Using a Neural Network to Predict Student Responses

†Susan Mengel, ‡William Lively
†University of Arkansas
‡Texas A&M University

Abstract

One of the important components in an intelligent tutoring system is the student model. This model is used to predict what the student may do next as well as to serve as a repository of past student solutions. The student model is important in that it can help to direct the student to unknown material when enough concepts have been mastered and to material that needs to be reviewed when the student is unsure. Some student models have tried to predict student solution steps by restricting the interface to the point where the student cannot make an unknown move. Others do not concentrate on prediction, but instead concentrate on remedying errors in problem solutions. Since the problem of prediction is difficult, another tool, the neural network, should prove useful. Neural networks have the ability to generalize over a set of student answers. This ability gives the network the capacity to answer as the student would on problems that the network has never seen before. Given this exciting possibility, research has been started using the backpropagation model of neural networks to learn a student's method in performing subtraction. The preliminary results reported in this paper are encouraging and serve to show the promise of neural networks in the student model of intelligent tutoring systems.

Introduction

Intelligent tutoring systems (ITS) traditionally consist of four components: the expert module, the student model, the tutoring module, and the interface module. The expert module contains the domain knowledge of a human expert and is used to solve problems and make inferences. The student model is utilized by diagnosis procedures to make an approximation of the student's state of knowledge. Since a student may not work problems as efficiently as an expert, the tutoring module embodies the strategies necessary to reduce the difference between the student's and the expert's performance. The interface module functions as the mediator between the student and the ITS in the hope that the student will learn the material and the ITS will interpret the student's actions appropriately. Specifically, the interface presents the ITS's messages to the student and sends the student's input to the system.

One of the components of an ITS is the focus of this paper, namely, the student model. Several techniques have been used by other researchers to implement student models in order to predict student responses or to diagnose student errors. These strategies include the use of planning techniques for prediction and of bug catalogs enumerating student mistakes to find student errors. Although they have not been used in previous ITS for prediction, neural networks are beginning to be recognized as a possible tool for predicting student answers. These networks are able to memorize a pattern set, to generalize over a pattern set, to classify like patterns together, and to associate different patterns together. These attractive qualities should prove to be an asset to ITS in the student model.

Student Model

Several approaches to constructing the student model have been implemented which do not use neural networks. These approaches include the overlay model, bug libraries, plan recognition, issue tracing, expert systems, model-tracing, condition induction, and path finding. The details of these implementations may be found in VanLehn[11] and typically use traditional symbolic processing mechanisms, such as heuristic search or rule-based representations.

Beale and Finlay[1] and Zerwekh[14] have used neural networks in user and student models, respectively. Beale and Finlay employed a network to classify experts and novices in an exploratory functional programming environment and a network to determine user tasks in an on-line bibliography system. Zerwekh's network classifies foreign language students based upon their performance in a series of lessons.

Researchers have higher expectations for the student model than what actually is implemented. According to Wenger[13], the student model should reflect those aspects of student behavior and knowledge that affect the student's learning ability. The model should be executable and should be able to change as more diagnostic data is made available to the system. Sleeman and Brown[9] add three issues that the student model must incorporate. The model must determine how to assign blame in the presence of a student failure to a problem in which more than one skill is necessary for solution development. The model must limit the number of hypotheses as to the student's erroneous behavior so a combinatorial explosion of possibilities does not occur. Finally, the model should be able to work in the presence of noisy data when the student makes errors because of fatigue or overload. Clancey[3] equates the student model to a simulation model consisting of the student's knowledge and the student's inference procedure used to operate upon that knowledge. This type of model can be used to predict what the student will do next and to find the basis in the student's past history of his problem solving procedure.

An alternative viewpoint for a less complex student model than the expectations described in the preceding paragraph is held by Self[8]. The student model may serve as a repository for student beliefs about the learning material. These beliefs should be able to be obtained only through the interface instead of through using a complex inferencing procedure to generate them. Further, the student model should be visible to the student so that the student can refine and replace beliefs as necessary (this process is aided by a collaborative tutoring module that serves to guide the student strategically through the learning material without making right or wrong value judgements upon the student's beliefs).

Neural Nets

Neural networks are mathematical simulations of the human brain. The human brain has a massively parallel architecture of some 10^{11} neurons where each neuron may be connected to as many as 1,000 to 10,000 other neurons. Communication occurs among the neuronal connections via chemicals (neurotransmitters) that can be inhibitory or excitatory. If enough excitatory input is received to override the inhibitory input and exceed a neuron's threshold, then the neuron *fires*; i.e., sends an electrical signal down its axon which causes it to release neurotransmitters to other neurons.

The computational neuron[7] (shown in Figure 1) receives n inputs (from the environment or from other connecting neurons) and produces one output (to the environment or to other neu-

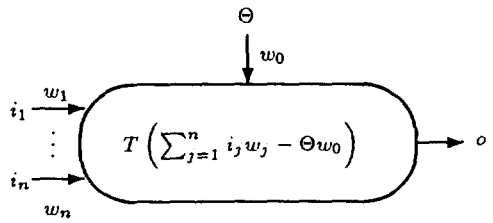

Fig. 1. Artificial neural processing unit.

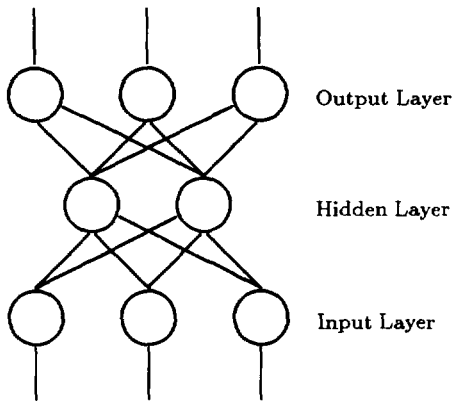

Fig. 2. Neural network.

rons). A neuron may have an additional input, Θ, that acts as a threshold or as an outside environmental input. If the input is from the environment, then such input should be received without modification by the neuron. If the input is from a connecting neuron, then the input may need to be attenuated or strengthened. For this purpose, the input may be multiplied by a weight.

To produce its output, the neuron may use a threshold (activation or signal) function to map the input to a specified range. Usually this range is very small ($[0,1]$) in order to keep neurons from having extremely large values. Using a threshold function is often necessary to cause an entire network of neurons to converge or *settle down* during training and recall.

Neurons generally are connected together into formations that are found in the brain[12]. For instance, some neural networks may be connected in a hierarchical fashion; i.e., a layer or field of neurons (a set of neurons that reside in the same group) may be connected to an upper layer of neurons, and that upper layer may be connected to yet another higher layer. An example of a three layer network is shown in Figure 2. Each field or layer has some number of neurons and the connections may be made *intra-field* where neurons in the same field are connected to each other, *inter-field* where neurons in one field are connected to neurons in different fields, or both. Usually, there exists an input field, an output field, and fields in between those. The in between fields are known as hidden layers.

The weights on the connections hold the information that the entire network has learned in order to perform correctly; i.e., the weights correspond to long term memory in humans. The weights also are characterized as long term memory because changes to them occur in very small increments; therefore, they do not *forget* quickly. The neuron, itself, corresponds to short term memory because it replaces its value every time new input is presented.

The weights can be modified as the network is trained so that the network produces the desired output. For example, the weight change of a weight can be given by an equation, such as the one for backpropagation[6]:

$$\Delta w = \lambda \delta o + \mu \Delta w_{previous}$$

where λ is the learning rate, δ is the output error of the neuron connected to the weight, o is the output of that neuron, and μ is the momentum term. The learning rate determines the size

a) Change 0-n to 9-n.

$$\begin{array}{r} 540 \\ -434 \\ \hline 115 \end{array}$$

b) Subtract smaller from larger digit.

$$\begin{array}{r} 456 \\ -374 \\ \hline 122 \end{array}$$

c) Add 10 to borrower.

$$\begin{array}{r} 70^{1}3 \\ -60\ 5 \\ \hline 10\ 8 \end{array}$$

d) Increment from point of borrow.

$$\begin{array}{r} 678 \\ 700^{1}3 \\ -511\ 5 \\ \hline 167\ 8 \end{array}$$

Fig. 3. Subtraction errors.

of the weight changes and the momentum term is designed to keep the network from becoming trapped in a local minimum. As the weight changes are made, the network travels through hills and valleys until it becomes trapped in a valley which is characterized by the fact that further weight changes have little effect upon the current value of the weights. If the valley is not the global minimum, but instead is a local minimum, then the network may not have learned properly; i.e., the network may not give the correct output for an input on which it has been trained.

The signals or information flow in a neural network may be feedforward (information flows in one direction from the input layer to the output layer), feedback (information flows in either direction; i.e., it can bounce back and forth from input to output layer), or lateral (information flows among neighboring neurons in the same field). A network with feedback is known as a recurrent network. Laterally connected neurons usually occur in the output field of competitive networks. A competitive network only allows one output neuron to fire thereby classifying its input. The output neuron also may send signals over the lateral connections preventing the other output neurons from firing.

Neural networks map data to data and can store two types of patterns or vectors: spatial, such as a single static image, and spatiotemporal, such as a sequence of spatial patterns. Patterns are generally sequences of one's and zero's and are given meaning by the designer of the neural network. If a neural network only stores single patterns $(A_1, \ldots, A_n)$, then it is autoassociative. If it stores pattern pairs $((A_1, B_1), \ldots, (A_n, B_n))$, then it is heteroassociative.

Student Model Experiments

To demonstrate the usefulness of neural networks in the student model, a set of three experiments was performed in order to analyze the predictive power of the networks even with incomplete or conflicting data. This section shows the progression of the experiments to elucidate the factors involved when using a neural network in the student model. First, the domain, data, and network learning paradigm of the experiments are considered. Then three experiments and their results are discussed.

Domain

The domain that was chosen is subtraction. The reason for this choice is that subtraction problems may be mapped into a problem pattern and then into an answer pattern for a neural network to learn. A more important reason for this choice is that Brown and Burton[2] have analyzed subtraction so well. In fact, they found 110 bugs students can commit when solving a subtraction problem and enumerated the bugs in the manual by Friend and Burton[4].

The subtraction errors students committed varied greatly and could be quite complex. Some students ignored the subtraction sign and performed addition. Many errors involved borrowing where $0 - n$ would be changed to $9 - n$. Also, $m - n$ would be changed to $n - m$ if n were greater than m. Some students would not borrow across a zero, they simply would add ten to the column that required a borrow. When borrowing across several zeroes, some students would add one to the last borrow in succession so that 7003 would become 678(13) after borrowing one from the seven. Examples of these errors are shown in Figure 3.

Network Training Data

For the experiments, tight control needed to be maintained over the data so that the predictive power of the networks could be measured accurately. To maintain this control, *synthetic* student data generated from the results found by Brown and Burton was used. Thus, all possible answers to subtraction problems presented to a network could be obtained. Real student data would not be as complete and so a network's results on missing data (i.e., data on which it had not been trained) could not be checked if the student had not worked the problem. Furthermore, repeating Brown and Burton's study by analyzing the work of 1300 students would have detracted from the goal of showing the predictive power of neural networks. Fortunately, their study is exhaustive and a good representation of what students might do when performing subtraction.

Network Learning Paradigm

Backpropagation was the paradigm used since a backpropagation network is known to be able to generalize over a pattern space and to learn far more patterns than the number of neurons in the network[6]. It is a feedforward paradigm and is sufficient for showing the predictive power of neural networks.

Experiments

The experiments were designed to see if neural networks can replicate the behavior of a student working subtraction problems correctly or incorrectly when the data from the student is complete, incomplete, or noisy. The overall approach consisted of the following major steps:

- design a network to perform subtraction
- train the network on complete data
- train the network on incomplete data
- train the network on erroneous data
- train the network on noisy data.
- evaluate the network's ability to model the student's cognitive processes under each scenario.

Specifically, the first part of the experiments involved designing and training a backpropagation network in order to see if the network could perform subtraction. Having determined that the network could perform subtraction, random patterns were deleted from the pattern set in increments of ten to see how the network performed in the absence of some information. Following this test, the network was trained on data which had some of the errors introduced from the manual by Friend and Burton[4]. Once again, after the complete pattern set with errors was used, random patterns were deleted in increments of ten to determine how well the network generalized with incomplete data. Further, conflicting data representing a student's careless or random errors was introduced into the pattern set to see if the network trained properly and reflected the gist of the problem solving process of the student.

The neural networks were simulated by a C program written by the author on a Sun-3[1] workstation running UNIX[2]. All of the following experiments were performed with this program. The program implements the equations of Rumelhart, Hinton, and Williams[6] with the following exception: the derivative of the sigmoid function is not used on the output layer error. Several experiments on networks without the derivative caused learning to occur much faster. The training of the network was stopped when the error of the output layer neurons reached a predetermined cutoff point. This error is calculated in the following manner. For each pattern, the error is $\frac{1}{2}\sum_i (t_i - o_i)^2$ where t_i is the target output for neuron i and o_i is the actual output. The error for all patterns is summed yielding the overall error (square error) used to determine the goodness of network learning. Since the networks may not learn all of the patterns, to stop training when the network will not learn the patterns any better is best. During training, the weights and biases were updated after each pattern was presented.

[1]Registered trademark of Sun Microsystems, Incorporated.
[2]Registered trademark of AT&T

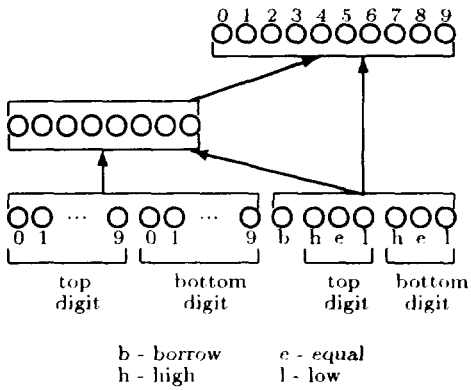

Fig. 4. First single-column subtraction network.

Experiment One: After discovering that subtraction could be expressed in terms of a pattern recognition task and that the area of subtraction had been researched well, the design of the network remained. This network needed to learn subtraction even with incomplete data since the number of problems a student may be expected to work is not infinite. Therefore, the domain was constrained to problems involving a larger three-digit number minus a smaller three-digit number. Unfortunately, the total number of patterns that the network could learn turned out to be 500,500 (1000 patterns: $999 - n$, 999 patterns: $998 - n$, ..., 1 pattern: $0 - 0$) even with the aforementioned restriction of three-digit problems. Obviously, a student cannot be expected to work this many problems. The network, however, would need almost that many patterns to train properly. To get around this difficulty, the number of neuronal connections was constrained and the pattern set was limited to 20 patterns to see if the network could train well on an incomplete number of patterns.

Several networks were tried to perform three-digit subtraction. None of them would even converge. Instead they diverged with the weights taking on ever increasing values.

The result of this experiment showed the difficulty of overcoming the *scaling problem* in neural networks. If the training pattern set is not large enough, then a network may not be able to learn the patterns. For any ITS, a network, however, must be designed so that the total pattern set does not exceed the number of problems a student is likely to work. Sometimes additional information can be added to each of the patterns to help the network train with a smaller pattern set as with Tesauro and Sejnowski's[10] backgammon network (the number of possible moves in backgammon is prohibitively large, but they were able to train the network on 3202 board positions with 429 input units denoting seven other characteristics besides the board configuration). Even with their success, finding another network with a much smaller pattern set seemed appropriate.

Experiment Two: Instead of using a three-digit problem, the use of a network that only operated upon a single column (two digits) at a time seemed better. The network so designed is shown in Figure 4. Part of the input layer consists of the two digits where a one would be placed in neuron n and zeroes in the rest for a particular digit. For instance, 0001000000 0000000001 would represent $3 - 9$ since a one is in the three's position for the top digit and a one is in the nine's position for the bottom digit. The next input neuron denotes whether the subtraction is performed in the presence of a borrow (1 - yes and 0 - no). The next six neurons denote whether the top digit is greater than (h), less than (l), or equal to (e) the bottom digit. For example, these neurons would take on the values 001100 for $3 - 9$ because three is less than nine. The six neurons are present to help the process of generalization. All of the input layer neurons are connected to the middle layer, but the borrow and comparison neurons also are connected to the top layer. This action was taken to let the additional neurons affect the generalization process in order to enhance the process and to give them more of an effect in the result of the subtraction. The output layer only has one neuron take on a value of one to denote the result of the subtraction. In the case of $3 - 9$ not in the presence of a borrow, the result of the network would be 0000100000 for four (in the presence

TABLE I: E2 - TRAINING PATTERN SETS

Correct Subtraction
Subtracts Smaller From Larger Digit
Writes 0 in Columns Requiring a Borrow
Thinks $n - n$ is n
Thinks $n - 0$ is 0
Thinks $0 - n$ is 0
Thinks $0 - n$ is n
Thinks $1 - n$ is 1
Changes $0 - n$ to $9 - n$
Borrows, but Writes 0 Where the Borrow is Needed

of a borrow, it would be 0001000000 for three). A hexadecimal representation was not used since the digits A through F are not present in the decimal number system. Interpreting a B as the answer for $3 - 9$ would be difficult. Instead, only the neuron with the largest value in the output layer becomes a one.

The operation of this network may be thought to be similar to the way a human might perform subtraction. First, the two digits in the least significant column would be subtracted, then the next two, and so forth. For the problem $634 - 299$, the network would get $4-9$ first, $3-9$ in the presence of a borrow second, and, finally, $6 - 2$ in the presence of a borrow. This network, however, does not propagate a borrow, itself, so an outside source would have to generate a borrow for the network if a borrow were needed. Since subtraction columns are used, the total pattern set consists of 200 patterns (ten $9 - n$ not in the presence of a borrow, ten $9 - n$ in the presence of a borrow, ten $8 - n$ not in the presence of a borrow, ten $8 - n$ in the presence of a borrow, ..., ten $0 - n$ not in the presence of a borrow, ten $0 - n$ in the presence of a borrow).

To test the network on incomplete data, random patterns were eliminated from the pattern set in increments of ten for up to 60 patterns. The limit of 60 was set in order to be able to perform more runs and because the remaining 140 patterns could be obtained from a small subtraction test consisting of about 28 five-digit problems.

The backpropagation simulator for this experiment was run on a Sun SPARCstation[3] which has considerable power. A limit was set of 20,000 iterations through the entire pattern set (i.e., one iteration is one time through the full pattern set) and of some preset square error value to stop the network when convergence had taken place. Instead of coming as close as possible to the smallest square error that could be obtained, a coarser value was used. Limitations on a training run were set in order to be able to have more training runs since 20,000 iterations through the pattern set could take around 20 cpu hours or about two days. The designer of an ITS may wish to be more careful when training the network, but certainly would not want a network to take too long to train. As much as possible, all learning parameters were kept the same, once again, to allow for more runs.

The data sets used for the network in this experiment are shown in Table I. First the correct subtraction training set was used and then nine sets with subtraction errors were used.

Table II shows the result of training the network to do correct subtraction. Each line in the table represents one pattern set on which the network was trained. Line one is for the complete pattern set of 200 patterns, line two is for the pattern set of 190 patterns constructed by the random deleting of 10 patterns from the complete set of 200, and so forth. The square error column shows the difference of the network output from the desired output after the network completed training. The percentage of the deleted patterns recognized and the training set patterns recognized is shown in the next two columns. The percentage of patterns in the complete pattern set of 200 the network recognized is shown in the next column. The last column shows how many times the network had to go through the training pattern set in order to learn the patterns. For example, the network was able to learn subtraction in 11,628 iterations through the complete pattern set and had a square error of 1.99. On the other hand, after eliminating just ten patterns, it learned the other 190 in 603 iterations and recognized 40% out of the ten eliminated even though those ten patterns had not been present in the training pattern set. After eliminating twenty patterns, the network did not even converge to the square error limit of 2.0.

[3]Registered trademark of Sun Microsystems, Incorporated.

Even with this odd behavior (probably the network encountered a local minimum), the results are encouraging as in the case with 30 patterns missing where 77% of the missing and 99% of the present patterns were learned for a total of 96% correctness. In contrast, a rule base designed to process the training pattern set of 140 patterns would not be able to handle the other 60 since rule bases can deal with only what is built into them. The network, however, could recognize 37% of the deleted patterns.

The first error on which the network was trained was the error where a student does not borrow at all, but instead subtracts the smaller from the larger digit. The network learned this error well as shown in Table III. In every case the network learned quickly and only missed up to 11% of the full set of patterns. This error is easy to learn since it is systematic; i.e., perform normal subtraction when a borrow is not needed and do the same thing, although wrong, when a borrow is needed.

The next error was the mistaken notion that $0 - n$ is 0. This error is harder to learn because it goes against the overall notion of the pattern set which essentially is correct. Indeed, Table IV displays the difficulty the network had with this error as in some cases the network did not converge completely and missed more of the patterns than in the runs with the previous error. The worst performance occurred when 60 of the patterns were deleted and only 77% of the full pattern set was recognized correctly with only 23% of the missing patterns correctly recognized.

The fourth through the eighth errors are of the same type as the previous error, rather peculiar and going against the overall pattern set. The ninth error, however, is systematic. The network's results on these errors may be found in Mengel[5].

The average number of patterns recognized by the network is shown in Table V. The results are encouraging since the network could learn the patterns present in the training set. The network also could recognize properly over 50% of the patterns. It did need some help in recognizing the missing patterns.

At this point, work on the network terminated since the network was having trouble recognizing peculiar errors. Additionally, a borrow was put into a pattern if correct subtraction required it even if the incorrect subtraction did not. Clearly, another network was needed.

The results of this experiment showed that the network has the ability to generalize over complete and incomplete pattern sets. A cause for concern to some ITS designers may be that the network did not recognize all patterns and performed poorly in other instances. That the student model be as accurate as possible is very important, but even humans cannot achieve perfect performance in all cases. Furthermore, other AI paradigms, such as rule bases, cannot handle unknown data at all.

One of the more important issues is how many parameters can be adjusted to affect the network's learning performance. One can reorder the pattern set, change the learning and momentum terms, change the square error cutoff, alter the initial setting of the weights before training, affect the updating of the weights (after each presentation of the pattern set or after each pattern), try numerous small changes to the backpropagation algorithm to speed it up, and redesign the network. Having to modify all of these parameters is beneficial rather than detrimental. Having the ability to modify the student model for students who may deviate from the norm is better. The alternative is having a static model where no changes can be made and certain students cannot be modeled.

The odd behavior of the network when training is a concern since in some situations it did not train in 20,000 iterations when m number of patterns were missing, but did just fine when n patterns were missing. This problem is due to the network encountering local minima. Changing the weight initializations and varying the learning rate may help to solve this difficulty.

Another concern is the time it took to train the network. Realistically, the network used in this experiment would not train on a personal computer as quickly as it does on the Sun SPARCstations. Once trained, however, the network is almost instantaneous in getting the result on a given input. Also, more powerful personal computers are becoming available.

The positive aspects of neural networks for ITS brought out by this experiment include the ability to generalize, to train from student data, and to recall patterns instantly. In total, over 70 training runs were made with the runs shown and runs to

TABLE II: E2 - CORRECT SUBTRACTION

Run	Square Error	Missing Patterns Recognized	Present Patterns Recognized	Total Recognized	Iterations
All Patterns	1.99	*NA*	100%	100%	11,628
10 Missing	1.99	40%	100%	97%	603
20 Missing	3.01	60%	99%	96%	20,000
30 Missing	1.99	77%	99%	96%	1,571
40 Missing	2.54	40%	100%	88%	20,000
50 Missing	1.99	34%	100%	84%	4,165
60 Missing	1.99	37%	100%	81%	5,365

TABLE III: E2 - SUBTRACTS SMALLER FROM LARGER DIGIT

Run	Square Error	Missing Patterns Recognized	Present Patterns Recognized	Total Recognized	Iterations
All Patterns	1.99	*NA*	100%	100%	386
10 Missing	1.99	100%	100%	100%	438
20 Missing	1.99	100%	100%	100%	386
30 Missing	1.99	63%	99%	94%	568
40 Missing	1.98	70%	99%	93%	1,331
50 Missing	1.99	70%	99%	92%	1,229
60 Missing	1.99	63%	100%	89%	498

TABLE IV: E2 - THINKS $0 - n$ IS 0

Run	Square Error	Missing Patterns Recognized	Present Patterns Recognized	Total Recognized	Iterations
All Patterns	2.30	*NA*	100%	100%	20,000
10 Missing	1.96	80%	99%	99%	4,660
20 Missing	6.50	30%	97%	90%	20,000
30 Missing	1.99	60%	100%	94%	14,405
40 Missing	1.96	40%	100%	88%	2,925
50 Missing	1.97	30%	99%	82%	3,160
60 Missing	1.96	23%	99%	77%	1,772

TABLE V: E2 - AVERAGE NUMBER OF PATTERNS RECOGNIZED

Run	Missing Patterns Recognized	Present Patterns Recognized	Total Recognized
All Patterns	*NA*	99%	99%
10 Missing	63%	99%	97%
20 Missing	61%	91%	88%
30 Missing	56%	99%	92%
40 Missing	43%	99%	88%
50 Missing	42%	99%	85%
60 Missing	38%	99%	81%

determine the optimal number of hidden units. An ITS designer must be careful and be willing to take the time necessary to build a good student model.

Experiment Three: The final network is much like the previous network with the addition of another hidden layer to help generalization and of an additional neuron in the output layer indicating whether a borrow needs to be propagated to the next column on the left. It is shown in Figure 5.

The borrow in the patterns is included only if the student actually makes one. Determining when the student borrows might be hard to do, but a carefully designed interface or test could help to discover when he is. Since the borrow is included in the output layer, it could be fed into the input layer's borrow in the style of a recurrent network. Further, the output borrow neuron would indicate when the student is and is not borrowing to help to remedy errors.

Once again, the backpropagation simulator was run on a Sun SPARCstation with a limit of 20,000 iterations over the pattern set and of some square error cutoff. This network also took around two days to train for the 20,000 iterations or about 20 cpu hours. As with the previous experiment, the learning parameters were kept the same.

The data sets used for the network in this experiment are shown in Table VI. First the correct subtraction training set was used and then sixteen sets with subtraction errors were used. Finally, six sets with noise were used.

Table VII shows how the network trained and performed with correct subtraction. This network did better than the previous network both in terms of training and accuracy and seemed to perform the worst when 60 patterns were eliminated as it recog-

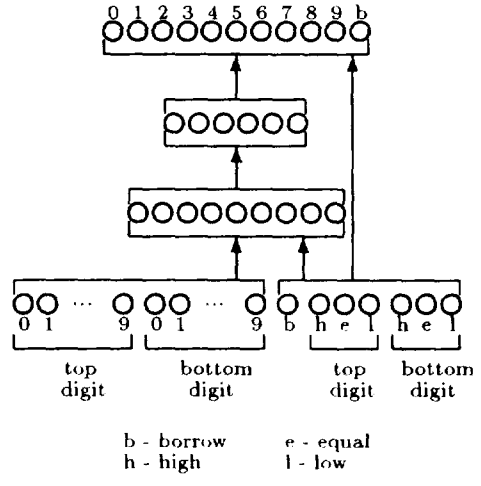

Fig. 5. Second single-column subtraction network.

TABLE VI: E3 - TRAINING PATTERN SETS

Correct Subtraction
Subtracts Smaller From Larger Digit
Thinks $0 - n$ is 0 (No Borrow)
Thinks $n - n$ is 9 (Borrow)
Writes 0 in Every Borrow Column (Borrow)
Writes 0 in Every Borrow Column; Thinks $0 - n$ is n (No Borrow)
Thinks $1 - n$ is 0 if Borrowed From (No Further Borrow)
Thinks $1 - 1$ is 0 if Borrowed From (No Further Borrow)
Thinks $n - n$ is 1 if Borrowed From (No Further Borrow)
Changes Top Number to 10 in Borrow Column (Borrow)
Changes a 1 to a 10 in Borrow Column (Borrow)
Subtracts 1 and Adds 10 in Borrow Column (Borrow)
Adds 10 in Borrow Column (No Borrow)
Borrows Across 1; Treats 1 as 0 (Borrow)
Does not Decrement Unless Bottom Smaller Than Top (Borrow)
Does not Decrement when Borrowing Across $0 - 0$ (No Borrow)
Decrements to Left of $0 - 0$, but Does not Change Top 0 (Borrow)
Correct Subtraction with No Missing Patterns (Noise Introduced)
Correct Subtraction with 10 Missing Patterns (Noise Introduced)
Correct Subtraction with 20 Missing Patterns (Noise Introduced)
Correct Subtraction with 30 Missing Patterns (Noise Introduced)
Correct Subtraction with 40 Missing Patterns (Noise Introduced)
Correct Subtraction with 50 Missing Patterns (Noise Introduced)
Correct Subtraction with 60 Missing Patterns (Noise Introduced)

TABLE VII: E3 - CORRECT SUBTRACTION

Run	Square Error	Missing Patterns Recognized	Present Patterns Recognized	Total Recognized	Iterations
All Patterns	1.99	*NA*	100%	100%	14,623
10 Missing	1.93	80%	100%	99%	1,098
20 Missing	1.98	85%	100%	99%	4,994
30 Missing	2.78	70%	100%	96%	8,898
40 Missing	1.99	75%	100%	95%	3,646
50 Missing	1.99	54%	100%	89%	5,207
60 Missing	1.96	38%	100%	82%	5,880

nized less than 50% of the missing patterns.

Tables VIII and IX show how the network performed on two errors upon which the previous network was trained. The first error involves subtracting the smaller from the larger digit and the network learned it much better than the previous one. The second error also was learned better even though it is peculiar. The student simply writes a zero for $0 - n$ and does not borrow from the column on the left.

The results of the network on the other subtraction error data sets may be found in Mengel[5]. The network performed well on recognizing these errors.

On average, the network performed satisfactorily as seen in Table X. In fact, this network did better than the previous network. Of course, it still lost accuracy when more and more patterns were eliminated, but it still recognized over 50% of the missing patterns. The results in this table are optimistic in nature since they show that the network did well. Further research looks promising.

Tables XI through XIV show how the network performed when noisy data was added to the correct subtraction pattern set (the rest of the tables may be found in Mengel[5]). These patterns differed from the original patterns in that the correct answer was changed to an incorrect answer or the propagated borrow was changed to the opposite of what it should be.

For each run, noisy patterns were added in increments of ten located randomly in the training pattern set. Up to 60 noisy patterns were included even though the event is unlikely that a student would have this many random answers in a given subtraction problem set. Necessarily, the resiliency of network must be determined to both small and large amounts of noise. Further, the results of noise when ten, twenty, thirty, forty, fifty, and sixty correct patterns are deleted from the full subtraction pattern set must be seen.

After training was completed, the full correct subtraction set was checked to see how the network would recognize the full set. The results are summarized in the tables. The network performed well when most of the correct subtraction patterns were present in the training pattern set. As more correct patterns were deleted and more noise was added, the network performance tended to degrade.

That the network recognized over 50% of the total correct subtraction pattern set on average is encouraging. Certainly, the addition of noise does have an effect on network performance, but, in reality, less noise may be present in real student data.

The results of this third experiment are far more encouraging than the previous experiment. The network is able to perform better on average than the previous network and performance does not degrade completely when noisy patterns are present in the pattern set. Likewise, noise is not handled easily in AI symbolic processing systems. While noise may be present in real student data, the network can learn the overall method of the student as shown in this experiment. With further design changes on the network, it may become very resilient to noise.

Of the 110 subtraction errors that Burton and Brown found, 16 of the errors can be handled by the network. The network can recognize more of the 110 errors, since many are similar to the errors chosen in this experiment. Errors that the network cannot deal with are those that require past and future contexts for each subtraction step. For instance, earlier in the chapter, an error is shown where the student incremented from the point of the borrow. To recognize this type of error, a network must know the column from where the borrow comes. Moving from a feedforward paradigm to a recurrent paradigm may help to recognize context sensitive errors.

In total, over 161 training runs were made with this network in order to determine its learning ability (additional runs were made in trying other networks as well). The information obtained in these runs may be used in the design of a better network.

Conclusion

The design of a neural student model must be a careful and thoughtful process. The domain must be translated into a pattern recognition task and provision for as many student answers (right or wrong) as possible must be made. The network, however, may have to undergo several changes before it characterizes a student's cognitive processes well. Certainly any software system must evolve to perform its task adequately. The design effort is worth the trouble to be able to capture each individual

TABLE VIII: E3 - SUBTRACTS SMALLER FROM LARGER DIGIT

Run	Square Error	Missing Patterns Recognized	Present Patterns Recognized	Total Recognized	Iterations
All Patterns	1.77	*NA*	100%	100%	347
10 Missing	1.99	100%	99%	99%	2,100
20 Missing	1.95	90%	100%	99%	276
30 Missing	1.98	93%	100%	99%	362
40 Missing	1.94	95%	100%	99%	495
50 Missing	1.95	84%	100%	96%	251
60 Missing	1.97	87%	100%	96%	525

TABLE IX: E3 - THINKS $0 - n$ IS 0 (NO BORROW)

Run	Square Error	Missing Patterns Recognized	Present Patterns Recognized	Total Recognized	Iterations
All Patterns	1.99	*NA*	100%	100%	1,938
10 Missing	1.92	80%	100%	99%	1,310
20 Missing	1.95	75%	100%	98%	1,626
30 Missing	1.94	73%	100%	96%	1,191
40 Missing	1.98	75%	100%	95%	1,106
50 Missing	1.88	58%	99%	89%	876
60 Missing	1.98	47%	99%	84%	1,135

TABLE X: E3 - AVERAGE NUMBER OF PATTERNS RECOGNIZED

Run	Missing Patterns Recognized	Present Patterns Recognized	Total Recognized
All Patterns	*NA*	99%	99%
10 Missing	82%	99%	99%
20 Missing	77%	99%	98%
30 Missing	70%	99%	95%
40 Missing	66%	99%	93%
50 Missing	61%	99%	90%
60 Missing	52%	99%	85%

TABLE XI: E3 - CORRECT SUBTRACTION WITH NO MISSING PATTERNS

Run	Square Error	Missing Patterns Recognized	Present Patterns Recognized	Total Recognized	Iterations
10 Added	24.77	*NA*	83%	83%	20,000
20 Added	18.23	*NA*	95%	95%	20,000
30 Added	32.84	*NA*	90%	90%	20,000
40 Added	44.69	*NA*	85%	85%	20,000
50 Added	64.32	*NA*	79%	79%	20,000
60 Added	69.35	*NA*	82%	82%	20,000

TABLE XII: E3 - CORRECT SUBTRACTION WITH 30 MISSING PATTERNS

Run	Square Error	Missing Patterns Recognized	Present Patterns Recognized	Total Recognized	Iterations
10 Added	34.44	40%	72%	67%	20,000
20 Added	21.54	53%	89%	84%	20,000
30 Added	58.04	30%	68%	63%	20,000
40 Added	55.12	37%	72%	67%	20,000
50 Added	58.56	23%	73%	66%	20,000
60 Added	62.39	30%	69%	63%	20,000

TABLE XIII: E3 - CORRECT SUBTRACTION WITH 60 MISSING PATTERNS

Run	Square Error	Missing Patterns Recognized	Present Patterns Recognized	Total Recognized	Iterations
10 Added	4.50	25%	96%	75%	20,000
20 Added	32.04	7%	77%	56%	20,000
30 Added	50.41	22%	63%	51%	20,000
40 Added	33.72	27%	79%	63%	20,000
50 Added	38.06	18%	84%	64%	20,000
60 Added	47.80	18%	74%	58%	20,000

TABLE XIV: E3 - AVERAGE BEHAVIOR OF NETWORK WITH NOISY PATTERNS

Run	Missing Patterns Recognized	Present Patterns Recognized	Total Recognized
10 Added	36%	84%	69%
20 Added	28%	81%	73%
30 Added	28%	79%	72%
40 Added	26%	75%	68%
50 Added	18%	76%	68%
60 Added	22%	75%	67%

student's procedure in order for the ITS to adapt better to a student's needs.

The experiments showed the behavior of neural networks in various situations. Experiment one demonstrated that the domain must be constrained so that the network can be trained with as few problems as one student might work. Experiment two gave more favorable results since the network was able to generalize from the training pattern set to unknown patterns. This experiment also revealed that the network has enough parameter adjustments to be able to be tailored to an individual student. With a program to vary these adjustments automatically, the designer's job would be facilitated. Experiment three yielded even more encouraging results. The network in this experiment was able to generalize even better than the network in experiment two and could handle noise with relative grace.

The results of experiments two and three indicate that neural networks are a viable tool with which to construct a student model. Neural networks are executable, can derive the overall gist of a pattern set, and can function in the presence of noisy data. They may be tailored to individual students through parameter adjustments and may be updated when new student data becomes available. In many ways, they fit the definition of the quintessential student model.

Since neural networks can generalize over a pattern set, they can be used to predict what a student might do next on a problem solving step. If a network were designed to learn the student's problem solving procedure, then it could train on the student's procedure and simply reflect that procedure on patterns it has not seen before. The network, therefore, could predict what the student might do next without having a restrictive interface that would force the student to perform a step unnatural to him as with model tracing. The student's own procedure, therefore, could be corrected, if necessary, to help the student to learn the domain better.

The next step in neural student modeling is to place a neural network into the student model of an intelligent tutoring system. The advantages and disadvantages of the neural student model in a working ITS need to be compared to the advantages and disadvantages of existing student models. This step must be taken in order to refine the neural student model and to make it more efficient at its task.

Acknowledgements

The authors wish to thank General Dynamics of San Diego, California, for helping to fund the research reported in this paper.

References

[1] R. Beale and J. Finlay, "User modelling with a neural system," Technical Report YCS118, Department of Computer Science, University of York, Heslington, York, England, 1989.

[2] J. S. Brown and R. R. Burton, "Diagnostic models for procedural bugs in basic mathematical skills," *Cognitive Science*, vol. 2, pp. 155-192, 1978.

[3] W. J. Clancey, "Qualitative student models," Technical Report STAN-CS-87-1171, Department of Computer Science, Stanford University, Stanford, CA, May 1986.

[4] J. E. Friend and R. R. Burton, *Teacher's Guide for Diagnostic Testing in Arithmetic: Subtraction*. Palo Alto, CA: Cognitive and Instructional Sciences, Xerox Palo Alto Research Center, 1980.

[5] S. Mengel, *Using Neural Networks to Predict Student Responses in Intelligent Tutoring Systems*. Doctoral Dissertation, Texas A&M University, Computer Science Dept., 1990.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Volume I*, D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. Cambridge, MA: The MIT Press, 1986, pp. 318-362.

[7] P. K. Simpson (General Dynamics, San Diego, CA), "A review of artificial neural systems I: foundations," *CRC Critical Reviews in Artificial Intelligence*, in review, May 1988.

[8] J. Self, "Bypassing the intractable problem of student modelling," in *Intelligent Tutoring Systems*. Montréal, Quebec, Canada: Université de Montréal, 1988, pp. 18-24.

[9] D. Sleeman and J. S. Brown, "Introduction: intelligent tutoring systems," in *Intelligent Tutoring Systems*, D. Sleeman and J. S. Brown, Eds. Orlando, FL: Academic Press, 1982, pp. 1-11.

[10] G. Tesauro and T. J. Sejnowski, "A parallel network that learns to play backgammon," *Artificial Intelligence*, vol. 39, pp. 357-390, 1989.

[11] K. VanLehn, "Student modeling," in *Foundations of Intelligent Tutoring Systems*, M. C. Polson and J. J. Richardson, Eds. Hillsdale, NJ: Lawrence Erlbaum Associates, 1988, pp. 55-78.

[12] P. D. Wasserman, *Neural Computing Theory and Practice*. New York: Van Nostrand Reinhold, 1989.

[13] E. Wenger, *Artificial Intelligence and Tutoring Systems*. Los Altos, CA: Morgan Kaufmann Publishers, 1987.

[14] R. Zerwekh, "Classifying competence levels using adaptive resonance theory: modeling learner performance," in *Fourth Conference on Neural Networks and Parallel Distributed Processing*. Indiana University and Purdue University, April 1991.