Machine Learning Research Group Working Paper 91-2

Refining Algorithms with Knowledge-Based Neural Networks:

Improving the Chou-Fasman Algorithm for Protein Folding[*]

Richard Maclin
Jude W. Shavlik

Computer Sciences Dept.
University of Wisconsin
1210 W. Dayton St.
Madison, WI 53706
email: maclin@cs.wisc.edu

ABSTRACT

We describe a method for using machine learning to refine algorithms represented as generalized finite-state automata. The knowledge in an automaton is translated into a corresponding artificial neural network, and then refined by applying backpropagation to a set of examples. Our technique for translating an automaton into a network extends the KBANN algorithm, a system that translates a set of propositional, non-recursive rules into a corresponding neural network. The topology and weights of the neural network are set by KBANN so that the network represents the knowledge in the rules. We present the extended system, FSKBANN, which augments the KBANN algorithm to handle finite-state automata. We employ FSKBANN to refine the Chou-Fasman algorithm, a method for predicting how globular proteins fold. The Chou-Fasman algorithm cannot be elegantly formalized using non-recursive rules, but can be concisely described as a finite-state automaton. Empirical evidence shows that the refined algorithm FSKBANN produces is statistically significantly more accurate than both the original Chou-Fasman algorithm and a neural network trained using the standard approach. We also provide extensive statistics on the type of errors each of the three approaches makes and discuss the need for better definitions of solution quality for the protein-folding problem.

1. INTRODUCTION

When addressing an unsolved problem, a normal approach is to start by researching the problem and seeing which approaches have been taken. The direction one takes often builds on earlier work. This strategy is generally ignored by most computerized empirical learning systems — systems that try to learn new concepts. Empirical learning systems often try to acquire a concept from scratch, ignoring knowledge that already exists about a problem. Ignoring this existing knowledge is dangerous, since the resulting learned concept may not contain important factors already identified in earlier work. The learning system may even be unable to solve the problem without the headstart prior knowledge would give it. The approach we present allows an empirical learning system to take advantage of prior knowledge about a

[*] A version of this report will appear as a chapter in *Computational Learning Theory and Natural Learning Systems*, S. Hanson, G. Drastal, and R. Rivest, editors, MIT Press, 1992.

problem in a systematic way. Our learning system takes the initial knowledge, and rather than attempting to learn the whole concept from scratch, refines the initial knowledge to better address the problem.

Our work extends the KBANN system (for Knowledge-Based Artificial Neural Networks) [Towell90]. We use the extended system to refine the Chou-Fasman algorithm [Chou78] for predicting the folded structure of globular proteins, a particularly difficult problem in molecular biology. KBANN uses knowledge represented as simple, propositional-calculus rules (referred to as a *domain theory*) to create an initial neural network that represents the knowledge contained in the rules. This network will hopefully be a better starting point for learning than a randomly-configured initial network. This technique has proven effective for complex problems such as gene recognition [Noordewier91, Towell90], even when the original domain theory is not particularly good at solving the problem. The main limitation of the KBANN technique is that it can only translate rules that are propositional (i.e. have no variables) and are non-recursive. This paper describes an addition to KBANN that extends it for a simple type of recursion.

We extend KBANN to include a simple form of recursion by representing *state* information in the neural network. State in the network represents the context of the problem. For example, if the problem is how to cross a room, the state variables may represent (among other things), whether or not the light is on in the room. The rules introduced to solve this problem can therefore take into account the state of the problem — rules to turn on the light would only be considered when the state of the problem indicated that the light is off. In this style of problem solving, the problem is not solved in one step, but instead as a series of actions, each leading to a new state, that hopefully leads to the goal state (turning on the light, navigating to the couch, moving Cindy's toy fire engine out of the way, etc.). The network similarly attempts to determine the resulting next state given the current state and input. This approach is recursive because the state calculated is used in the next step as the previous state for the network. The extended KBANN system, called Finite-State KBANN (FSKBANN), translates domain theories that use state information, usually represented as generalized finite-state automata [Hopcroft79]. As in KBANN, FSKBANN first translates the domain theory into a neural network, and then refines the network using backpropagation [Rumelhart86] with a set of examples.

The choice of neural networks as the empirical learning system on which to build was made for a couple of reasons. One basic reason is that networks provide a very general mechanism for representing concepts. A neural network, given the proper number of hidden units and hidden layers, can learn almost any type of concept [Hornik89]. A second reason for using neural networks is that they generally deal very well with noisy and incorrect data [Shavlik91]. As for limitations of neural networks, one basic problem is how to go about selecting the topology of the network. Determining the best topology for a neural network is problem dependent and is often done by experimentation [Weiss89], which can be a long and costly process. Another problem is deciding what to input to the network. This choice can greatly affect a neural network, since if not enough features are provided, the network may be unable to learn the concept, but if too many are provided the network may be unable to determine which features are critical. Both of these problems are handled to some extent by the KBANN approach. The rules in the domain theory not only determine the initial topology for the network, but are also used to tell the network which variables to focus on initially.

We chose the problem of protein secondary-structure prediction (defined in section 3) for a number of reasons. The secondary-structure prediction problem is an open problem which will become increasingly critical as the Human Genome Project [Watson90] continues to produce more data. The problem is attractive because there exist a number of algorithms, proposed by biological researchers, which attempt to solve this problem, but with only limited success. The Chou-Fasman algorithm [Chou78] is the focus of our work because it is one of the best-known and widely-used algorithms in the field. This algorithm cannot be represented using the original KBANN system, but can be represented for FSKBANN. The secondary-structure problem is also of interest because a number of researchers have applied standard neural networks to this problem [Holley89, Qian88]. Our work combines these two approaches to achieve a more accurate result than either neural networks or the Chou-Fasman algorithm does alone; Table 1 illustrates a broad overview of our task.

Table 1. An overview of our approach to the protein-folding problem.

Given:	An algorithm for predicting secondary structure (e.g., the Chou-Fasman algorithm [Chou78]).
	and
	Sample proteins with known secondary structure from the Brookhaven X-ray Crystallography Database [Bernstein77].
Do:	Refine the initial algorithm using sample proteins.
Predict:	The secondary structure of new proteins.

This paper presents and empirically analyzes the FSKBANN algorithm for problem solving in domains where prior knowledge exists. The next section presents the basic KBANN algorithm and discusses how we extended the algorithm to handle state information. Section 3 contains a description of the protein-folding problem, including a discussion of a number of approaches that have been taken. Following that are a number of experiments we performed to test the utility of the FSKBANN approach for this problem. The fourth section describes some enhancements to our approach which we are currently exploring. We conclude with a review of work in related fields, specifically other work that attempts to solve the protein-structure problem, network methods that use state information, and other algorithms that refine domain theories.

2. THE KBANN ALGORITHM

The KBANN algorithm [Towell90] translates a domain theory represented as simple rules into a promising initial neural network; it uses the rules to define a network topology and also to initialize the weights of the network. This section outlines standard KBANN and gives a simple example of how the algorithm works. This is followed by a description of an extension to the standard algorithm for domain theories represented as finite-state automata (FSAs).

2.1. Standard KBANN

KBANN translates domain theories represented as propositional, non-recursive rules. An overview of the algorithm appears in Table 2. The algorithm takes as input a set of rules along with a specification of which propositions are to be used as input and output units. KBANN produces a network where each proposition has a corresponding unit that is highly active when the proposition is true and inactive when the proposition is false. The process of assigning propositions to units is described below in Step 1. After it creates the initial network, KBANN adds extra links with low weights to the network, as described in Step 2. In Step 3, KBANN trains the network using backpropagation, thereby refining the original domain theory.

Table 2. Overview of the KBANN algorithm.

Step 1.	Translate the domain theory into a neural network.
Step 2.	Add extra links and input units to the network.
Step 3.	Train the network using backpropagation.

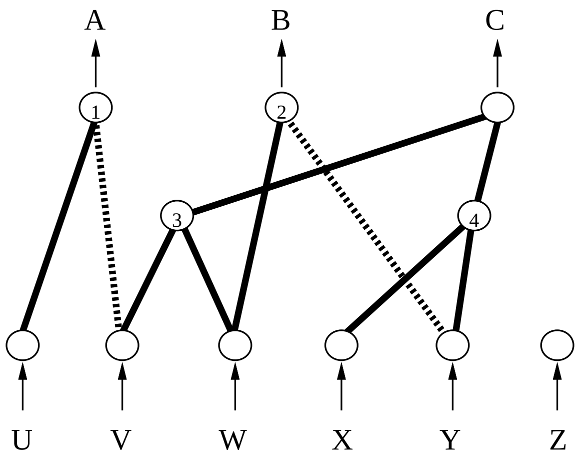
Step 1. Setting the Initial Topology and Weights

The rules in the domain theory determine the topology of the network. KBANN represents each rule by a separate unit. For each antecedent of a rule, KBANN adds a connection in the network from the unit representing the antecedent to the unit representing the rule's consequent. The weight of the connection is equal to a constant, ω (generally 4.0), for positive antecedents and $-\omega$ for negated antecedents. The bias of the new unit is $(n - \frac{1}{2})\omega$ where n is the number of positive antecedents in the rule being translated. The resulting unit is active only when all of the positive antecedents are on and all of the negative antecedents are off.

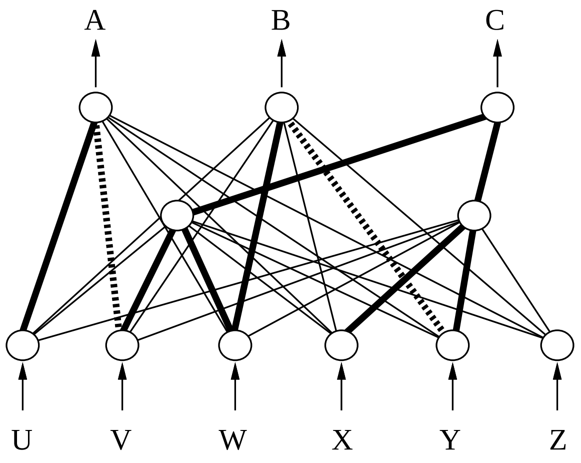
As an example, consider translating rule 3 in Figure 1a. KBANN creates the unit marked 3 in Figure 1b, which has connections from the input units V and W with weights of ω. The bias of this unit is set to $\frac{3}{2}\omega$. Each of the units corresponding to one of Figure 1a's rules is marked with the rule number in Figure 1b. Links with weight ω appear as solid lines and links with weight $-\omega$ are dashed.

(1)	A	$\leftarrow$	$U \wedge \neg V$
(2)	B	$\leftarrow$	$W \wedge \neg Y$
(3)	C	$\leftarrow$	$V \wedge W$
(4)	C	$\leftarrow$	$X \wedge Y$

(a)



(b)



(c)

Figure 1. Example of the KBANN process: (a) a set of sample rules, (b) the neural network representing the rules, and (c) the network after adding extra links.

The units in Figure 1b representing rules 1 and 2 from Figure 1a are the units that represent the propositions A and B, respectively. However, C is represented by a unit that does not correspond to a rule. When a proposition is the consequent of only one rule (as A and B are), then the unit representing that rule directly corresponds to that proposition; but when more than one rule imply a proposition (as with C), that proposition must be handled slightly differently. In this latter case (an implicit disjunction), KBANN generates a unit for each rule. It then generates a new unit to represent their disjunction. Each of the units representing rules are connected to the unit representing the consequent with connections of weight ω. The bias of the unit representing the consequent is set to ½ ω. Thus, in the example in Figure 1, the unit representing C is active when *either* of the units representing rules 3 and 4 are active.

Step 2. Adding Extra Connections

Once the initial topology and initial weights are set, KBANN adds lowly-weighted connections to the network. These connections allow the network to add antecedents to rules during the learning phase. The basic algorithm for connecting units is to connect every unit at level *n* to every unit at level *n-1* (unless there is a connection already). KBANN determines the level of a unit recursively: a unit's level is one higher than the level of the highest unit connected to it. In Figure 1b, assuming the input units are all at level zero, the units labeled 1-4 are at level one, and the unit corresponding to proposition C is at level two. KBANN then connects each of the level-one units to the level-zero units, and the level-two units to the level-one units (an exception is made for output units; output units are never connected to other output units). The resulting network appears in Figure 1c, with the low-weight connections shown as thin solid lines.

Step 3. Refining the Domain Theory

After setting up the network, KBANN trains the network on a set of examples using backpropagation [Rumelhart86], though any neural network learning algorithm could be used. The domain theory serves as a bias giving the network a (hopefully) good starting point for backpropagation search through weight space.

Evaluation of KBANN

KBANN has proven to be very effective in the *promoter* problem [Towell90] and the *splice-junction* problem [Noordewier91], two tasks from molecular biology. KBANN achieves for the promoter problem an error rate of only 3.8% as compared to an error rate of 7.6% for neural networks alone, and 11.3% for a non-learning solution. For the splice-junction problem, KBANN and standard neural networks have about the same error rate for large numbers of examples, but for small numbers of training examples, KBANN makes less than half the number of errors standard neural networks do.

2.2. Finite-State KBANN

One limitation of KBANN is that its domain theories must be expressed as propositional, non-recursive rules. This makes it impossible to easily represent domain theories such as the Chou-Fasman algorithm, which predicts protein secondary structure. We address this limitation by extending KBANN to represent finite-state automata. The difficulties in representing an FSA in a neural network are (1) representing the state of the automaton, and (2) capturing the notion of "scanning" an input string. FSKBANN addresses both of these issues.

Maintaining Past State

FSKBANN uses a neural-network structure similar to the networks introduced in both [Elman90, Jordan86] to represent state. The network takes as input the previous state and the input values, and determines the next state. It uses this new state as the previous-state input in the next step. This "copying-back" process acts as if there were recurrent links, with fixed weights, from the output units representing the current state to the input units representing the previous state. We chose this representation over a fully-recurrent neural network implementation because of its simplicity. The major difference between our networks and fully-recurrent networks is that no weight adjustment occurs in our network for the links from the output units to the inputs.

Scanning the Input

FSKBANN captures the notion of scanning a series of input values and successively determining the current state by activating the network once for each input value (as in [Sejnowski87]). Each input value is combined with the previously-determined state as input to the network, and the network determines the resulting state. The concept of input value is generalized for our networks. An FSA generally examines a single input value at a time. Instead of requiring a single input value, FSKBANN takes a vector of values representing the current input (i.e. an input value may be represented by many input propositions; for example, the input could contain an amino acid and its 12 neighbors in a protein).

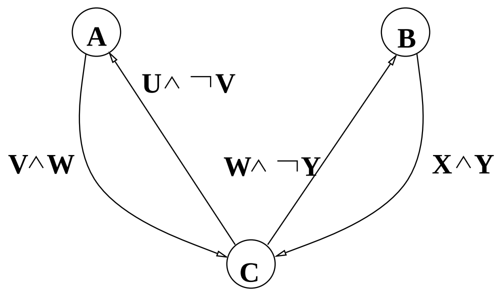
Mapping FSAs to Neural Networks

The final addition needed to the standard KBANN algorithm to handle FSAs is a step that transforms an FSA into a set of rules involving state. FSKBANN translates each of the transition arcs in the FSA into a rule where the antecedents of the rule are the original state plus the predicate with which the arc is labeled, and the consequent of the rule is the resulting state. As an example, consider the arc labeled $U \wedge \neg V$ from state C to state A in the FSA in Figure 2a. FSKBANN translates this arc into rule 1 in Figure 2b. Figure 2b displays the rules derived from the FSA in Figure 2a.

FSKBANN maps these rules into a network where the input units are the input propositions and the previous state; the output of the network represents the resulting state. The network resulting from Figure 2b's rules is shown in Figure 2c. FSKBANN concludes by adding extra links to the network and then training the network.

Algorithmic Details

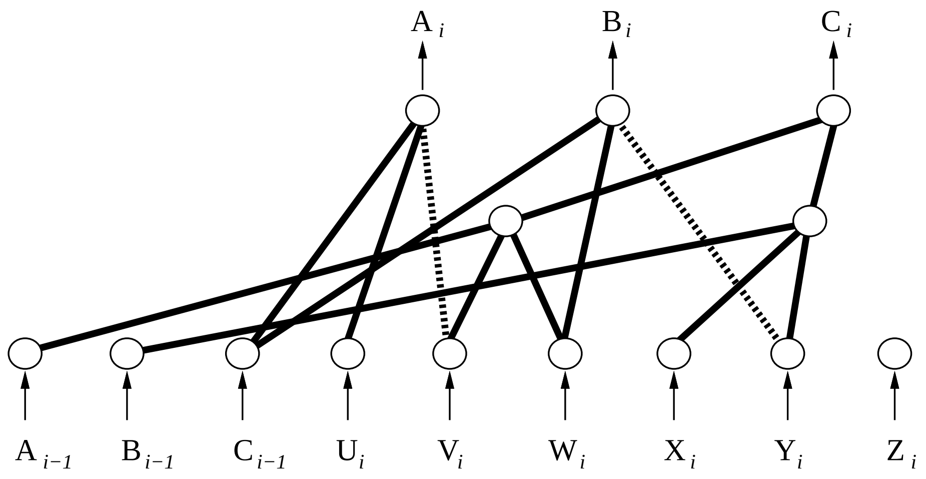
An important consideration in training neural networks with state information is which output to use as the previous state. There are two possibilities: the teacher's (correct) output or the actual output produced by the network? Each approach has its advantages and drawbacks. If the network's output is used, and that output is incorrect, then in the next training step the network will be trying to learn the wrong transition. Using the teacher's output avoids this problem, but it is possible that the network will become dependent on receiving the correct last output as input. This can be a problem in domains where the trained network will not achieve very high accuracy. In this case, using the actual output is an advantage because the network can still learn to use the state information, but will not place too much faith in the correctness of the information. Our experiments for the protein-folding problem (see Section 4) used the



(a)

$$
\begin{array}{llcl}
(1) & A_i & \leftarrow & C_{i-1} \wedge U \wedge \neg V \\
(2) & B_i & \leftarrow & C_{i-1} \wedge W \wedge \neg Y \\
(3) & C_i & \leftarrow & A_{i-1} \wedge V \wedge W \\
(4) & C_i & \leftarrow & B_{i-1} \wedge X \wedge Y
\end{array}
$$

(b)



(c)

Figure 2. Example of the FSKBANN process: (a) a sample finite-state automaton, (b) the rules derived from the FSA, and (c) the initial FSKBANN translation of the FSA.

approach of training with the actual output value because we did not expect to achieve perfect accuracy for our predictions. In preliminary testing, using the teacher's output led to substantially worse results.

Our additions to KBANN extend it to a larger class of problems, by enriching the vocabulary with which the user can express domain theories. In the next section we describe how FSKBANN can represent and refine the Chou-Fasman algorithm for predicting protein secondary structure.

3. THE PROTEIN-FOLDING PROBLEM

Globular proteins are long strings of amino acids, several hundred elements long on average. There are 20 amino acids in all (represented by different capital letters). The string of amino acids making up a given protein constitutes the *primary* structure of the protein. Once a protein forms, it folds into a three-dimensional shape; the spatial location of each of the amino acids in this structure is known as the *tertiary* structure of the protein. Tertiary structure is important because the form of the protein strongly influences its function.

3.1. Predicting Protein Secondary Structure

At present, few means exist to determine the tertiary structure of a protein. These include X-ray crystallography and magnetic resonance processes: both of which are costly and time consuming. An alternative solution is to predict the *secondary* structure of a protein as an approximation to the tertiary structure. The secondary structure of a protein is a description of the local structure for each amino acid. One prevalent system of determining secondary structure divides a protein into three different types of structures: (1) α-helix regions, (2) β-sheet regions, and (3) random coils (all other regions). Figure 3 is a diagram of the tertiary structure of a protein and how the shape is divided into regions of secondary structure.

Table 3 shows a sample mapping between a protein's primary and secondary structures. The basic task is usually defined as follows: given the primary structure of the protein, produce a secondary-structure assignment for each of the amino acids in the protein. In Section 4's

Figure from [Richardson89, pg. 35].

Figure 3. *Ribbon* drawing of the three-dimensional structure of a protein (from [Richardson89, pg. 35]). The areas resembling springs are α-helix structures, the flat arrows represent β-sheets, and the remaining regions are random coils.

experiments, we focus on learning how to perform this task.

Table 3. Primary and secondary structures of a sample protein.

Primary (20 possible amino acids)	P	S	V	F	L	F	P	P	K	P	...
Secondary (three possible local structures)	c	c	β	β	β	β	c	c	c	α	...

Predicting the secondary structure from the primary structure is only one part of the protein-folding problem. Other methods attempt to predict *super-secondary* structures, which are descriptions of combinations of secondary structures [Chothia84]. Another problem is how to combine the information about secondary structure and primary structure (and possibly super-secondary structure) to determine the tertiary structure [Cohen89]. For a more complete review of the protein-folding problem see [Fasman89].

3.2. Approaches to Predicting Secondary Structure Based on Biological Information

A number of different algorithms have been proposed in the biological literature for predicting protein secondary structure. Generally these algorithms focus on predicting the secondary structure for an amino acid using only local information (i.e., information about the string of amino acids immediately before and after it in the protein). The algorithm we focus on is that of Chou and Fasman [Chou78]. Their algorithm is similar for both α-helix and β-sheet prediction, so we will only discuss α-helix prediction. Figure 4 provides an overview of the algorithm. The first step of the process is to find α-helix *nucleation sites*. Nucleation sites are amino acids that are very likely to be part of α-helix structures, according to the conformation probabilities and rules reported by Chou and Fasman. From these sites, their algorithm extends the structure both forward and backward along the protein, as long as the probability of being part of a α-helix structure remains high enough. After both α-helix and β-sheet regions have been predicted, the Chou-Fasman algorithm compares the relative probabilities of regions to resolve predictions that overlap.

Table 4 contains results from the Chou-Fasman and some related non-learning algorithms. Note that in the data sets used to test the algorithms, 54-55% of the amino acids in the proteins are part of coil structures, so 54% accuracy can be achieved trivially by always predicting coil. Another important point is that many biological researchers believe that algorithms which only take into account local information can only achieve limited accuracy [Wilson85] (generally believed to be at most 80% accuracy). Accuracy is limited because the structure can be affected by a portion of the protein that is far away along the protein, but due to three-dimensional folding has come back into proximity. The relatively limited success of these results has led other researchers to try different means of creating prediction algorithms, including neural-network approaches.

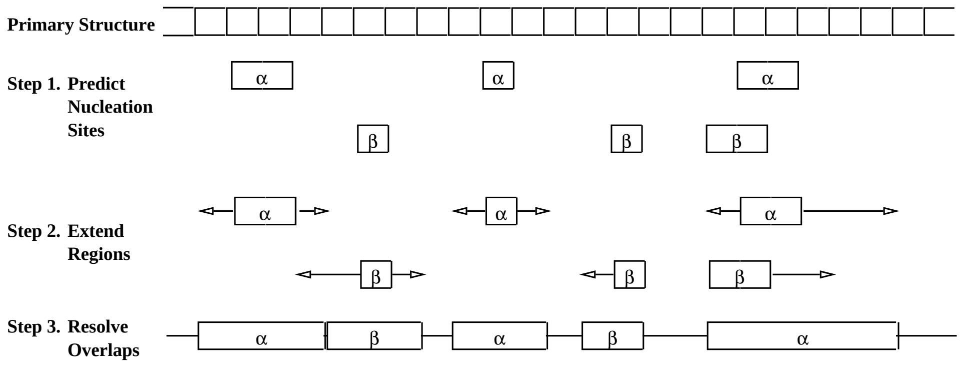


Figure 4. Steps of the Chou-Fasman algorithm.

Table 4. Accuracies of various (non-learning) structure-prediction algorithms.

Method	Accuracy	Comments
Chou-Fasman [Chou78]	48%	original algorithm, data from [Qian88]
Chou-Fasman [Chou78]	58%	reworked version, from [Nishikawa83]
Chou-Fasman [Chou78]	58%	reworked version, data from [Qian88]
Lim [Lim74]	50%	from [Nishikawa83]
Robson et al. [Robson76]	50%	from [Nishikawa83]
Garnier and Robson [Garnier89]	58%	data from [Qian88]

3.3. Neural-Network Approaches

Several researchers have attempted to use neural networks to solve the protein-folding problem [Holley89, Qian88]. The neural networks in these efforts have as input a *window* of amino acids consisting of the central amino acid being predicted, plus some number of the amino acids before and after it in the sequence. The amino acid for each window position is represented by 21 input units (one for each of the amino acids, plus one for "off the end"). The network output, which predicts the structure of the central amino acid, includes a unit for each type of secondary structure, though Holley and Karplus use only two output units for α-helix and β-sheet, predicting coil if neither of these is active. The networks also include some number of hidden units in a single layer between the input and output units (the number was varied in both studies); Figure 5 is a diagram of this type of network. The networks are trained on a set of proteins with known secondary structure and then tested on a separate set of proteins. Table 5 presents results from two studies using this approach. One reason for the differences in the results is that these studies use different sets of proteins for testing. Since the results are still somewhat disappointing, a possible strategy is to combine the neural network approach with the knowledge from the Chou-Fasman algorithm to produce a better network solution. This is our approach.

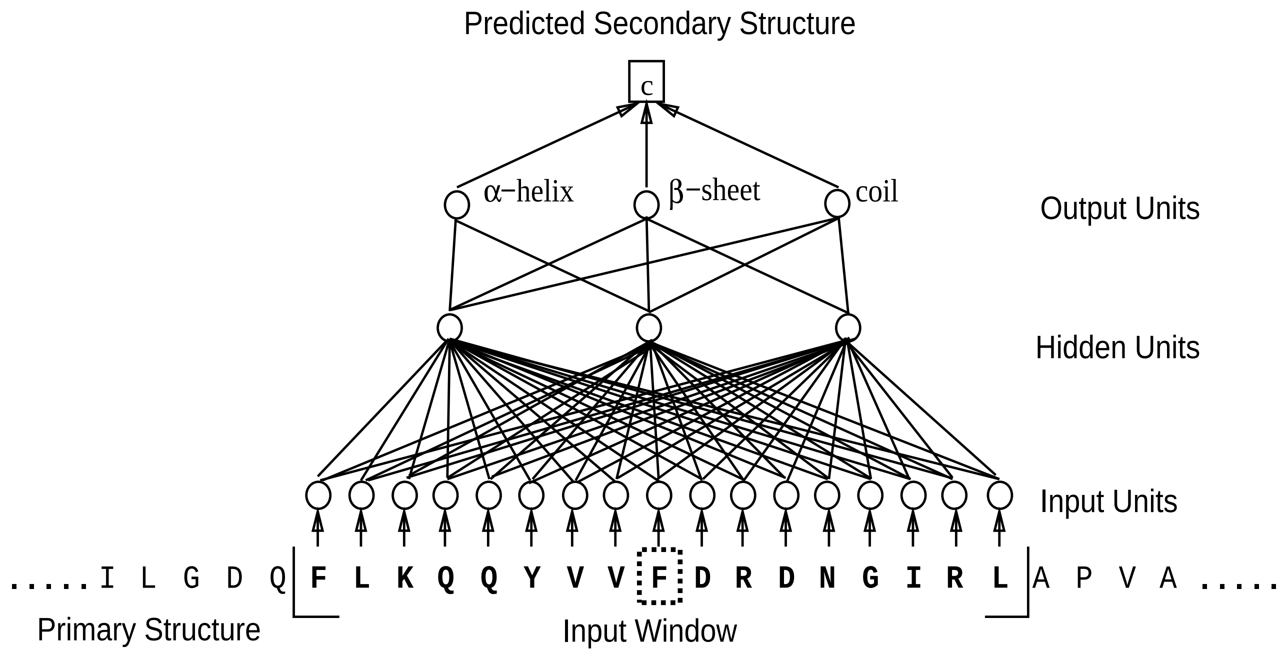

Predicted Secondary Structure
c
α−helix
β−sheet
coil
Output Units
Hidden Units
Input Units
.....I L G D Q F L K Q Q Y V V F D R D N G I R L A P V A .....
Primary Structure
Input Window

Figure 5. General neural network architecture used by Holley & Karplus and Qian & Sejnowski.

Table 5. Neural network results for the secondary-structure prediction task.

Method	Accuracy	Number of Hidden Units	Window Size
Holley & Karplus [Holley89]	63.2%	2	17
Qian & Sejnowski [Qian88]	62.7	40	13

3.4. Representing the Chou-Fasman Algorithm as a Finite-State Automaton

The primary problem in representing the Chou-Fasman algorithm as a set of rules is capturing the notion of extending a region. In a neural network without state information, to know if an α-helix region can be extended to the current window position, the network would have to determine if a nucleation site is predicted just to the left of the window, two steps to the left of the window, three steps to the left of the window, etc. Not only does this require a lot of replication of rules (since the rules to recognize a nucleation site would have to be copied for each window position), but the nucleation site might actually be outside the window. We solved this problem by representing the Chou-Fasman algorithm as an FSA (see Figure 6); once the algorithm goes into state `helix`, it will remain there until `break-helix` is true.

The notion of transition in Figure 6's automata is complex. Each transition is actually a set of rules dependent on the input window and the current state. Table 6 displays some samples of different types of rules derived from the FSA (the full set of rules appear in the Appendix). Not all of the rules involve state information; some of the rules are used to prove propositions that are used in other rules. Rule 1 is an example that does not involve state. It is used to prove the proposition `break-helix`, which is used in rule 3. `Break-helix` is defined in terms of the propositions `helix-break@0`[†] and `helix-break@1`; these propositions are then defined by other propositions (see Appendix). Rule 2 is an example that results in a new state, but does not refer to the previous state. This type of rule occurs when a transition to a state should occur not matter what the previous state. Rule 2 says that no matter what the previous state is, if `init-helix` is true, the resulting state should be `helix`. Rule 3 is an example where a transition is made from one state to another. It says that the automaton should transition from state `helix` to state `coil` if `break-helix` is true.

[†]The "@" in these propositions refers to a window position. For example, `Helix-break@0` means that the proposition `helix-break` is true about the amino acid at window position 0 (the central amino acid).

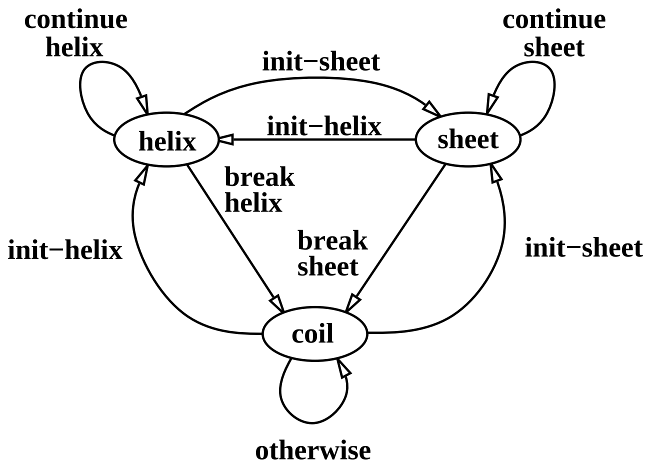


Figure 6. The finite-state automaton used to represent the Chou-Fasman algorithm.

Table 6. Sample rules from the Chou-Fasman finite-state automaton.

(1)	break-helix	$\leftarrow$	helix-break@0 $\wedge$ helix-break@1
(2)	helix_i	$\leftarrow$	init-helix
(3)	coil_i	$\leftarrow$	$\text{helix}_{i\text{-}1} \wedge$ break-helix

Figure 7 is a diagram of the type of network we used to represent the Chou-Fasman domain theory. One important aspect to consider about our approach is that the FSA only extends the structure in a single direction as it scans the primary structure. To extend the structure in both directions, our network actually scans the primary structure in both directions and averages the predictions.

4. EXPERIMENTAL STUDY

We performed a number of experiments on the protein structure-prediction problem to evaluate FSKBANN. Our study demonstrates that FSKBANN has a small but statistically-significant gain in accuracy over both standard artificial neural networks (ANNs) and over a non-learning version of the Chou-Fasman algorithm.

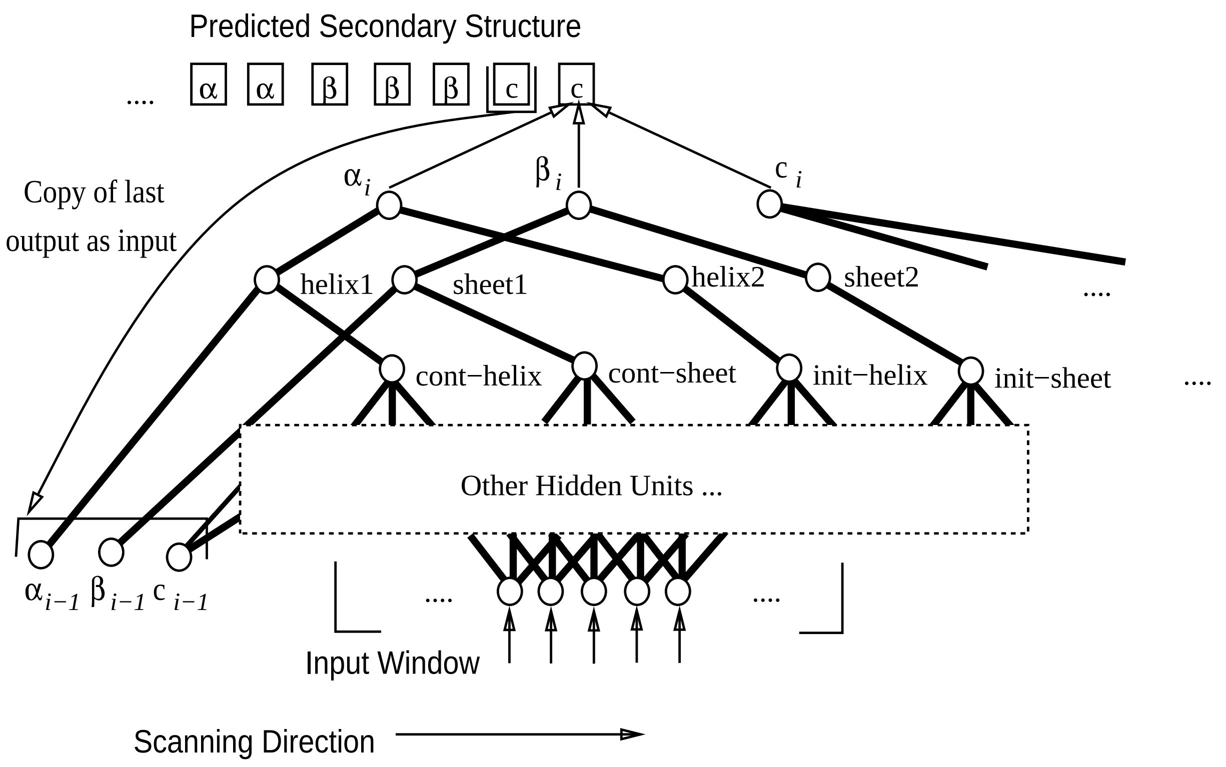


Figure 7. General neural-network architecture used to represent the Chou-Fasman algorithm.

This section describes the data used in our experiments and our training method. Following that, we present and analyze a number of experiments. The experiments evaluate: (1) the effectiveness of the FSKBANN approach versus standard ANNs and the Chou-Fasman algorithm, (2) the value of using state information in ANNs, and (3) performance on test-set proteins as a function of the number of training examples. We conclude with an in-depth empirical analysis of the strengths and weaknesses of the three different methods.

4.1. Experimental Details

We performed our experiments using the data set from Qian and Sejnowski [Qian88]. Their data set consists of 128 proteins with a total of 21,623 amino acids, for an average length of 169 amino acids per protein. Of these amino acids, 54.5% are part of coil structures, 25.2% part of α-helix structures, and 20.3% part of β-sheet structures. We divided the proteins randomly into training and test sets containing two-thirds (85 proteins) and one-third (43 proteins) of the original proteins, respectively.

We used backpropagation [Rumelhart86] to train the neural networks in the two network approaches (FSKBANN and standard ANNs). *Patience* [Fahlman90] was our criterion for stopping learning on the training set. The patience criterion states that training should continue until the error rate has not decreased for some number of training cycles. For this study we set the number of epochs to be four. We use patience rather that some minimum accuracy on the training set because the noisiness of the data set prevents anything close to perfect learning on the training set.

During training, we divided the proteins into two portions — a training set and a *tuning* set [Lang90, pg. 41-42]. We employ the training set to train the network and the tuning set to estimate the generalization of the network. For each epoch, the system trains the network on each of the amino acids in the training set; it then assesses accuracy on the tuning set. We retain the set of weights achieving the highest accuracy for the tuning set and use this set of weights to measure test set accuracy. The system randomly chooses a "representative" tuning set; it considers a tuning set representative if the percentages of each type of structure in the tuning set roughly approximate the percentages of all the training proteins (i.e. the percentages of α-helix, β-sheet and random coil structures in the tuning set should be close to the percentages present in the training proteins). The system does not consider the testing set when comparing the percentages. It chooses the tuning set by randomly picking a set of proteins and then evaluating their "representativeness" and repeating the process until a tuning set is chosen that meets the criterion. Through empirical testing (not reported here), we found that a tuning set size of five proteins achieved the best results for both FSKBANN and ANNs. It is important to consider that this style of training is different than that reported by Qian and Sejnowski. They tested their network periodically and retained the network that achieved the highest accuracy for the test set.

Our experiment-running system trains the FSKBANN networks twice for each protein — scanning forward and then in reverse over the protein. It then averages the two predictions for each amino acid. This simulates extending secondary structure in both directions. The system trained the ANNs both ways — scanning both directions, and scanning only one direction (as in [Qian88]). We report results for ANNs of scanning only one direction, because these results are slightly superior to the results for scanning both directions. Our ANNs use the same number of hidden units that we used in the FSKBANN networks (28) to represent the domain theory. We use 28 hidden units rather than the 40 suggested by Qian and Sejnowski for two reasons. One, we wished to hold the network size constant when comparing FSKBANN and standard ANNs. Two, Qian and Sejnowski found only minor differences in accuracy when the number of hidden units ranged from zero to 40.

A final item to note is that our system does some post-processing of the network output (as suggested in [Holley89]). For each amino acid the system predicts the secondary structure by choosing the output unit with the highest activation. The system then eliminates any "short" sequences. Short sequences are sequences of α-helix or β-sheet predictions that are less than four or two amino acids in length, respectively. Since such short sequences do not in general occur in real proteins [Kabsch83], the prediction for these sequences is replaced with coil.

4.2. Results and Analysis

Table 7 contains results averaged over the 10 test sets. The statistics reported are the percent accuracy overall, the percent accuracy by secondary structure, and the correlation coefficients for each secondary structure. The correlation coefficient [Mathews75] is defined by the formula:

$$C_S = \frac{TP_S TN_S - FP_S FN_S}{\sqrt{(TP_S + FP_S)(TP_S + FN_S)(TN_S + FP_S)(TN_S + FN_S)}}$$

In this formula, s is the secondary structure whose coefficient is being calculated, and TP_s, TN_s, FP_s, and FN_s are the number of true positives, true negatives, false positives, and false negatives for that structure, respectively. The correlation coefficient is good for evaluating the effectiveness of the prediction for each of the classes of secondary structure separately. The resulting gain in overall accuracy for FSKBANN over both ANNs and the non-learning Chou-Fasman method is statistically significant at the 0.5% level (i.e. with 99.5% confidence) using a *t*-test.

Table 7. Comparison of FSKBANN, ANN, and the non-learning Chou-Fasman algorithm.

	Accuracy				Correlation Coefficients		
Method	Total	Helix	Sheet	Coil	Helix	Sheet	Coil
Chou-Fasman	57.3%	31.7%	36.9%	76.1%	0.24	0.23	0.26
ANN	61.8	43.6	18.6	86.3	0.35	0.25	0.31
FSKBANN	63.4	45.9	35.1	81.9	0.37	0.33	0.35

The apparent gain in accuracy for FSKBANN over ANN networks appears fairly small (only 1.6 percentage points), but this number is somewhat misleading. The correlation coefficients give a more accurate picture. They show that the FSKBANN does better on both α-helix and coil prediction, and much better on β-sheet prediction. The reason that the ANN solution still does fairly well in overall accuracy is that it predicts a large number of coil structures (the largest class) and does very well on these predictions.

The gain in accuracy for FSKBANN over the Chou-Fasman algorithm is fairly large and exhibits a corresponding gain in all three correlation coefficients. It is interesting to note that the FSKBANN and Chou-Fasman solutions produce approximately the same accuracy for β-sheets, but the correlation coefficient demonstrate that the Chou-Fasman algorithm achieves this accuracy by predicting a much larger number of β-sheets.

To evaluate the usefulness of the domain theory as the number of training instances decreases and also to estimate the value of collecting more proteins, we performed a second series of tests. We divided each of the training sets into four subsets: the first contained the first 10 of the 85 proteins; the second contained the first 25; the third contained the first 50; and the fourth had all 85 training proteins. This process produced 40 training sets. Each of these training sets was then used to train both the FSKBANN and ANN networks. Figure 8 contains the results of these tests. FSKBANN shows a gain in accuracy for each training set size (statistically significant at the 5% level, i.e. 95% confidence).

The results in Figure 8 demonstrate a couple of interesting trends. One, the FSKBANN networks do better no matter how large the training set, and two, the shape of the curve indicates

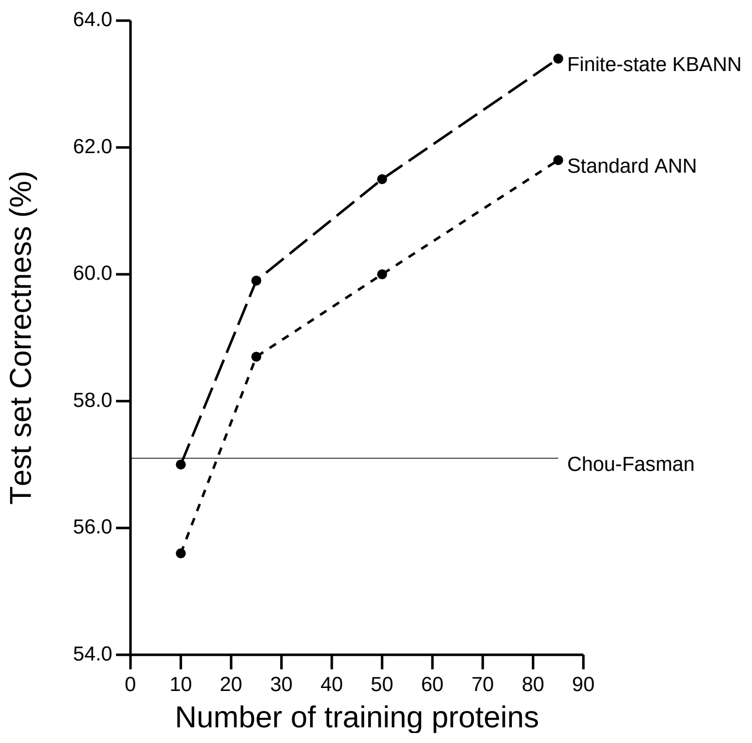


Figure 8. Percent correctness on test proteins as a function of training-set size.

that accuracy might continue to increase if more proteins were used for training. The one anomaly for this curve is that the gain in accuracy for the 10 training proteins is not very large. One would expect that when the number of training instances is very small, the domain knowledge would be a big advantage. The problem here is that for a small training set it is possible to obtain random sets of proteins that are not very indicative of the overall population. Individual proteins generally do not reflect the overall distribution of secondary structures for the whole population; many proteins have large numbers of α-helix regions and almost no β-sheets, and others have large numbers of β-sheet regions and almost no α-helices. Thus in trying to learn to predict a very skewed population, the network may produce a poor solution. This is mitigated as more proteins are introduced, causing the training population to more closely match the overall population.

FSKBANN networks are different from standard ANNs in two ways: (1) their topology is determined by the domain theory, and (2) they copy the output of the network back as the state of the network in the next step. To evaluate the utility of the second feature alone, we trained ANNs similar to those in Qian and Sejnowski's work, but added to these networks state information (i.e. the output was copied back as part of the input for each step). Table 8 compares results from these tests with results from using standard ANNs alone. These results show that state information alone is not enough to increase the accuracy of the network prediction.

Table 8. Effect of state variables on the standard neural-network approach.

	Accuracy				Correlation Coefficients		
Method	Total	Helix	Sheet	Coil	Helix	Sheet	Coil
ANN	61.8%	43.6%	18.6%	86.3%	0.35	0.25	0.31
ANN (with state)	61.7	39.2	24.2	86.0	0.32	0.28	0.31

Finally, to analyze the detailed performance of the various approaches, we gathered a number of additional statistics concerning the FSKBANN, ANN, and Chou-Fasman solutions. These statistics analyze the results in terms of *regions*. A *region* is a consecutive sequence of amino acids with the same secondary structure. We consider regions because the measure of accuracy obtained by comparing the prediction for each amino acid does not adequately capture the notion of secondary structure as biologists view it [Cohen91]. For biologists, knowing the number of regions and the approximate order of the regions is nearly as important as knowing exactly the structure within which each amino acid lies. Consider the two predictions in Figure 9. The first prediction misses completely the third α-helix region, so it has four errors. The second prediction is slightly skewed for each α-helix region and ends up having six errors, even though it appears to be a better answer. The statistics we have gathered try to assess how well each solution does on predicting α-helix regions (see Table 9) and β-sheet regions (see Table 10).

The results of Table 9 and Table 10 give a picture of the strengths and weakness of each approach. Table 9 shows that the FSKBANN solution overlaps slightly fewer actual α-helix regions than the ANNs, but that these overlaps tend to be somewhat longer. On the other hand, the FSKBANN networks *overpredict* fewer regions than ANNs (i.e. predict fewer α-helix regions

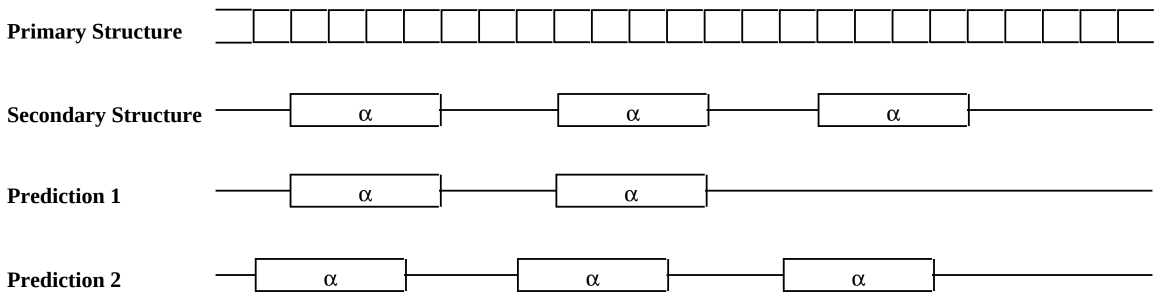


Figure 9. Two possible predictions for secondary structure.

Table 9. Region-oriented statistics for α-helix prediction.

Occurrence		Description	FSKBANN	ANN	Chou−Fasman
Actual	α−helix	Average length of an actual helix region (number of regions).	10.17 (1825)	10.17 (1825)	10.17 (1825)
Predicted	α−helix	Average length of a predicted helix region (number of regions).	8.52 (1774)	7.79 (2067)	8.00 (1491)
Actual Predicted	α−helix α−helix	Percentage of time an actual helix region is overlapped by a predicted helix region (length of overlap).	67% (6.99)	70% (6.34)	56% (5.76)
Actual Predicted	other α−helix	Percentage of time a predicted helix region does not overlap an actual helix region.	34%	39%	36%

that do not correspond to an actual α-helix region). Table 9 also indicates that the FSKBANN and ANN networks more accurately predict the occurrence of regions than the Chou-Fasman algorithm does.

Table 10 demonstrates that FSKBANN's predictions overlap a much higher percentage of actual β-sheet regions than either the Chou-Fasman algorithm or ANNs alone. The overall accuracy for β-sheet predictions is approximately the same for FSKBANN and the Chou-Fasman method, because the length of overlap for the Chou-Fasman method is much longer than for FSKBANN (at the cost of predicting much longer regions). The ANN networks do extremely poorly at predicting overlapping actual β-sheet regions. The FSKBANN networks do as well as the ANNs at not overpredicting β-sheets, and both do better than the Chou-Fasman method. Taken together, these results indicate that the FSKBANN solution does significantly better than the ANN solution on predicting β-sheet regions without having to sacrifice much accuracy in predicting α-helix regions.

Finally, the results indicate that the FSKBANN solution does a much better job of avoiding the problem of overpredicting coil regions than the ANN solution. The results also suggest that more work needs to be done on developing methods of evaluating solution quality, so that solutions that accurately predict all three classes are favored over solutions that only do well at predicting the largest class.

Table 10. Region-oriented statistics for β-sheet prediction.

Occurrence		Description	FSKBANN	ANN	Chou−Fasman
Actual	β−sheet	Average length of an actual sheet region (number of regions).	5.00 (3015)	5.00 (3015)	5.00 (3015)
Predicted	β−sheet	Average length of a predicted sheet region (number of regions).	3.80 (2545)	2.83 (1673)	6.02 (2339)
Actual Predicted	β−sheet β−sheet	Percentage of time an actual sheet region is overlapped by a predicted sheet region (length of overlap).	54% (3.23)	35% (2.65)	46% (4.01)
Actual Predicted	other β−sheet	Percentage of time a predicted sheet region does not overlap an actual sheet region.	37%	37%	44%

5. FUTURE WORK

We are considering a number of extensions to FSKBANN in the hopes of increasing the gain in predictive accuracy. The extensions for the structure-prediction problem can be broken down into two classes: changes to the architecture and training style of the neural network; and additions to the domain theory. Beyond producing a better structure-prediction network, we are also studying the problem of translating the trained network back into a human-readable form. Finally, we are interested in applying FSKBANN to other problems including biological (e.g. recognizing splice-junctions, finding reading frames) and other problems (such as natural language understanding).

5.1. Different Neural Network Architectures

Our studies and other work in the field ([Holley89, Qian88]) suggest a number of changes to the method of training the networks. One such observation noted by Holley and Karplus [Holley89] is that the activity of the units corresponding to helix and sheet prediction seem to correlate with how likely the prediction is correct. Table 11 indicates that this trend is even more apparent in our networks. This table reports the accuracy of those predictions when only one of the three output units (helix, sheet, coil) is above the *threshold* and the others are below (*1 - threshold*). So for example, 32% of the time one of the units activation was above 0.8, the others below 0.2, and the accuracy of these predictions was 80%.

Table 11. Predictive accuracy and coverage as a function of output-unit activation.

Threshold	Accuracy	%Predicted
0.6	69%	70%
0.7	75	49
0.8	80	32
0.9	86	16

This observation suggests a possible change to the prediction method: instead of predicting all of the protein's structure in one scan, predict only the strongest activated areas first, then feed these predictions back into the network for the next scan. This might be an advantage since helix and sheet regions consist of multiple amino acids, so predicting one indicates others might be adjacent. Figure 10 outlines how this process of structure prediction might work. The question then becomes how should the information be fed back into the network and which predictions should be chosen? Figure 11 illustrates the general structure of the type of network we plan to use to feed the information back in. There is only one difference between this and the networks in earlier figures. The input for each window position not only represents the amino

Primary Structure	P	S	V	F	L	F	P	P	K	P	...
Initial Prediction	·	·	·	·	·	·	·	·	·	·	...
Step 1	·	·	·	·	β	·	·	c	·	·	...
Step 2	c	·	·	·	β	β	·	c	·	·	...
Step 3	c	c	·	β	β	β	·	c	α	α	...
.											
.											
.											
Final Prediction	c	c	β	β	β	β	c	c	α	α	...

Figure 10. Predicting structure by filling in the most likely structures and using that information for subsequent predictions.

acid in the primary sequence, but also the current estimate of the probability of each of the three types of secondary structure (initially these values would be the *a priori* probabilities for each category). The network then predicts a set of probabilities for each of the positions on the protein. The algorithm would next evaluate this vector of probabilities for the best predictions and threshold the best predictions before the next scan. This type of network would allow the system to take advantage of information about predictions that have already been made. The network would be trained similar to FSKBANN networks — scanning a protein and making a prediction for each amino acid, or it would be trained using a recurrent technique.

A second idea, originally suggested in Qian and Sejnowski's work [Qian88] is to use two neural networks instead of one (which they call *cascaded networks*). This idea is appropriate for translating the Chou-Fasman algorithm, since this algorithm has two largely separate parts: finding nucleation sites and extending sequences. Figure 12 sketches a scheme for using two networks. For FSKBANN, the first network would be constructed using a domain theory about nucleation sites and would try to predict just the probability for each position being part of a particular structure. The second neural network would use these probabilities, along with primary-structure information and knowledge about extending structures. It would incorporate knowledge about the past predictions as it was scanning along the protein (since this information is needed for the rules to extend structures). This second network might exhibit the effects

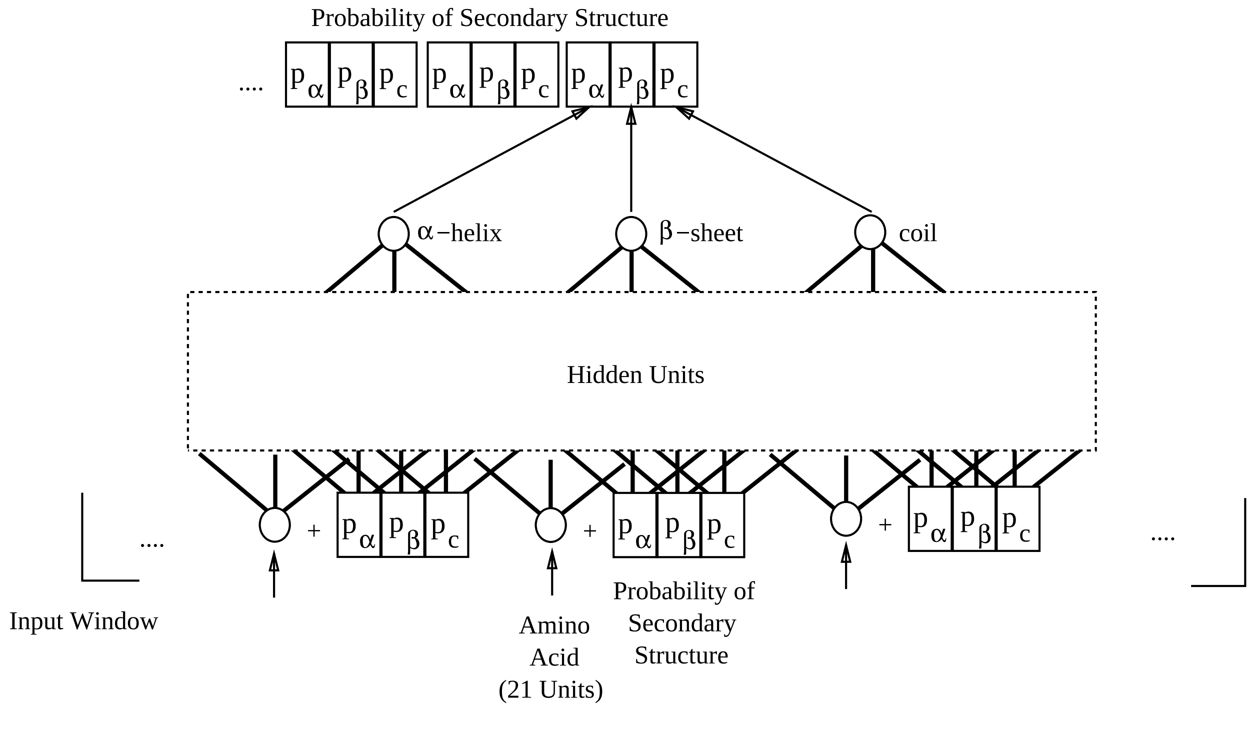
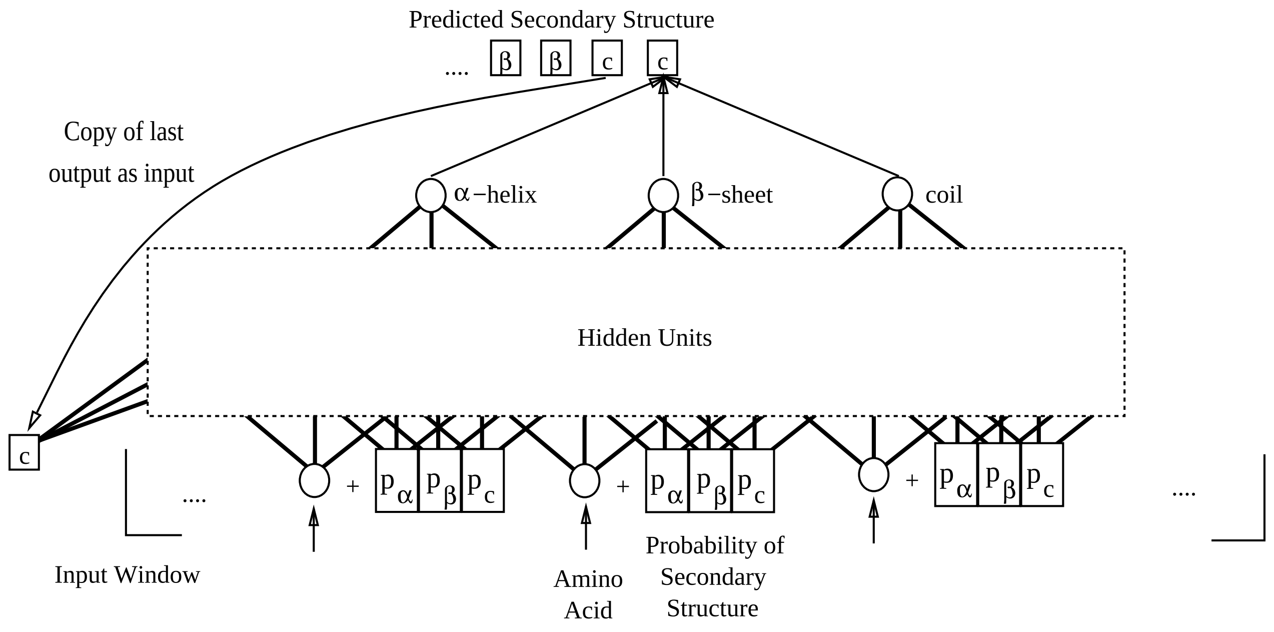
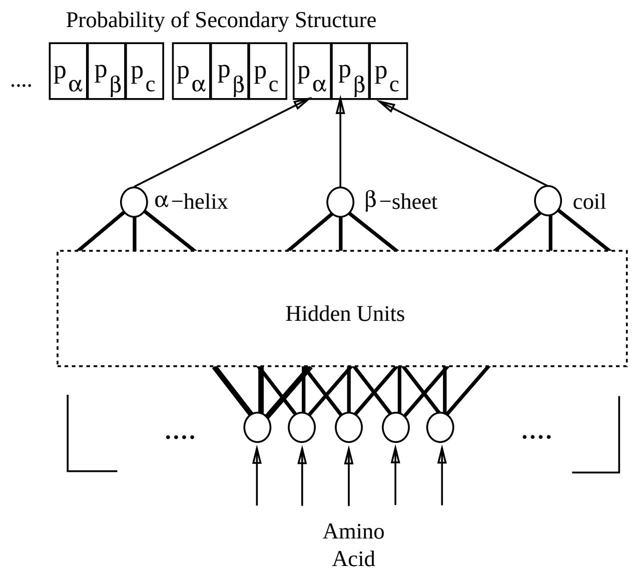

Probability of Secondary Structure
....
p_α p_β p_c
α−helix
β−sheet
coil
Hidden Units
....
+
Input Window
Amino Acid (21 Units)
Probability of Secondary Structure

Figure 11. Network that includes current prediction probabilities as part of the input.



Predicted Secondary Structure
.... β β c c
Copy of last output as input
α−helix
β−sheet
coil
Hidden Units
c
....
+
p_α p_β p_c
....
Input Window
Amino Acid
Probability of Secondary Structure

(b)



(a)

Figure 12. Networks for cascaded prediction: (a) first network that calculates the probability for each structure for each amino acid, and (b) second network that combines predictions from first network with primary structure to predict secondary structure.

discussed by Qian and Sejnowski of connecting areas of secondary structure separated by one anomalous prediction and leaving out single, anomalous predictions.

A final idea is to redefine the task of learning an FSA somewhat. Instead of focusing on predicting the state at each step, one can view the problem as predicting locations where state transitions occur. That is, rather than predicting the structure for a particular amino acid, the goal would be to find places between amino acids where the structure changes from one type to another. Figure 13 contains a sample network for this approach. The transition that the network predicts is between the middle two amino acids. The network has output units representing transitions from one type of structure to a different type of structure. In the cases where the structure does not change between amino acids, the network would predict no transition. An open problem with this approach is deciding how to combine the predicted transitions into a consistent secondary structure.

5.2. Additions to the Structure-Prediction Domain Theory

Besides altering the network architecture, we are considering a number of changes to the domain theory and problem representation. KBANN uses a domain theory to give the network a "good" set of initial weights, since search starts from that location in weight space. Therefore

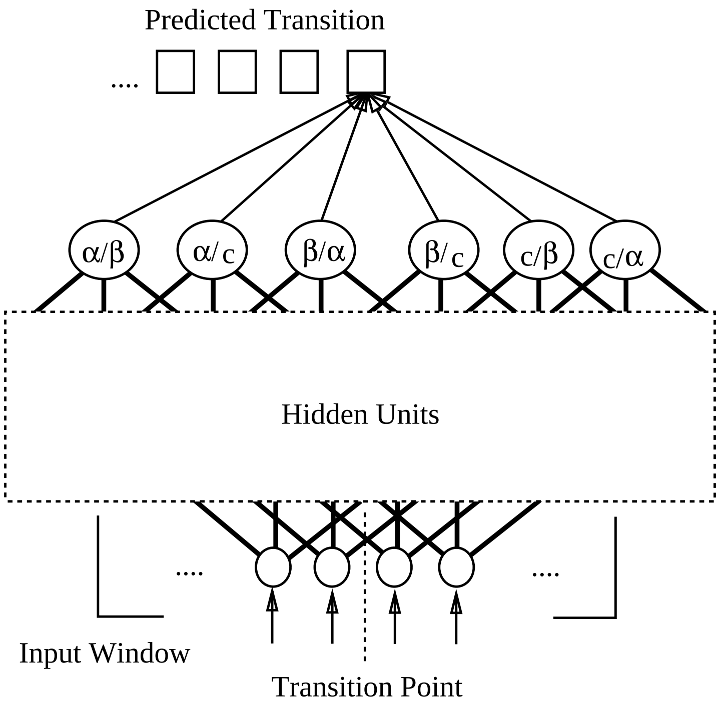


Figure 13. Network to predict finite-state transitions.

augmenting the Chou-Fasman domain theory with other information might increase the accuracy of the solution, since training will start from a "better" location.

A basic way of augmenting the domain theory is to include information from other domain theories such as the GorII and GorIII algorithms [Garnier89] and Lim's method [Lim74]. The advantage of adding these other approaches is that they may have strengths in prediction where Chou-Fasman is weak. Another advantage is that we can add these domain theories without having to consider how they interact with the others, since backpropagation training can incorporate the different pieces of information together [Jacobs91].

Hunter [Hunter91] suggests using an input representation of the amino acids that is more meaningful. As with other neural network approaches to this problem, our input encoding uses one bit for each of the possible amino acids (so for each amino acid one bit is on and the others are off). Hunter instead suggests a 37-bit representation where the bits encode properties of the amino acid that are meaningful to biologists and which are shared with other amino acids. The encoding is still unique for each of the amino acids, so no information is lost, but the network might be able to find a better solution since it can focus not only on different amino acids, but on different properties of the amino acids.

5.3. Translation of Trained Networks into Symbolic Terms

A basic problem of the KBANN approach is extracting information in a human-readable form from the trained network. Towell and Shavlik have approached this problem in the KBANN system [Towell91] and the problem has also been examined by other researchers [Fu91]. We need to address rule extraction for the augmented networks of FSKBANN which include state information, so as to extract learned FSAs.

5.4. Other Problems

Finally, there is the question of using FSKBANN in other domains. A couple of biological problems that seem natural for this type of approach are the splice-junction problem [Noordewier91] where the states are *intron* and *exon*, and the reading-frame problem [Staden90] where the states are *reading frames 1, 2,* and *3*. Also of interest is evaluating this approach for problems in other fields such as natural language (as in [Elman90]). The task could involve learning to recognize simple sentences involving a simple (regular) grammar where some information about the grammar is known, but some is missing or incorrect.

6. RELATED RESEARCH

Our work shares similarities with research in three areas: algorithms for predicting protein structure; neural networks that use state information; and systems that refine domain theories.

6.1. Methods of Predicting Protein Secondary Structure

There have been a number of different algorithms proposed for predicting protein secondary structure. These can loosely be divided into ones that use biological knowledge (and are non-learning methods) and those that use a learning mechanism.

Non-Learning Methods

The three most widely used approaches in the biological literature [Fasman89] for predicting protein secondary structure are the Chou-Fasman [Chou78, Prevelige89], Robson [Garnier89, Robson76], and Lim [Lim74] algorithms. Chou-Fasman relies on information concerning the likelihood for each type of amino acid to be part of a particular secondary structure. This information is summed over local areas of the protein to determine the likelihood for that area to be part of a particular structure. The Robson et al. [Robson76] and the later GorII and GorIII [Garnier89] solutions are based on information theory. These approaches, like neural networks, base prediction on a window of information around a central amino acid (from position -8 to +8 around the central amino acid). For every window position, the Robson algorithm determines the relevance of the amino acid for predicting each type of secondary structure. Computerized versions of the Chou-Fasman and Robson techniques that we implemented and tested on the Qian and Sejnowski test data exhibit 58% accuracy (see Table 4). The Lim method [Lim74] is the only one that tries to account for long-range interactions; it uses a stereochemical theory of globular proteins' secondary structure. Later solutions [Garnier89, Prevelige89] include theories of a fourth type of secondary structure, β-turns, which usually are classified as coils. The main advantage of FSKBANN over these algorithms is that while FSKBANN contains some of the biological information, it also has a mechanism for learning.

Learning Methods

A number of investigators use learning algorithms and sample folded proteins to try to predict secondary structure of new proteins. Holley & Karplus [Holley89] and Qian & Sejnowski [Qian88] use simple one hidden-layer neural networks to try to predict secondary structure. Both studies focus on varying the hidden unit size and window size, achieving very different results (as shown in Table 5) for these parameters, though both report test set accuracies around 63%. Qian and Sejnowski also use a cascaded architecture which produces a 1.6 percentage point improvement in accuracy over their single network results. Stolorz et al. [Stolorz91] use a perceptron architecture to evaluate a different error function, *mutual information*, which produces a one percentage point gain in accuracy over a standard perceptron. The interesting thing about the Stolorz measure is that it improves helix and sheet prediction at the expense of coil prediction, a desirable effect since coil (making up 54% of the training data), tends to be overpredicted in many other neural-network techniques.

Zhang et al. also use machine learning [Zhang90]. Their method combines information from a statistical technique, a memory-based reasoning algorithm, and a neural network. They divide the training set into halves, where each of the three components is trained on one half of the training set. Further training is done using the other half of the training set to learn to combine results from the three different components, using a second neural network. The best results they report are 66.4% for a training set size of 96 proteins [Zhang91].

The major difference between these learning approaches and FSKBANN is that only FSKBANN incorporates domain knowledge. FSKBANN also differs in that the neural networks used in our studies uses state information, unlike the above approaches.

6.2. Methods of Representing State Information in Neural Networks

Several researchers have proposed neural-network architectures for incorporating information about state. The idea of retaining a state or context across training patterns occurs in work aimed at solving natural language problems [Cleeremans89, Elman90, Mozer91, Porat91, Servan-Schreiber91, St. John90]. Each of these approaches provides a mechanism for preserving one or more of the past activations of some units for the next input sequence.

Jordan [Jordan86] and Elman [Elman90] introduced the particular recurrent network topology we use in FSKBANN. Their networks have a set of hidden units called *context units* which preserve the state of the network. At each time step the previous value of the context units are copied back as input to the system. These networks allow for the possibility of keeping multiple past contexts as input to the system. One difference between our networks and those of Elman and Jordan is that our context units are output units instead of hidden units, since the output of our network is the state.

The idea of using the type of network introduced by Jordan to represent a finite-state automaton was first discussed by Cleeremans et al. [Cleeremans89]. They state that this type of network can perfectly learn to recognize a grammar derived from a finite-state automaton. The major difference between our research and that of Cleeremans et al. is we focus on using an initial domain theory expressed as a finite-state automaton, rather than attempting to learn it from scratch.

6.3. Methods of Refining Domain Theories

There have been a number of efforts aimed at refining approximate domain theories. Ourston and Mooney's EITHER system [Ourston90] uses a domain theory to focus the corrections that an inductive learning system performs. This system constructs an explanation (a proof), possibly incomplete, of instances of a concept. The inductive learner examines the missing portions or errors in the concept's explanation to suggest changes to the domain theory that would make the explanations correct.

Flann and Dietterich [Flann89] examine a method that could fix a domain theory that is too general. Their system takes several examples of the concept being refined and produced explanations for each using the domain theory. These explanations are passed to an inductive learning system which finds the maximally-specific shared explanation structure. Our research differs with these efforts in that FSKBANN refines domain theories with recursive rules, and does not assume the domain theory is overly general.

VanLehn's SIERRA system [VanLehn87] extends a domain theory expressed as a grammar. This system constructs the *maximal partial parse* of a set of training examples to determine where there are gaps (or "holes") in the parse for the examples. SIERRA uses an inductive learning algorithm to fill the gap in the grammar so that these examples can be correctly parsed. SIERRA depends on receiving a good sequence of training examples because it assumes that each set of training examples presented shares the same gaps and that each set presented will not have too many gaps. Since FSKBANN uses a neural network methodology to refine the domain theory it is less vulnerable to poor sequences of data.

7. CONCLUSIONS

We present and evaluate FSKBANN, a system that broadens the KBANN approach to a richer, more expressive vocabulary. FSKBANN provides a mechanism for translating domain theories represented as generalized finite-state automata into neural networks. These networks can be further trained, using techniques such as backpropagation, to refine the initial domain knowledge. The extension of KBANN to domain theories that include knowledge about state significantly enhances the power of KBANN; rules expressed in the domain theory can take into account the current problem-solving context (i.e. the state of the solution).

We tested FSKBANN by refining the Chou-Fasman algorithm for predicting protein secondary structure. The FSKBANN-refined algorithm proved to be more accurate than both standard neural network approaches to the problem and a non-learning version of the Chou-

Fasman algorithm. The FSKBANN solution proved even more effective when considered in terms of how well it does for each class of secondary structure; FSKBANN shows gains in predictive power for all three classes over both ANNs and the non-learning Chou-Fasman algorithm.

Finally, our work demonstrates the need for better criteria for evaluating solutions to the protein-structure problem, since a direct, straightforward measure of how predicted structures match the actual structures is often misleading. A better definition of solution quality, possibly based on edit distance [Cohen91], might favor solutions that do well at predicting all three classes of secondary structure.

The success of FSKBANN for the secondary-structure problem indicates that it may prove to be a useful tool for addressing other problems that include state information. However, more work must be done both in improving the neural-network refinement process and the extraction of symbolic knowledge from the trained network.

Acknowledgements

This research was partially supported by National Science Foundation Grant IRI-9002413 and Office of Naval Research Grant N00014-90-J-1941. The authors would like to thank Terrence Sejnowski for providing the data used in testing. We would also like to thank Geoffrey Towell, Mark Craven, Lorien Pratt, Charles Squires and Gary Lewandowski for their valuable discussion.

References

[Bernstein77]
F. Bernstein, T. Koeyzle, G. Williams, J. Meyer, M. Brice, J. Rodgers, O. Kennard, T. Shimanouchi and M. Tatsumi, "The protein data bank: A computer-based archival file for macromolecular structures.", *Journal of Molecular Biology 112*(1977), pp. 525-542.

[Chothia84]
C. Chothia, "Principles that determine the structure of proteins", *Annual Review of Biochemistry 53*(1984), pp. 537-572.

[Chou78]
P. Chou and G. Fasman, "Prediction of the secondary structure of proteins from their amino acid sequence", *Advanced Enzymology 47*(1978), pp. 45-148.

[Cleeremans89]
A. Cleeremans, D. Servan-Schreiber and J. McClelland, "Finite state automata and simple recurrent networks", *Neural Computation 1*, 3 (1989), pp. 372-381.

[Cohen89]
F. Cohen and I. Kuntz, "Tertiary structure prediction," in *Prediction of Protein Structure and the Principles of Protein Conformation*, G. Fasman (ed.), Plenum Press, New York, 1989, pp. 647-705.

[Cohen91]
B. Cohen, S. Presnell, F. Cohen and R. Langridge, "A proposal for feature-based scoring of protein secondary structure predictions", *Proceedings of the AAAI-91 Workshop on Artificial Intelligence Approaches to Classification and Pattern Recognition in Molecular Biology,* Anaheim, CA, July 1991, pp. 5-20.

[Elman90]
J. Elman, "Finding structure in time", *Cognitive Science 14*, 2 (1990), pp. 179-211.

[Fahlman90]
S. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Neural Information Processing Systems 2*, D. Touretzky (ed.), 1990, pp. 524-532.

[Fasman89]
G. Fasman, "The development of the prediction of protein structure," in *Prediction of Protein Structure and the Principles of Protein Conformation*, G. Fasman (ed.), Plenum Press, New York, 1989, pp. 193-316.

[Flann89]
N. Flann and T. Dietterich, "A study of explanation-based methods for inductive learning", *Machine Learning 4*, 2 (1989), pp. 187-226.

[Fu91]
L. Fu, "Rule learning by searching on adapted nets", *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, CA, July 1991, pp. 590-595.

[Garnier89]
J. Garnier and B. Robson, "The GOR method for predicting secondary structures in proteins," in *Prediction of Protein Structure and the Principles of Protein Conformation*, G. Fasman (ed.), Plenum Press, New York, 1989, pp. 417-465.

[Holley89]
L. Holley and M. Karplus, "Protein structure prediction with a neural network", *Proceedings of the National Academy of Sciences (USA) 86*(1989), pp. 152-156.

[Hopcroft79]
J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation,* Addison Wesley, Reading, MA, 1979.

[Hornik89]
K. Hornik, M. Stinchcombe and H. White, "Multilayer feedforward networks are universal approximators", *Neural Networks 2*(1989), pp. 359-366.

[Hunter91]
L. Hunter, "Representing amino acids with bitstrings", *Proceedings of the AAAI-91 Workshop on Artificial Intelligence Approaches to Classification and Pattern Recognition in Molecular Biology*, Anaheim, CA, July 1991, pp. 110-117.

[Jacobs91]
R. Jacobs, M. Jordan, S. Nowlan and G. Hinton, "Adaptive mixtures of local experts", *Neural Computation 3*, 1 (1991), pp. 79-87.

[Jordan86]
M. Jordan, "Serial order: A parallel distributed processing approach," Technical Report 8604, University of California, Institute for Cognitive Science, San Diego, 1986.

[Kabsch83]
W. Kabsch and C. Sander, "Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometric features", *Biopolymers 22*(1983), pp. 2577-2637.

[Lang90]
K. Lang, A. Waibel and G. Hinton, "A time-delay neural network architecture for isolated word recognition", *Neural Networks 3*(1990), pp. 23-43.

[Lim74]
V. Lim, "Algorithms for prediction of α-helical and β-structural regions in globular proteins", *Journal of Molecular Biology 88*(1974), pp. 873-894.

[Mathews75]
B. Mathews, "Comparison of the predicted and observed secondary structure of T4 Phage Lysozyme", *Biochimca et Biophysica Acta 405*(1975), pp. 442-451.

[Mozer91]
M. Mozer and J. Bachrach, "SLUG: A connectionist architecture for inferring the structure of finite-state environments", *Machine Learning 7*, 2/3 (1991), .

[Nishikawa83]
K. Nishikawa, "Assessment of secondary-structure prediction of proteins: Comparison of computerized Chou-Fasman method with others", *Biochimica et Biophysica Acta 748*(1983), pp. 285-299.

[Noordewier91]
M. Noordewier, G. Towell and J. Shavlik, "Training knowledge-based neural networks to recognize genes in DNA sequences," in *Advances in Neural Information Processing Systems*, Vol. 3, , Morgan Kaufmann, Denver, CO, 1991.

[Ourston90]
D. Ourston and R. Mooney, "Changing the rules: A comprehensive approach to theory refinement", *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, July 1990, pp. 815-820.

[Porat91]
S. Porat and J. Feldman, "Learning automata from ordered examples", *Machine Learning 7*, 2/3 (1991), .

[Prevelige89]
P. J. Prevelige and G. Fasman, "Chou-Fasman prediction of the secondary structure of proteins: The Chou-Fasman-Prevelige algorithm," in *Prediction of Protein Structure and the Principles of Protein Conformation*, G. Fasman (ed.), Plenum Press, New York, 1989, pp. 391-416.

[Qian88]
N. Qian and T. Sejnowski, "Predicting the secondary structure of globular proteins using neural network models", *Journal of Molecular Biology 202*(1988), pp. 865-884.

[Richardson89]
J. Richardson and D. Richardson, "Principles and patterns of protein conformation.," in *Prediction of Protein Structure and the Principles of Protein Conformation*, G. Fasman (ed.), Plenum Press, New York, 1989, pp. 1-98.

[Robson76]
B. Robson and E. Suzuki, "Conformational properties of amino acid residues in globular proteins", *Journal of Molecular Biology 107*(1976), pp. 327-356.

[Rumelhart86]
D. Rumelhart, G. Hinton and R. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the microstructure of cognition. Volume 1: Foundations*, D. Rumelhart and J. McClelland (eds.), MIT Press, Cambridge, MA, 1986, pp. 318-363.

[Sejnowski87]
T. J. Sejnowski and C. R. Rosenberg, "Parallel Networks that Learn to Pronounce English Text", *Complex Systems 1*(1987), pp. 145-168.

[Servan-Schreiber91]
D. Servan-Schreiber, A. Cleeremans and J. McClelland, "Graded state machines: The representation of temporal contingencies in simple recurrent networks", *Machine Learning 7*, 2/3 (1991), .

[Shavlik91]
J. Shavlik, R. Mooney and G. Towell, "Symbolic and neural learning algorithms: An experimental comparison", *Machine Learning 6*(1991), pp. 111-143.

[St. John90]
M. St. John and J. McClelland, "Learning and applying contextual constraints in sentence comprehension", *Artificial Intelligence 46*, 1-2 (1990), pp. 217-257.

[Staden90]
R. Staden, "Finding protein coding sequences in genomic sequences", *Methods in Enzymology 183*(1990), pp. 163-180.

[Stolorz91]
P. Stolorz, A. Lapedes and Y. Xia, "Predicting protein secondary structure using neural net and statistical methods," Technical Report LA-UR-91-15, Theoretical Division, MS B213, Los Alamos National Laboratory, Los Alamos, NM, 1991.

[Towell90]
G. Towell, J. Shavlik and M. Noordewier, "Refinement of approximate domain theories by knowledge-base neural networks", *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, July 1990, pp. 861-866.

[Towell91]
G. Towell and J. Shavlik, "Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules," Working Paper, Computer Sciences Department, University of Wisconsin, Madison, WI, 1991.

[VanLehn87]
K. VanLehn, "Learning one subprocedure per lesson", *Artificial Intelligence 31*(1987), pp. 1-40.

[Watson90]
J. Watson, "The human genome project: Past, present, and future", *Science 248*(1990), pp. 44-48.

[Weiss89]
S. Weiss and I. Kapouleas, "An empirical comparison of pattern recognition, neural nets, and machine learning classification methods", *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989, pp. 781-787.

[Wilson85]
I. Wilson, D. Haft, E. Getzoff, J. Tainer, R. Lerner and S. Brenner, "Identical short peptide sequences in unrelated proteins can have different conformations: A testing ground for theories of immune recognition", *Proceeding of the National Academy of Sciences (USA) 82*(1985), pp. 5255-5259.

[Zhang90]
X. Zhang, "Exploration on protein structures: Representation and prediction," Ph.D. Thesis, Department of Computer Science, Brandeis University, Waltham, MA, May 1990.

[Zhang91]
X. Zhang, Personal communication, 1991.

Appendix. The Chou-Fasman domain theory.

The Chou-Fasman algorithm [Chou78] involves three activities: (1) recognizing nucleation sites, (2) extending sites, and (3) resolving overlapping predictions. In this appendix, we provide more details of these three steps and describe our representation of their algorithm as a collection of rules.

To recognize nucleation sites, Chou and Fasman assign two *conformation* values to each of the 20 amino acids. The conformation values represent how likely an amino acid is to be part of either a helix or sheet structure, with higher values being more likely. They also group the amino acids into classes of similar conformation value. The classes for helix are *formers*, *high-indifferent*, *indifferent,* and *breakers*. Those for sheet are *formers*, *indifferent,* and *breakers*. Table 12 defines the values for the various types of breakers and formers.

Table 13 contains the rules we use to represent the Chou-Fasman algorithm; `x@N` is true if *x* is the amino acid *N* positions from the one whose secondary structure the algorithm is predicting. It predicts an α-helix nucleation site if for some consecutive set of six amino acids, at least four are helix formers and less than two are helix breakers. (Two helix high-indifferent amino acids count as a helix former.) A rule to determine if a location is a nucleation site simply adds the helix-former and helix-breaker values for a window of amino acids six wide, and if the totals are greater than four and less than two respectively, predicts a helix nucleation site (proposition `init-helix` in our rules). Nucleation of β-sheets is similar to α-helix nucleation, except that the window is only five amino acids wide and a sheet nucleation site is predicted if there at least three sheet formers and less than two sheet breakers.

The third step of the algorithm — resolving overlaps — is the reason we use the numbers in Table 12 rather than making the formers and breakers Boolean properties. Chou and Fasman suggest that the conformation values of regions be compared to resolve overlaps. This is done in our networks by weighting the links from various amino acids according to the numbers in Table 12. For example, a combination of four alanines (`A`'s) will produce a higher activation of the `init-helix` unit than a combination of four phenylalanines (`F`'s).

The Chou-Fasman algorithm continues to predict α-helix as long as the predicate `cont-helix` is true. The rules define `cont-helix` mostly in terms of helix-breaking rules — a helix continues as long as a break region is not encountered. An α-helix break region occurs when an helix-breaker amino acid is immediately followed by either another helix-breaker or a helix-indifferent amino acid. A helix is also broken when encountering the amino acid proline (`P`). The process of extending β-sheet structures works similarly. The algorithm predicts coil as the default.

Table 12. Former and breaker values[‡] for the amino acids.

helix-former(E)	=	1.37	helix-former(A)	=	1.29	helix-former(L)	=	1.20
helix-former(H)	=	1.11	helix-former(M)	=	1.07	helix-former(Q)	=	1.04
helix-former(W)	=	1.02	helix-former(V)	=	1.02	helix-former(F)	=	1.00
helix-former(K)	=	0.54	helix-former(I)	=	0.50			
helix-former(*others*)	=	0.00						
helix-breaker(N)	=	1.00	helix-breaker(Y)	=	1.20	helix-breaker(P)	=	1.24
helix-breaker(G)	=	1.38						
helix-breaker(*others*)	=	0.00						
sheet-former(M)	=	1.40	sheet-former(V)	=	1.39	sheet-former(I)	=	1.34
sheet-former(C)	=	1.09	sheet-former(Y)	=	1.08	sheet-former(F)	=	1.07
sheet-former(Q)	=	1.03	sheet-former(L)	=	1.02	sheet-former(T)	=	1.01
sheet-former(W)	=	1.00						
sheet-former(*others*)	=	0.00						
sheet-breaker(K)	=	1.00	sheet-breaker(S)	=	1.03	sheet-breaker(H)	=	1.04
sheet-breaker(N)	=	1.14	sheet-breaker(P)	=	1.19	sheet-breaker(E)	=	2.00
sheet-breaker(*others*)	=	0.00						

[‡] We produced these values using the tables reported by Chou and Fasman [Chou78, pg. 51]. We normalized the values for formers by dividing the conformation value of the given former by the conformation value of the weakest former. So for example, the helix former value of alanine (A) is 1.29 since the helix conformation value of alanine is 1.45 and the conformation value of the weakest helix former phenylalanine (F) is 1.12. Breaker values work similarly except that the value used to calculate the breaker value is the multiplicative inverse of the conformation value.

We did not directly use the values of Chou and Fasman for two reasons. One, we wanted smaller values, to decrease the number of times three very strong helix-formers would add up to more than 4 (and similarly for sheets). Two, breaker conformation values tend to be numbers between 0 and 1 with the stronger breakers being close to 0. We wanted the breaker value to be larger the stronger the breaker, so we used the inverse of the breaker's conformation value (restricting the result to not exceed 2).

Table 13. The Chou-Fasman algorithm expressed as inference rules.

Rules for recognizing nucleation sites.

init-helix	$\leftarrow$	$\left[\sum_{position=0}^{5} helix-former(amino-acid@position)\right] > 4$
		$\wedge \left[\sum_{position=0}^{5} helix-breaker(amino-acid@position)\right] < 2$
init-sheet	$\leftarrow$	$\left[\sum_{position=0}^{4} sheet-former(amino-acid@position)\right] > 3$
		$\wedge \left[\sum_{position=0}^{4} sheet-breaker(amino-acid@position)\right] < 2$

Rules for pairs of amino acids that terminate helix structures.

helix-break@0	$\leftarrow$	$N@0 \vee Y@0 \vee P@0 \vee G@0$
helix-break@1	$\leftarrow$	$N@1 \vee Y@1 \vee P@1 \vee G@1$
helix-indiff@1	$\leftarrow$	$K@1 \vee I@1 \vee D@1 \vee T@1 \vee S@1 \vee R@1 \vee C@1$
break-helix	$\leftarrow$	$helix\text{-}break@0 \wedge helix\text{-}break@1$
break-helix	$\leftarrow$	$helix\text{-}break@0 \wedge helix\text{-}indiff@1$

Rules for pairs of amino acids that terminate sheet structures.

sheet-break@0	$\leftarrow$	$K@0 \vee S@0 \vee H@0 \vee N@0 \vee P@0 \vee E@0$
sheet-break@1	$\leftarrow$	$K@1 \vee S@1 \vee H@1 \vee N@1 \vee P@1 \vee E@1$
sheet-indiff@1	$\leftarrow$	$A@1 \vee R@1 \vee G@1 \vee D@1$
break-sheet	$\leftarrow$	$sheet\text{-}break@0 \wedge sheet\text{-}break@1$
break-sheet	$\leftarrow$	$sheet\text{-}break@0 \wedge sheet\text{-}indiff@1$

Rules for continuing structures.

cont-helix	$\leftarrow$	$\neg P@0 \wedge \neg break\text{-}helix$
cont-sheet	$\leftarrow$	$\neg P@0 \wedge \neg E@0 \wedge \neg break\text{-}sheet$

Rules for predicting helix: either by nucleation or propagating from the last state.

$helix_i$	$\leftarrow$	*init-helix*
$helix_i$	$\leftarrow$	$helix_{i-1} \wedge cont\text{-}helix$

Rules for predicting sheet: either by nucleation or propagating from the last state.

$sheet_i$	$\leftarrow$	*init-sheet*
$sheet_i$	$\leftarrow$	$sheet_{i-1} \wedge cont\text{-}sheet$

Rules for predicting coil (the default).

$coil_i$	$\leftarrow$	$helix_i \wedge break\text{-}helix$
$coil_i$	$\leftarrow$	$sheet_i \wedge break\text{-}sheet$
$coil_i$	$\leftarrow$	$coil_{i-1}$