcs	1ten	1tfd	1tgx	1the	1tie	1tif	1tig	1tml	1tnd
1tpl	1trb	1trh	1trk	1tup	1ubq	1udh	1urn	1vca	1vcc
1vhh	1vmo	1vrt	1vsf	1wap	1wsy	1xad	1xxa	1xyz	2aai
2acu	2ak3	2ayh	2aza	2bbk	2bop	2btf	2cbe	2cht	2ci2
2cmd	2cpl	2ctv	2cym	2fcr	2fke	2fxb	2glr	2glt	2gn5
2gsq	2had	2hft	2hhm	2hip	2hpd	2hpr	2kau	2lao	2liv
2ltn	2mad	2mcm	2mnr	2msb	2nck	2nrd	2olb	2oxi	2pab
2paz	2pf2	2phl	2phy	2pia	2pk4	2por	2reb	2rhe	2rn2
2rsl	2rsp	2sar	2scp	2sil	2sn3	2sns	2sod	2stv	2tgi
2tmd	2trx	3adk	3b5c	3cd4	3chy	3cox	3dfr	3dni	3gap
3grs	3hsc	3il8	3lad	3ovo	3rp2	3rub	3sc2	3sdp	3tgl
4blm	4bp2	4cla	4fgf	4gcr	4htc	4icd	4tms	4ts1	5cts
5dfr	5er2	5fbp	5fd1	5fx2	5gst	5hvp	5pep	5rub	5sga
5tnc	6ebx	6lyz	6taa	7enl	7lzm	7nn9	7pcy	7tim	821p
8acn	8atc	8cat	8cpa	8cpp	8dfr	8fab	8gpb	8i1b	8rxn
8tln	9abp	9ldt	9pap	9pti	9rnt	9rsa	9wga	9xia	

Table 4.1: The PDB identifiers of 339 proteins used in the β-sheet classification experiments

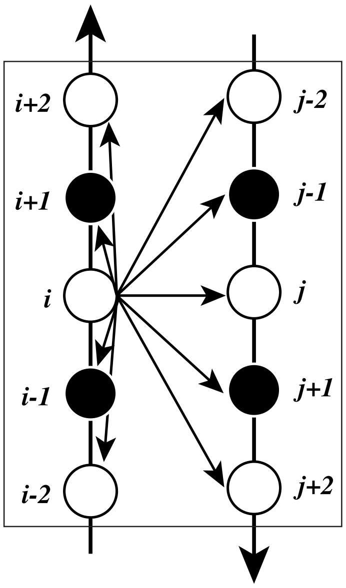


Figure 4.1: A diagram of the windowing scheme used for each β-sheet example created as input for the neural network.

of 20 bit values corresponding to an alphabetic list of amino acid identities, a single bit was set for a particular residue. Thus for the amino acid (A)lanine the first value of 20 would be set to 1 with all others set to 0, the second bit set to 1 for (C)ysteine and all others to 0, and so on. When incorporated into the input of a training or test example this added and additional 200 values; 10 residues in a window each with 20 values in each bit vector.

4.3.3 Feature Set Extraction

A subset of possible amino acid properties were selected on its ability to accurately and succinctly describe a residue's physical properties. These properties were derived from the amino acids as an independent entities as well as in the context of the environment in which the were found. Initially 12 properties were chosen from a previously published set [49]. Their names and values are listed in table 4.2. This list was later pruned to six properties after closer inspection of their importance in providing meaningful information to a neural network performing statistical analysis of β-sheet structure. The six properties were referred to as "properties-lite" and are shown in bold in table 4.2. All properties were normalized in a feature preprocessing step prior to neural network training to account for relative differences in magnitudes and metrics. In addition to the six properties, two other features were used in training: strand membership and chain separation. The strand membership of each amino acid residue was assigned based on the scope of the windowed region of β-sheet. Because the window is centered on a pair of residues $i - j$ in a β-sheet , with two adjacent residues included on either side of each residue in a strand, there is the possibility that the ends of the window could include residues not belonging to the β-sheet. To add this as an additional feature a boolean value was used in training experiments that indicated whether each residue in the window belonged to a sheet or not. In the prediction of a β-strand pairing in a novel protein this information could be derived from predicted strand regions.

Chain separation was a value indicating the sequential distance between the residues i and j that the windowed example was centered on. This information was provided to the network based on previous work that has indicated that the individual β-strands that compose a β-sheet are frequently close sequentially. Strands

Property	A	R	N	D	C	E	Q
Anti-parallel Sheet Propensity	0.9	0.82	0.54	0.54	1.19	0.85	0.85
Aromaticity	0.0	0.0	0.0	0.0	0.0	0.0	0.0
β-Strand Propensity	0.83	0.93	0.89	0.54	1.19	0.37	1.1
β-Turn Propensity	0.67	0.95	1.56	1.46	1.19	0.74	0.98
Bulkiness	11.5	14.28	12.82	11.68	13.46	13.57	14.45
Charge	0.0	1.0	0.0	-1.0	0.0	-1.0	0.0
Free Energy of Transfer	1.24	0.24	0.77	0.71	1.6	0.65	0.65
H-Donor	0.0	1.0	1.0	0.0	0.0	0.0	1.0
H-Acceptor	0.0	0.0	1.0	1.0	0.0	1.0	1.0
Parallel β-Strand Propensity	1.0	0.62	0.53	0.53	1.13	0.48	0.48
Polarity	8.1	10.5	11.6	13.0	5.5	12.3	10.5
Volume	0.68	0.67	0.59	0.56	0.59	0.63	0.64
	G	**H**	**I**	**L**	**K**	**M**	**F**
Anti-parallel Sheet Propensity	0.56	1.12	1.54	1.26	0.82	1.19	1.4
Aromaticity	0.0	0.5	0.0	0.0	0.0	0.0	1.0
β-Strand Propensity	0.75	0.87	1.6	1.3	0.74	1.05	1.38
β-Turn Propensity	1.56	0.95	0.47	0.59	1.01	0.6	0.6
Bulkiness	3.3	13.69	21.4	21.4	15.71	16.25	19.8
Charge	0.0	0.5	0.0	0.0	1.0	0.0	0.0
Free Energy of Transfer	1.24	1.01	1.48	1.36	0.0	1.3	1.36
H-Donor	0.0	0.5	0.0	0.0	1.0	0.0	0.0
H-Acceptor	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Parallel β-Strand Propensity	0.79	0.38	2.6	1.42	0.62	1.13	1.17
Polarity	9.0	10.4	5.2	4.9	11.3	5.7	5.2
Volume	0.58	0.64	0.83	0.83	0.74	0.71	0.74
	P	**S**	**T**	**W**	**Y**	**V**	
Anti-parallel Sheet Propensity	0.42	0.87	1.3	1.4	1.68	1.53	
Aromaticity	0.0	0.0	0.0	2.0	1.3	0.0	
β-Strand Propensity	0.55	0.75	1.19	1.37	1.47	1.7	
β-Turn Propensity	1.52	1.43	0.96	0.96	1.14	0.5	
Bulkiness	17.43	9.47	15.77	21.67	18.08	21.57	
Charge	0.0	0.0	0.0	0.0	0.0	0.0	
Free Energy of Transfer	0.89	1.01	0.95	1.24	0.83	1.42	
H-Donor	0.0	1.0	1.0	0.5	1.0	0.0	
H-Acceptor	0.0	0.0	0.0	0.0	0.0	0.0	
Parallel β-Strand Propensity	0.35	0.7	0.59	1.17	1.08	2.63	
Polarity	8.0	9.2	8.6	5.4	6.2	5.9	
Volume	0.7	0.58	0.66	0.71	0.68	0.79	

Table 4.2: The amino acid properties as set by Webster, Nambudripad, and Buturovic [49] and used as input for the neural network. The properties shown in bold are the subset used in the classification experiments

connected by small loop regions, known as β-turns, are common in many protein folds [33]. Again, for prediction of strand pairing in a novel protein, this information would be generated by secondary structure prediction that would assign the location of β-strands in the polypeptide sequence.

4.3.4 Topological Access Calculation

To classify β-sheet structures according to their context in protein structure a novel method of solvent accessibility was developed. This method is termed topological access to denote its analysis of the arrangement of β-sheets in terms of a proteins natively folded structure. This technique uses a modified version of solvent accessibility as defined by Fred Richards [15]. The accessibility of each residue in a β-sheet was calculated after striping away all but the β-carbons of the side-chains away (using hydrogen in place of a β-carbon for Glycine), and using an empirically derived solvent sphere. In this way, the orientation of the side-chains relative to the protein core, as well as their proximity to the proteins solvent accessible surface could be estimated.

Using the results of this calculation, a β-sheet classification scheme was devised based on the types of accessibility observed in a subset of β-sheets from the data set of true examples. For each pairing of residues i and j in a windowed β-sheet, an assignment was made using their accessibility and the overall accessibility of the individual strand in which they were found. To make the assignments, the *topological access* of a protein was calculated and the values were mapped onto a three-dimensional model and visualized using different ribbon coloring using Midas-Plus [38]. Inspection of 23 different protein structures yielded 194 labeled examples representing each of 11 different clases. The 10 different classes of β-strand pairing are shown in figure 4.2. An 11th miscellaneous class was created for all examples that did not fit the criteria of the other 10 classes. For purposes of training, this scheme had 12 classes, the 12th being the false class.

4.3.5 Neural Network Architectures

Two different experiments were performed using the β-sheet data sets that were constructed. The first experiment involved the use of a single layer neural network and the data set of 207 proteins containing β-sheets. The second experiment used a two layer neural network and the enlarged data set of 339 proteins.

Both neural networks used a feed-forward, error back-propogation architecture, with a normalized exponential, or softmax, activation function at the output nodes, a maximum likelihood learning rule, and an adaptive learning rate. The output of the network was the posterior probability that a given example was a true or false β-strand pairing.

Using a softmax activation function, a generalization of the logistic activation function, the probabilities of true and false sum to unity. This is accomplished by

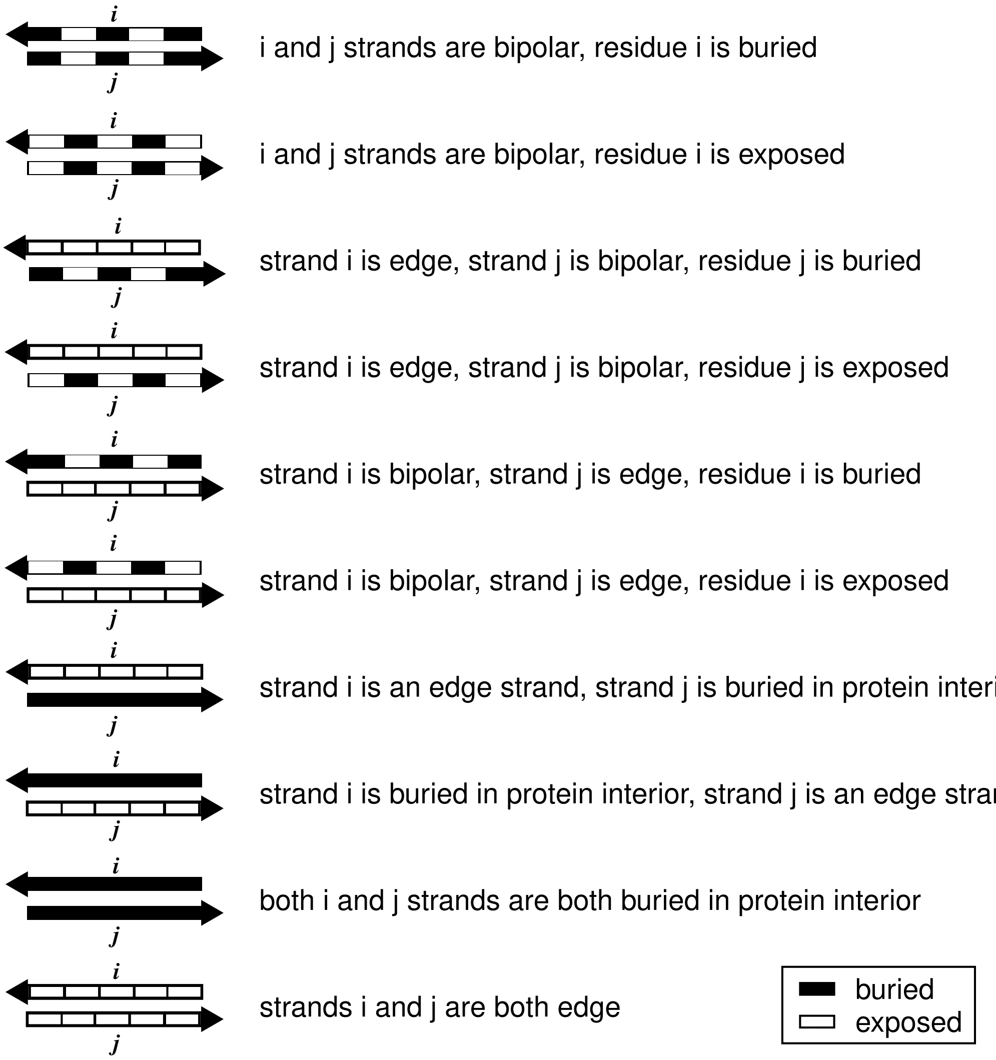


Figure 4.2: The 10 classes of strand pairings possibilites based on an amino acid residues topological solvent accessibility. An 11th miscellaneous class was added for all other pairings of β-strands that did not fit one of these catagories. The assignment of i and j is based on the windowing of figure 4.1

normalizing the an input by all other inputs. The output of the network, o_k,

$$o_k = \frac{e^{a_k}}{\sum_{k'} e^{a_{k'}}} \tag{4.1}$$

is therefore the natural log of the weighted sum of an input, a_k, normalized by the sum of the natural log of all other weighted inputs $a_{k'}$. It can be easily shown that outputs of the network under these conditions can be interpreted as probabilities of an example belonging to a specific class, here true or false, conditioned on the feature inputs.

The error function for both neural networks was a simplified case of maximum likelihood. In this type of learning, the goal is to maximize the likelihood that an example belongs to its target class. This is accomplished by minimizing the negative log of the output likelihood. In this case the negative average of the sum of the log of the ratio of the conditional probability of an example x_i belonging to the true class, and the unconditional density, is calculated as

$$-\frac{1}{N}\sum log\frac{P(true|x_i)}{P(true)} \tag{4.2}$$

In a process analogous to gradient descent learning, the resulting error is then back-propogated through the net and weights are undated accordingly. An adaptive learning rate was calculated based on the difference in error between consecutive epochs. This helps attenuate the magnitude of the change made to the weights to prevent destructive harmonics in the minimization process as well as shorten training times.

The two layer network used in subsequent experiments utilized a competitive learning architecture using a learning rule as described for the single layer network. In this type of network topology, classification of examples by a particular output node is encouraged by updating the weights connected to that output node with the highest probability of the example belonging to the target class.

Due to the time needed to classify all of our true examples, a training heuristic known as *boot-strapping* was employed in the learning process. In this case, learning was accomplished in a two step process. First the network was presented with all true examples that had been classified as belonging to a particular β-sheet type according to the scheme in figure 4.2. Second the network was presented with true examples that were not classified. The network then, based on training on true classified examples, classified the unknown trues. The class membership derived by the network was then assigned to each example used in all subsequent training epochs. In this way, the network was given enough information to classify β-sheet membership, then allowed to classify the remaining unknown trues on its own.

4.4 Results

The results of classifying whether a pair of anti-parallel β-strands were a true contact, using a variety of different features as input to the network, are shown in table 4.3. Using all 12 original properties, solvent accessibility, strand membership,

Properties	Accessibility	Strand Membership	Chain Separation	Number of Inputs	Accuracy %
X	X	X	X	151	87.45
		X	X	11	86.73
X		X	X	131	86.45
X	X	X		150	81.44
X			X	121	80.74
X		X		130	80.72
			X	10	80.53
		X		10	76.10
X				120	63.32
X	X			140	62.11

Table 4.3: Accuracy of network to correctly classify a β-stand contact as true or false using a variety of inputs

and chain separation yielded the best results for this series of experiments with an accuracy of up to 87% in correctly discriminating a true contact from a false one. In the absence of strand membership and chain separation data the accuracy fell to 63%. Due to possible biases in these inputs this is probably a better assessment of the network's accuracy for this particular set of experiments. These results were gathered using *splits*, random divisions of the true and false examples into training and test sets. This was repeated many times for each particular experiment to avoid unintended biases in the presentation of examples. The final figure for accuracy using a set of inputs was an average of the results of all splits.

The results of the two layer network used to classify the type of contact, again using a variety of training features, is shown in table 4.4. The best result for this experiment included a subset of 6 amino acid properties and their identities and yielded an accuracy of 62% in correctly classifying examples of anti-parallel β-strands

Inputs	Accuracy (%)
Amino Acid Identity	61.91
Amino Acid Identity plus Properties-Lite	62.84
Polarity	62.06
Properties-lite	59.26

Table 4.4: Percent accuracy in true/false contact discrimination with multiple classes of true contacts.

as belonging to one of 12 different types of contact. The network generally trained well on the labeled true examples used in bootstrapping, but often over trained on the unlabeled examples. The training and test error for false examples was generally weak and varied inversely with the performance of labeled true examples.

4.5 Discussion

Results of the true-false contact classifier are encouraging. By generating a prediction of β-strands in a novel protein sequence, creating all pairing possibilities, and presenting this to network, it could be possible to predict the topology of the β-sheets with reasonable accuracy. Several factors must be considered though in evaluating the performance of the network and the its value as a predictor.

The performance of the network when presented with only amino acid properties is probably a better estimate of the power of the predictive technique. Since information such as accessibility, strand membership and chain separation would only be available as predictions from a novel protein sequence, the error introduced at this stage could significantly degrade the its predictive power. It is clear from the results of the experiments that in the absence of this information the networks accuracy falls dramatically.

The accuracy of the network in the presence of chain separation information might be artificially inflated due to the generation of false examples. Since false examples were generated by randomly pairing β-strands, without accounting for the distribution of true example chain separation, this may have given the network an unintended edge by biasing the false examples with high values for this feature. This could be remedied by preprocessing the false data in such a was as to create a chain separation distribution similar to that of the true examples.

The network focused on strand membership when trained using this feature information. Based on the weights generated as a result of training, the network was presumably finding symmetry in the membership inputs and basing its classification predominantly with this feature rather than augmenting this with other properties. Again, in the case of predicted strand membership information which has inherently lower accuracy at the beginning and ending residues, this could lower the networks prediction accuracy significantly if it were to rely heavily on this as input.

After analysis of the results, and a closer look at training data, it also became clear that the single layer network was attempting to learn an XOR function as described in Chapter 2. This was due in part to the bipolar characteristics of some of the training examples where i and j alternated between polar, or charged, and hydrophobic residues. In one case the network was attempting to learn an example where the hydrophobicity alternated one way, starting with a polar residue at position i for instance, and in another where it alternated when i was hydrophobic. This is analogous to assigning a strand a series of boolean values representing a polar or charged state or not. In one case the vector 1 0 1 0 1 would be presented to the net as true, while the next, also true, would be 0 1 0 1 0. If this was indeed true there is no set of weights that can be arrived at in a single layer network that can solve this problem.

The results of the β-strand classifier experiments, though not as good as the true false discriminator, did give some insight into the strengths and weaknesses of this technique. The number of true examples assigned a particular class membership was probably the biggest limitation of this experiment. Examination of the performance of specific classes indicated those with more labeled members performed better. This is most likely due to better training during boot-strapping and subsequent assignments of unlabeled examples. Also, in the case where the network incorrectly classified an example as false, the highest true probability was frequently the correct class indicating that the network was beginning to recognize class membership. Though labor intensive and susceptible to misclassification, manually adding additional labeled examples could improve the accuracy of the network in this regard.

Examination of the weights following training with polarity alone indicated the network could use this parameter to make reasonable predictions of pairing. This feature may provided the network with enough information to recognize the bipolar nature of β-sheet structure and therefore improve discrimination of true pairings.

Adding amino acid residue identity, though valid in the context of novel predictions, adds enough inputs to begin to present more features than the number of training examples can sample. Known as the *curse of dimensionality*, this creates a situation where there are more free-parameters than training examples. In the case of adding identities as inputs, this adds 10 bit vectors each with 20 values, or 200 values in addition to the 6 property features for each residue.

Bibliography

[1] Bajorath J. Stenkamp R. Aruffo. A. Knowledge-based model building of proteins: concepts and examples. *Protein Science*, 2:1798–1810, 1993.

[2] Chothia C. Lesk A.M. The relation between divergence of sequence and structure in proteins. *EMBO J.*, 5:823–826, 1986.

[3] Hobohm U Sander C. Enlarged representative set of protein structures. *Protein Science*, 3:522–524, 1994.

[4] Holm L. Sander C. Dali: a network tool for protein structure comparison. *Trends in Biochemical Sciences*, 20(11):478–480, 1995.

[5] Kabsch W. Sander C. A dictionary of protein secondary structure. *Biopolymers*, 22:2577–637, 1983.

[6] Lifson S. Sander C. Specific recognition in the tertiary structure of β-sheets of proteins. *Journal of Molecular Biology*, 139:627–639, 1980.

[7] Fischer D. Eisenberg D. Protein fold recognition using sequence-derived predictions. *Protein Science*, 5:947–955, 1996.

[8] Plaxco K.W. Simons K.T. Baker. D. *Journal of Molecular Biology*, 22:985–994, 1998.

[9] Altschul SF Gish W Miller W Myers EW Lipman DJ. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[10] Altschul S.F. Madden T.L. Schaffer A.A. Zhang J. Zhang Z. Miller W. Lipman D.J. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[11] Sayle R.A. Milner-White E.J. RASMOL: biomolecular graphics for all. *Trends in Biochemistry Science*, 20:374, 1995.

[12] Rosenblatt F. *Principles of Neurodynamics: Preceptrons and the Theory of Brain Mechanics.* Spartan, 1962.

[13] Sietsma J. Dow R. J. F. Creating neural networks that generalize. *Neural Networks*, 4(1):67–79, 1991.

[14] Gregoret L.M. Cohen F.E. Novel method for the rapid evaluation of packing in protein structures. *Journal of Molecular Biology*, 211(4):959–974, 1990.

[15] Lee B. Richards F.M. *Journal of Molecular Biology*, 55:379–400, 1971.

[16] Lydia M. Gregoret. *Packing and protein structure.* PhD thesis, University of California San Francisco, 1991.

[17] Gerstein M Hegyi H. The relationship between protein structure and function: a comprehensive survey with application to the yeast genome. *Journal of Molecular Biology*, 288(1):147–64, 1999.

[18] Neurath H. Evolution of proteolytic enzymes. *Science*, 224(4647):350–357, 1984.

[19] M. Hansen, J. Bentz, A. Baucom, and L. Gregoret. DINAMO: A coupled sequence alignment editor/molecular graphics tool for interactive homology modeling of proteins. In *Proceedings of the Third Pacific Symposium on Biocomputing*, pages 106–117. World Scientific Press., 1998.

[20] Tim J.P. Hubbard. Use of β-strand interaction pseudo-potentials in protein structure prediction and modeling. In *Proceedings of the 27th Annual Hawaii International Conference on System Sciences*, pages 336–344, 1994.

[21] Henikoff S. Henikoff J.G. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America*, 89:10915–10919, 1992.

[22] Rumelhart D.E McClelland J.L. *Parallel Distributed Processing: Explorations in the microcstructure of cognition.* MIT Press, 1986.

[23] Jones D.T. Taylor W.R. Thornton J.M. A new approach to protein fold recognition. *Nature*, 358:86–89, 1992.

[24] Miller R.T. Jones D.T. Thornton J.M. Protein fold recognition by sequence threading: tools and assessment techniques. *FASEB*, 10:171–178, 1996.

[25] Nelsoney C.L. Kelly J.W. Progress towards understanding β-sheet structure. *Bioinorganic and Medicinal Chemistry*, 4(6):739–766, 1996.

[26] Bentz J. Baucom A. Hansen M. Gregoret L. DINAMO: Interactive protein alignment and model building. *Bioinformatics*, 15(4):309–316, 1999.

[27] Smith C.K. Regan L. Guidlines for protein design: the energetics of β-sheet side-chain interactions. *Science*, 270:980–982, 1995.

[28] Blum E.K Li L.K. Approximation theory and feedforward networks. *Neural Networks*, 4(4):511–515, 1991.

[29] Jones L.K. Constructive approximations for neural networks by sigmoidal functions. *Proceedings of the IEEE*, 78(8):1586–1589, 1991.

[30] Stawiski E.W. Baucom A.E. Lohr S.C. Gregoret L.M. Predicting protein function from structure: unique structural features of proteases. *Proceedings of the National Academy of Sciences of the United States of America*, 97(8):3954–3958, 2000.

[31] Bernstein F.C. Koetzle T.F. Williams G.J. Meyer E.E Brice M.D. Rodgers R.J. Kennard O. Shimanouchi T. Tasumi M. The Protein Data Bank: a computer-based archival file for macromolecular structures. *Journal of Molecular Biology*, 112:535–542, 1977.

[32] Gerstein M. *Acta Crystallagraphica*, 48:271–276, 1992.

[33] Hutchinson E. G. Thornton J. M. A revised set of potentials for β-turn formation in proteins. *Protein Science*, 3:2207–2216, 1994.

[34] Guex N. Peitsch M.C. SWISS-MODEL and the Swiss-PdbViewer: and environment for comparative protein modeling. *Electrophoresis*, 18:2714–2723, 1997.

[35] Widrow B. Hoff M.E. Adaptive switching circuits. *WESCON Convention Record*, 4:96–104, 1960.

[36] Rost B. Saqi M.A. Sayle R.A. Bates P.A. Sternberg M.J. Recognition of analogous and homologous protein folds: analysis of sequence and structure conservation. *Journal of Molecular Biology*, 269:423–439, 1997.

[37] Werbos P.J. *Beyond regression: new tools for prediction and analysis*. PhD thesis, Harvard University, Cambridge, MA, 1974.

[38] Ferrin T.E. Huang C.C. Jarvis L.E. Langridge R. The MIDAS display system. *Journal of Molecular Graphics*, 6:13–37, 1988.

[39] Sayle R.A. Chemscape Chime. World Wide Web, 2000. http://www.mdli.com.

[40] Pauling L. Corey R.B. *Proceedings of the National Academy of Sciences of the United States of America*, 37:729, 1951.

[41] B. Rost. PHD: predicting one-dimensional protein structure by profile-based neural networks. *Methods in Enzymology*, 2666:525–39, 1996.

[42] Minsky M.L. Papert S.A. *Perceptrons.* MIT Press, 1969.

[43] Krogh A. Riis S.K. *Predictions of β-sheets in proteins.* MIT Press, 1996.

[44] Navaratnam N. *et al. Escherichia coli* cytidine deaminase provides a molecular model for ApoB RNA editing and a mechanism for rna substrate recognition. *Journal of Molecular Biology*, 275:695–714, 1998.

[45] Thompson J.D. Higgins D.G. Gibson T.J. CLUSTAL W: improving the sensivity of progressive multiple sequence alignments through sequence weighting, position specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

[46] Moro S. Guo D. Camaioni E. Boyer J.L. Harden T.K. Human P2Y1 receptor: molecular modeling and site-directed mutagenesis as tools to identify agonist and antagonist recognition sites. *Journal Medical Chemistry*, 41:1456–1466, 1998.

[47] Sali A. Potterton L. Yuan F. van Vlijmen H. Karplus M. Evaluation of comparative protein modeling by MODELLER. *Proteins*, 23:318–326, 1995.

[48] McCulloch W.S. Pitts W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathmatical Biophysics*, 5:115–133, 1943.

[49] T. Webster, R. Nambudripad, and L. Buturovic, 1997. Personal communication with members of the Temple Smith laboratory, Boston University.

[50] A. Zemla, C. Venclovas, J. Moult, and K. Fidelis. Processing and analysis of CASP3 protein structure predictions. *Proteins: Structure, Function, and Genetics*, Supplement 3(1):22–29, 1999.