CRCD Experiences at the University of Central Florida: An NSF Project

Michael Georgiopoulos[*], Jose Castro[*], Erol Gelenbe[**], Ronald Demara[*], Avelino Gonzalez[*], Marcella Kysilka[*], Mansooreh Mollaghasemi[*], Annie Wu[*], Ingrid Russell[***]

(*) University of Central Florida
(**) University of Central Florida and Imperial College
(***) University of Hartford

Abstract
Machine Learning has traditionally been a topic of research and instruction in computer science and computer engineering programs. Yet, due to its wide applicability in a variety of fields, its research use has expanded in other disciplines, such as electrical engineering, industrial engineering, civil engineering, and mechanical engineering. Currently, many undergraduate and first-year graduate students in the aforementioned fields do not have exposure to recent research trends in Machine Learning. This paper reports on a project in progress, funded by the National Science Foundation under the program *Combined Research and Curriculum Development (CRCD)*, whose goal is to remedy this shortcoming. The project involves the development of a model for the integration of Machine Learning into the undergraduate curriculum of those engineering and science disciplines mentioned above. The goal is increased exposure to Machine Learning technology for a wider range of students in science and engineering than is currently available. Our approach of integrating Machine Learning research into the curriculum involves two components. The first component is the incorporation of Machine Learning modules into the first two years of the curriculum with the goal of sparking student interest in the field. The second is the development of new upper level Machine Learning courses for advanced undergraduate students. The paper will focus on the details of the integration of a machine learning module (related to neural networks) applied to a Numerical Analysis class, taught to sophomores and juniors in the Engineering Departments at the University of Central Florida. Furthermore, it will report results on the assessment and evaluation of the effectiveness of the module by the students taking the class. Finally, based on the assessment results some conclusions will be drawn regarding the potential of the modules in attracting undergraduate students into research, specifically machine-learning research.

1. Introduction

This paper reports on the progress of a grant supported by the National Science Foundation under the auspices of the program entitled Combined Research and Curriculum Development (CRCD). The title of the grant is “Machine Learning Advances for Engineering Education” and its purpose is to introduce Machine Learning research into the undergraduate science and engineering curriculum. Our proposal relies on two simple ideas to achieve its goals. The first idea is the introduction of machine learning modules in the sophomore and junior level required classes. A machine-learning module is a homework assignment that has a machine learning flavor to it, while at the same time achieves the goals of the homework assignment for the class under consideration. The objective of the machine-learning module is to spark the interest in machine learning of a large number of undergraduate students at the early stage of their studies. This way a number of them will be motivated to take our senior-level Machine Learning classes. Introducing senior-level Machine Learning classes is the second idea of our proposed work in the effort to achieve the goals of this grant. These senior Machine Learning classes are called *Current Topics in Machine Learning I* (CTML-I) and *Current Topics in Machine Learning II* (CTML-II). In CTML-I the PIs in this grant (2 from CS, 2 from CpE, 1 from EE, and 1 from IEMS) have the opportunity to expose the students to current machine learning research topics related to their research experiences. After the students are exposed to the PIs' research in the CTML-I class they have the opportunity to work on an appropriately assigned machine-learning project in CTML-II. Through the combined work in CTML-I and CTML-II classes we are anticipating that the students will not only learn about current machine learning research topics but they will also be exposed to "hands-on" research in this field. The belief is that through the CRCD's cumulative experiences undergraduate students will be offered the opportunity to appreciate machine learning research, produce results of their own in an appropriately assigned research project, and hopefully be motivated to pursue graduate studies in this research area, or other research areas of interest to them.

This paper focuses on the work conducted towards the creation and delivery of a Machine Learning module in the Numerical Methods class taught for EE and CpE majors at the University of Central Florida. The module deals with the perceptron and pocket machine learning algorithms. Both algorithms were designed to solve classification problems that are linearly separable (i.e., problems for which the decision boundaries between data belonging to different classes are hyperplane structures). The lecture produced to explain this assignment focuses on a quick overview of the neural network field, a concise discussion of the perceptron and the pocket learning algorithms and an elaboration of their differences. The students were requested to write code (in MATLAB®) that implements each one of these algorithms. They were also asked to experiment with these algorithms on some simple problems (e.g., XOR, parity problem) to compare their performances. Performance comparison measures used in this study were the convergence speed of the algorithm to a solution, and the accuracy of the solution obtained. The students were also requested to evaluate the effectiveness of the module in achieving the goals of the class and the goals of the CRCD project. This module was taught and evaluated twice (Spring 03 and Summer 03). The paper elaborates on the material taught for this module, the details of the module assignment, the work that the students had to complete, and the evaluation of this experience by the students

2. Machine Learning Module in EGN3420 (Engineering Analysis class)

The EGN3420 Engineering Analysis is an applied numerical methods course at the University of Central Florida that concentrates on the topics of absolute and relative errors, Taylor series expansion, one–dimensional root finding techniques, introduction to matrices and determinants, computer solutions to linear equations, Gauss–Jordan, curve fitting, numerical integration and differentiation, interpolation and solutions to differential equations.

The course uses the MATLAB® programming environment and expects that students registering for this course have had some previous programming experience. A series of homeworks are assigned to the students that have to be programmed in MATLAB®. The homework requires considerable amount of time investment from the student. It is worth noting though that these homeworks do not constitute a significant portion of the student's grade.

The module described in this paper uses a neural network learning technique, called perceptron learning (see Rosenblatt, 1962). It learns to associate input patterns with their corresponding label, a feat that the neural network accomplishes through an appropriately designed learning process. During this process the neural network is taught to map certain input patterns to their associated labels. The module assignment, can be used to introduce the concepts of MATLAB® programming, matrix operations and gradient descent optimization methods and therefore is compatible with the learning objectives of the course. It was assigned for the first time in the spring semester of 2003, and then subsequently in the summer of 2003. In both instances the module was assessed and evaluated by the students, and the results of the spring 2003 assessment were used to modify the teaching materials and delivery from one semester to the next with the intent of improving the module's usefulness. The main focus of the learning module is the pocket algorithm. A condensed version of the handout given to the students and an explanation of the pocket algorithm is given in the following section.

4. Perceptron–Based Learning Algorithms Module:

The sections below describe in sufficient detail the module assignment that was handed out to students and the associated teaching material. The teaching material was necessary for the students to understand perceptron learning algorithms, and eventually implement one of the perceptron learning variants, called "pocket algorithm". Portions of the module were discussed in class.

4.1 The Module Assignment

1. Write the code for the single layer perceptron algorithm. Incorporate enough explanatory statements to explain what each part of your code is doing. Define the variables that you are using within your code at the beginning of your code. Test your code with the Parity-5 problem.
2. Write the code for the pocket algorithm in MATLAB®. Incorporate enough explanatory statements to explain what each part of your code is doing. Define the variables that you are using within your code at the beginning of your code. Test your code with the Parity-5 problem.
3. Compare the performance of the perceptron learning algorithm and the pocket algorithm on the Parity-5 problem, and another two-class classification problem of your choice.

What you should turn in (Deliverables)
The deliverables for this homework are:

- 3.5” 1.44 Mbyte disk containing
  - The source code in a file called hw?XXXX where ? is the number of your homework and XXXX should be the last four digits of your SS#. If you have more than one source file then name them hw?XXXXa, hw?XXXXb etc. The main function should be in hw?XXXXa.
    - Source code should have comments, guidelines for correct commenting of source code shall be given in class.
  - The executable (.exe) of the program.
- A write up with:
  - A title page with your name and SS# and Homework number.
  - An index.
  - A problem statement (portions of this write up).
  - A program description stating the important algorithms in your homework.
  - A programmer’s log.
  - Comments (feedback)

The programmer’s log is a table that displays the activities that you incurred in this homework and the time spent on each of the tasks, the format for the table should be:

Activity	Date/Time Start	Date/Time End	Total Time	Comments
Designing				
Coding				
Debugging				
Coding				

Try to be as honest as possible in the entries of this table, you will not be graded for it, it is for feedback purposes.

Grading Policy

- Program 70 %
- Write up 30 %

Sample student code that was submitted to the instructor of the class, as part of the requirements of this assignment, is included in Appendix I.

4.2 Objective

One of the purposes of the perceptron machine–learning module assignment is for the students to understand the basics of neural networks, such as definition, neuronal models, popular neural network structures, and their utility. Another purpose is to understand the specifics of the single layer perceptron architecture, the perceptron learning algorithm and the pocket algorithm. Furthermore, they should be able to learn, if they do not know it already, how to read vectors/matrices, compute inner products, use “for loops” and “if then else” statements in a programming language of their choice. Finally, they are supposed to run some experiments with the perceptron and pocket learning algorithms and compare their performance on appropriate data sets.

4.3 Teaching Material: Preliminaries on Neural Networks

A neural network (NN) is a network of many simple processors (units, nodes, neurons), each one of which has a small amount of local memory. Unidirectional channels (connections) that carry numerical data connect these processors. A neural network resembles the human brain in two respects:

1. Knowledge is acquired by a network through a learning process.
2. Interneuron connection strengths known as synaptic weights are used to store the knowledge.

The model of a neuron is pictorially shown in Figure 1. We may describe neuron *j* through a set of equations that describe its net input (net_j) and output (y_j)as follows:

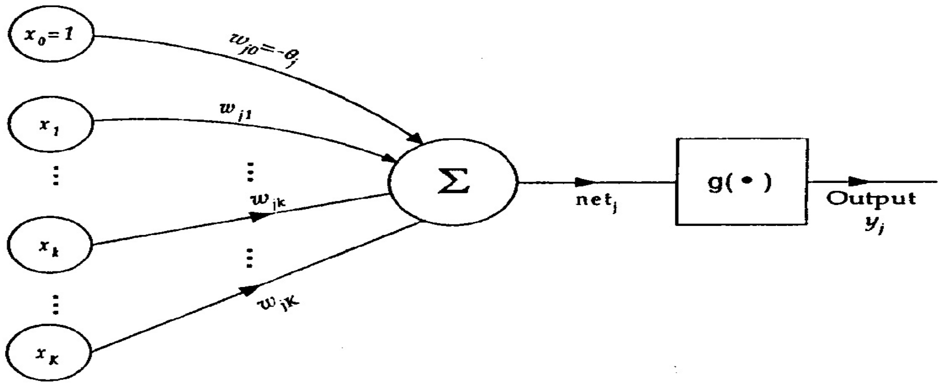

Figure 1: Model of a neuron

$$net_j = \sum_{k=0}^{K} w_{jk}\, x_k$$
$$y_j = g(net_j)$$

where $x_1, x_2, ..., x_K$ are the input signals; $w_{j1}, w_{j2}, .., w_{jK}$ are the synaptic weights converging to neuron *j*; net_j is the cumulative effect of all the neurons connected to neuron *j* and the internal threshold of neuron *j*; $g(\cdot)$ is the activation function of neuron *j*; and y_j is the output signal of neuron *j*. Two examples of typical activation functions *g* are shown in Figures 2 and 3.

4.4 Teaching Material: Single Layer Perceptron Neural Network

The simplest possible layered neural network is the single layered neural network that consists of a layer of input nodes (input layer) and a layer of output nodes (output layer). Figure 4 depicts a single layered neural network architecture consisting of *K* input nodes and *I* output nodes. We call it a single-layered neural network (or single-layer perceptron) because only the nodes in the output layer are neurons.

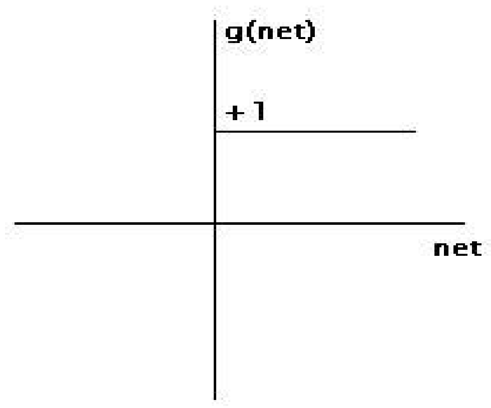

Figure 2: Threshold activation function

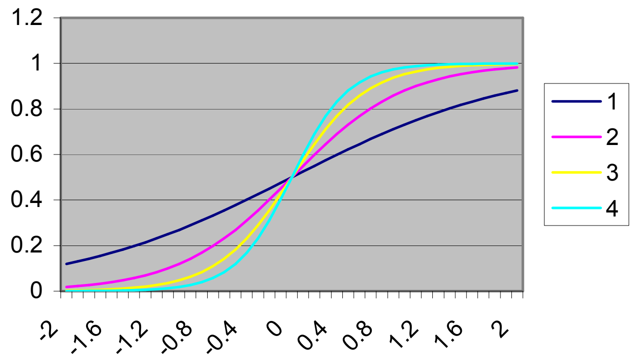

Figure 3: Sigmoid activation function $1/(1+e^{-ax})$ with varying slope parameter a=1,2,3,4

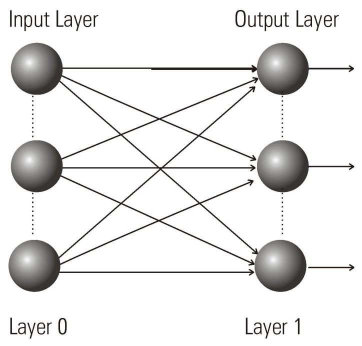

Figure 4: Single layer perceptron architecture

Once the neuronal model has been defined and the neural network structure has been chosen an appropriate learning algorithm is needed. The goal here is to establish the correct mapping between a set of inputs $\{\mathbf{x}(p)\}_{p=1}^{PT}$ and a set of corresponding outputs $\{\mathbf{d}(p)\}_{p=1}^{PT}$. This collection of

x's and **d**'s is called a training collection and the phase that the network is going through to establish the correct mapping between **x**'s and **d**'s is called *training phase*. For the single layer perceptron NN, Rosenblatt (Rosenblatt, 1962) has introduced an algorithm that establishes all the required mappings, more than 40 years ago. This algorithm is known by the name *perceptron learning algorithm*. The assumption made with this algorithm is that the activation function is of the threshold type (e.g., the output of a neuron assumes two distinct values +1 or –1) and that the data given to us are linearly separable. It is worth mentioning that the constraint that the data need to be linearly separable is a strong constraint, and as a result there are many useful problems that the single-layer perceptron and the perceptron learning algorithm cannot solve (see Minsky and Papert, 1969). Nevertheless, the single layer perceptron is one of the simplest neural network architectures and capable of solving problems that are linearly separable or almost linearly separable.

4.5 Teaching Material: Perceptron Learning Algorithm, Pocket Algorithm

The perceptron learning algorithm is described in a sequence of steps below.

Step 1: Initialize the weights and the bias converging to node *i* (i.e., the weights w_{ik}^1 's for $0 \le k \le K$). The weights are initialized to small random values. The value of the index *p* is initialized to 1.

Step 2: Present the input pattern $\mathbf{x}(p)$ from the training collection.

Step 3: Calculate the output of node *i* due to the presentation of pattern $\mathbf{x}(p)$, by using the following equation:

$$y_i^1(p) = g(net_i^1(p)) = g\left[\sum_{k=0}^{K} w_{ik}^1 \, x_k(p)\right]$$

Step 4: Check to see whether the actual output $y_i^1(p)$ matches the desired output $d_i(p)$. We now distinguish three cases:

- Step 4a: If $y_i^1(p) = d_i(p)$ then move to Step 5.
- Step 4b: If $y_i^1(p) = 1$ and $d_i(p) = -1$, then modify the weight vector $\mathbf{w}_i$ converging to node *i* by $\Delta\mathbf{w}_i$, where
$$\Delta\mathbf{w}_i = -\mathbf{x}(p)$$
- Step 4c: If $y_i^1(p) = -1$ and $d_i(p) = +1$, then modify the weight vector $\mathbf{w}_i$ converging to node *i* by $\Delta\mathbf{w}_i$, where
$$\Delta\mathbf{w}_i = +\mathbf{x}(p)$$

Step 5: We now distinguish two cases: (a) If $p \neq PT$, go to step 2 and present the next in sequence input pattern, and (b) If $p = PT$ and the weights did not change in the last PT presentations of the input patterns, then the training is complete. If on the other hand, the weights changed at least once in the previous PT list presentations, then we go to step 2 to present the first in sequence input pattern corresponding to index p=1.

As Gallant points out in his paper (Gallant 1990) perceptron learning is not very well behaved for non-separable problems. While it will eventually visit an optimal set of weights, it will not converge to *any* set of weights. Even worse the algorithm can go from an optimal set of weights to a worst-possible set in one iteration, regardless of how many iterations have been taken previously. The pocket algorithm makes perceptron learning well behaved by adding positive feedback in order to stabilize the algorithm.

The basic idea of the pocket algorithm is to run perceptron learning while keeping an extra set of weights "in your pocket". Whenever the perceptron weights have a longest run of consecutive correct classifications of randomly selected training patterns than the pocket weights, these perceptron weights replace the pocket weights. The pocket weights are the outputs of the algorithm.

4.6 Teaching Material: Perceptron Learning Algorithm, and Pocket Algorithm on the XOR Problem

As an example of the perceptron learning and the pocket algorithm consider the XOR problem with the following training examples:

$$\begin{aligned} \mathbf{x}(1) &= (+1 -1 -1) & d(1) &= -1 \\ \mathbf{x}(2) &= (+1 -1 +1) & d(2) &= +1 \\ \mathbf{x}(3) &= (+1 +1 -1) & d(3) &= +1 \\ \mathbf{x}(4) &= (+1 +1 +1) & d(4) &= -1 \end{aligned}$$

The first entry of every training example above is +1 because it corresponds to the constant bias input. Figure 5 gives a typical sequence of iterations. At iteration 8 both pocket and perceptron weights are optimal, classifying three out of the four examples correctly. But at iteration 9 the initial perceptron weights (0, 0, 0) have been reached. These misclassify every training example whereas the pocket weights, (1 -1 -1), still get three out of the four correct. As iterations continue changes to the pocket weights will become less and less frequent. The pocket algorithm can operate in an off-line training mode or an in an on-line training mode. In the off-line training mode we have a fixed collection of input/output pairs that is given to us and data from this collection are picked randomly to be presented at the input nodes of the SLP-NN architecture. In the on-line training mode we do not have a fixed collection of input/output pairs and the data come in, one at a time. The SLP-NN is trained on the data as they come in and no storage of data is feasible in the on-line training mode. In the original paper by Gallant the on-line training mode is referred to as training with ∞ of training data. The pseudo-code of the "pocket algorithm" for the on-line training phase is given below. The off-line training phase can be found in Gallant, 1990. Note that π stands for the vector of current perceptron weights, *W* stands for the vector of pocket weights, run_π stands for the number of consecutive correct classifications using the perceptron weights π, run_W is the number of consecutive correct classifications using the pocket weights *W*.

Start with an initial current weight vector $\pi = (0,0,...,0)$

```
Randomly pick a training example x(p) with desired classification d(p)
If π correctly classifies x(p) then
        $run_\pi = run_\pi + 1$
        If $run_\pi > run_w$ then

                $W = \pi$
                $run_W = run_\pi$

Otherwise, form a new vector π of perceptron weights by following the perceptron algorithm.
Set $run_\pi = 0$.
End of this iteration. If the specified number of iterations has not been taken then proceed with
the presentation of another randomly chosen training example. The above procedure is repeated.
```

4.6 Teaching Material: Verification of the Pocket Calculations for the XOR Problem

Iteration 1: Present pattern **x** (4). Current initial weight vector is $\pi = (0, 0, 0)$. Is x (4) correctly classified ? No, because $\mathbf{x}^T(4)\,\pi = 0$, while it should have been negative. Hence, π changes to $\pi - \mathbf{x}(4) = (-1, -1, -1)$.

Iteration 2: Present pattern **x** (4). Current weight vector is $\pi = (-1, -1, -1)$. Is **x** (4) correctly classified? Yes, because $\mathbf{x}^T(4)\,\pi = -3 < 0$, as it should be to produce an output of -1. Hence, π remains unchanged at $\pi = (-1, -1, -1)$. Furthermore, Run_π increases by one, thus becoming equal to 1. Also, since $Run_\pi = 1 > Run_W = 0$, *W* becomes equal to $\pi = (-1, -1, -1)$, and also Run_W becomes equal to $Run_\pi = 1$.

Iter.	π	run_π	W	run_w	Choice	OK?	Action
1.	$<0\ \ 0\ \ 0>$	0	$<0\ \ 0\ \ 0>$	0	***x(4)***	no	$\pi = \pi - \mathbf{x}(4)$ $Run_\pi = 0$
2.	$<-1\ \ -1\ \ -1>$	0	$<0\ \ 0\ \ 0>$	0	***x(4)***	yes	$Run_\pi = Run_\pi + 1$ $W = \pi$ $Run_w = Run_\pi$

					x(2)	no	$\pi = \pi + \mathbf{x}(2)$ $Run_\pi = 0$
3.	<-1 -1 -1 >	1	<-1 -1 -1 >	1			
					x(3)	no	$\pi = \pi + \mathbf{x}(3)$ $Run_\pi = 0$
4.	< 0 -2 0 >	0	<-1 -1 -1 >	1			
					x(4)	yes	$Run_\pi = Run_\pi + 1$
5.	< 1 -1 -1 >	0	<-1 -1 -1 >	1			
					x(2)	yes	$Run_\pi = Run_\pi + 1$ $W = \pi$ $Run_w = Run_\pi$
6.	< 1 -1 -1 >	1	<-1 -1 -1 >	1			
					x(3)	yes	$Run_\pi = Run_\pi + 1$ $Run_w = Run_\pi$
7.	<1 -1 -1 >	2	<1 -1 -1 >	2			
					x(1)	no	$\pi = \pi - \mathbf{x}(1)$ $Run_\pi = 0$
8.	<1 -1 -1 >	3	<1 -1 -1 >	3			
					…		
9.	<0 0 0 >	0	< 1 -1 -1 >	3			

Figure 5: Perceptron and Pocket algorithm iterations for the XOR problem

Iteration 3: Present pattern $\mathbf{x}(2)$. Current weight vector is $\pi = (-1, -1, -1)$. Is $\mathbf{x}(2)$ correctly classified ? No, because $\mathbf{x}^T(2)\,\pi = -1 < 0$, while it should have been positive to produce an output of $+1$. Hence, π changes to $\pi + \mathbf{x}(2) = (0, -2, 0)$, and Run_π is reset back to the value of 0.
Iteration 4: Present pattern $\mathbf{x}(3)$. Current weight vector is $\pi = (0, -2, 0)$. Is $\mathbf{x}(3)$ correctly classified ? No, because $\mathbf{x}^T(3)\,\pi = -2 < 0$, while it should have been positive to produce an output of $+1$. Hence, π changes to $\pi + \mathbf{x}(3) = (+1, -1, +1)$, and Run_π is reset back to the value of 0.
Iteration 5: Present pattern **x** (4). Current weight vector is $\pi = (+1, -1, +1)$. Is **x** (4) correctly classified? Yes, because $\mathbf{x}^T(4)\,\pi = -1 < 0$, as it should be to produce an output of -1. Hence, π remains unchanged at $\pi = (+1, -1, -1)$. Furthermore, Run_π increases by one, thus becoming equal to 1. Also, since $Run_\pi = 1$ is not greater than $Run_W = 1$, W stays equal to $\pi = (-1, -1, -1)$.
Iteration 6: Present pattern **x** (2). Current weight vector is $\pi = (+1, -1, -1)$. Is **x** (2) correctly classified? Yes, because $\mathbf{x}^T(2)\,\pi = -1 > 0$, as it should be to produce an output of $+1$. Hence, π remains unchanged at $\pi = (+1, -1, -1)$. Furthermore, Run_π increases by one, thus becoming equal to 2. Also, since $Run_\pi = 2 > Run_W = 1$, W becomes equal to $\pi = (+1, -1, -1)$, and also Run_W becomes equal to $Run_\pi = 2$.
Iteration 7: Present pattern **x** (3). Current weight vector is $\pi = (+1, -1, -1)$. Is **x** (3) correctly classified? Yes, because $\mathbf{x}^T(3)\,\pi = +1 > 0$, as it should be to produce an output of $+1$. Hence,

π remains unchanged at $\pi = (+1, -1, -1)$. Furthermore, Run_π increases by one, thus becoming equal to 3. Also, since $Run_\pi = 3 > Run_W = 2$, Run_W becomes equal to $Run_\pi = 3$. No need to change W because W is already equal to $\pi = (+1, -1, -1)$.
Iteration 8: Present pattern $\mathbf{x}(1)$. Current weight vector is $\pi = (+1, -1, -1)$. Is $\mathbf{x}(1)$ correctly classified ? No, because $\mathbf{x}^T(1)\,\pi = +3 > 0$, while it should have been negative to produce an output of -1. Hence, π changes to $\pi - \mathbf{x}(1) = (0, 0, 0)$, and Run_π is reset back to the value of 0. W remains intact and equal to $(+1, -1, -1)$, and Run_W remains unchanged and equal to 3.
Iteration 9: At this iteration the current weight vector of the perceptron learning algorithm is equal to the all zeros vector, thus misclassifying all patterns, while the stored pocket weight vector $W = (+1, -1, -1)$ correctly classifies 3 out of the 4 vectors. This is the advantage of the pocket algorithm, compared to the perceptron algorithm.

4.7 Discussion Issues

The module allowed discussion of various issues related to Machine Learning including: (a) Gradient descent mathematical procedures for optimization, (b) Neural network training and testing. (c) Linearly separable versus non–linearly separable problems, and (d) Convergence issues in neural network training.

5. Assessment and Evaluation of the Module

A questionnaire was handed out to the students at the end of the Spring 2003 semester. This questionnaire served the purpose of evaluating and assessing the effectiveness of the module. A copy of this questionnaire is included in Appendix II. The questionnaire was divided into two major sections. The first section assessed the knowledge that was transferred to the students pertaining to what neural networks are, what they can do, the specifics of perceptron learning and pocket learning algorithms, and their similarities and differences. The second section asked the students to assess the effectiveness of the module in terms of difficulty, usefulness in helping them learn the material of the class, and whether it sparked their interest in learning more about machine learning. The results of the second section of the questionnaire were tabulated, and provided below.

Results for Spring 2003

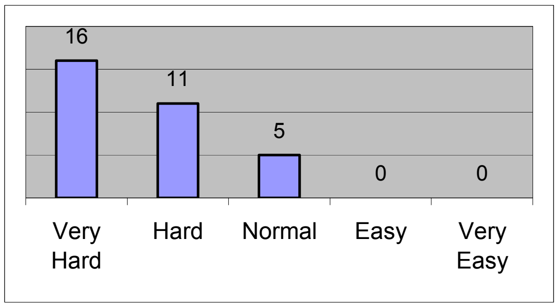

Figure 6: How hard did you find it?

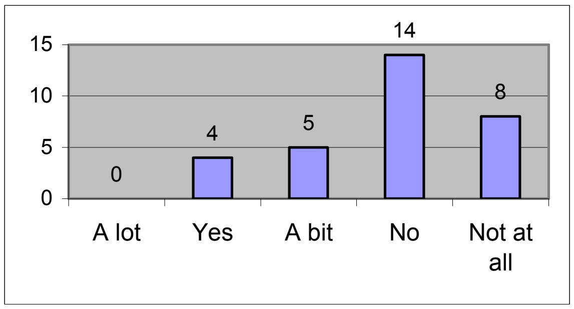

Figure 7: Did you like the module?

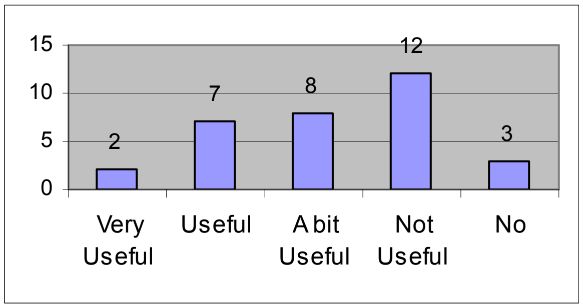

Figure 8: How useful did you find it?

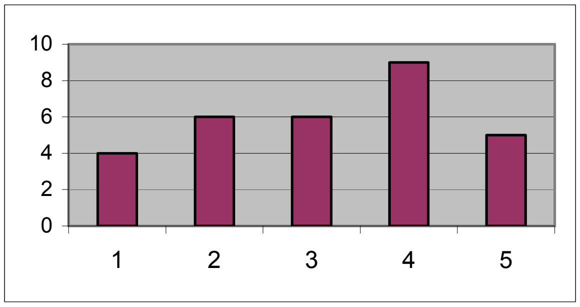

Figure 9: Did it show you ML (1/No–5/Yes))?

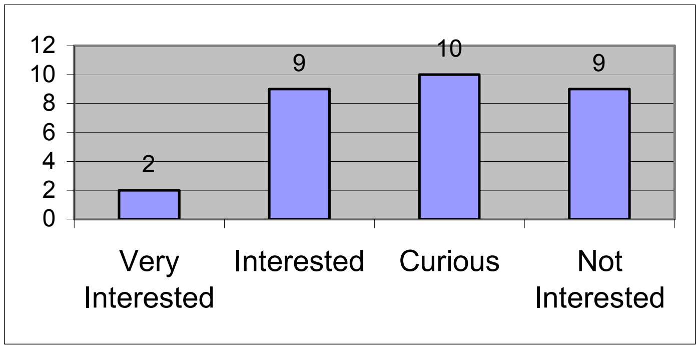

Figure 10: Are you interested in ML?

In review, students seemed to find the module very hard, and this reflected negatively on how much they liked it. Strangely enough, they did grade the module well in terms of its relationship with machine learning and about a third of the class demonstrated interest in learning more about the subject.

Modifications were applied to the module to try to make it more palatable to the students for the summer of 2003. The original module specification was broken down into 3 sections:

1. A theoretical neural networks handout.
2. A detailed explanation of the perceptron learning and pocket algorithm.
3. A module handout specifying grading and deliverables.

Originally (in Spring 2003) all these sections were lumped together and the students were handed a 10-12 pager handout, as part of their homework assignment. After reviewing the feedback from the spring 2003 experience we felt that it was best to separate the handout in 3 different sections, as described above. Furthermore, more time was dedicated in class to explain the concepts of neural networks and the corresponding pocket and perceptron learning algorithms. Despite these efforts the perception of the students regarding the module's difficulty did not change substantially. Since, the Spring 2003 student feedback, illustrated in the graphs above, was very similar with the Summer 2003 student feedback, we are omitting the details of the Summer 2003 student feedback. Once more, the students rated the module high in terms of how much it is related to Machine Learning and more than a third said they would be interested or very interested in knowing more about the subject of Machine Learning.

6. Comments/Conclusions

While the sample data is small at this time, students seemed to indicate concern about the complexity of the homework. There seems to be a correlation between the programming experience of the students and their acceptance of the module. The students of the numerical methods course come from various backgrounds and not all of them feel comfortable with programming. We feel that this has impacted negatively on the students' perception of the module. Nevertheless their overall interest in the subject was positive, even though most of them found it challenging. One has to keep in mind that the purpose of the module is to make students aware of Machine Learning and its potential of solving engineering problems, with the ultimate intent of convincing some of these students to register for the Current Topics in Machine Learning I (where the learn a lot more about Machine Learning) and eventually to register for the Current Topics in Machine Learning II, where they will have a hands-on experience with Machine Learning research. Based on the module evaluation we feel that this module served this purpose (of generating student interest in Machine Learning).

In this paper, we presented an overview of some of our experiences with an NSF project in progress, whose goal is to integrate research results in Machine Learning into the undergraduate and first year graduate engineering and science curriculum. We focused on one of the methods that we have pursued to achieve this goal (that of the machine learning modules). This method relies on incorporating simple but effective Machine Learning modules in appropriate introductory computer programming classes of the engineering and computer science curriculum with the goal of attracting some of these students into the senior (newly introduced) machine learning classes. In the senior classes the students will be exposed to machine learning research and they will also be requested to participate in machine learning research. It has been our experience that despite the inherent difficulties of incorporating modules (of machine learning content) in existing sophomore and junior classes, difficulties that were conveyed to us by the students exposed to the pocket algorithm module, this approach is one of the many working approaches that we are currently pursuing to attract undergraduate students in our machine-learning research.

References

1. Gallant, I., "Perceptron_Based Leraning Algorithms," IEEE Transcations on Neural Networks, Vol. 1, No. 2, pp. 179-191, 1990.

2. Minsky, M., and Papert, S., *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA, 1969.

3. Rosenblatt, F., *Principles of Neurodynamics: Perceptrons and the theory of brain machines*, Spartan Books, Washington, D.C., 1962.

4. Rumelhart, D. E., Hinton, G. E., Williams, R. J., “Learning internal representations by error propagation,” *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, Foundations, D. E. Rumelhart and J. L. McClelland, editors, pp. 318-362, Cabridge, MS: MIT Press, 1986.

Acknowledgement

The authors from the University of Central Florida acknowledge the partial support from NSF through a CRCD grant number 0203446 entitled "Machine Learning Advances for Engineering Education".

Biographical Information

MICHAEL GEORGIOPOULOS is a Professor of the School of Electrical Engineering and Computer Science at the University of Central Florida. His research interests lie in the areas of neural networks and applications of neural networks in pattern recognition, image processing, smart antennas and data-mining. He is an Associate Editor of the IEEE Transactions on Neural Networks since 2001.

INGRID RUSSELL is a Professor of Computer Science at the University of Hartford. Her research interests are in the areas of artificial neural networks, pattern recognition, semantic web technologies, and computer science education. She has been involved in several computer science curriculum projects. Most recently she chaired the Intelligent Systems focus group of the IEEE-CS/ACM Task Force on Computing Curricula 2001.

JOSE CASTRO is currently a Ph.D. student in the computer Engineering program of the University of Central Florida. His current area of interest is neural networks, parallel algorithms and data-mining. His Ph.D. topic is “Modification of the ARTMAP algorithm for efficient parallel processing on large data-sets.

ANNIE WU is an Assistant Professor at the School of Electrical Engineering and Computer Science at the University of Central Florida. Her research interests are in the areas of genetic algorithms, machine learning, biological modeling, and visualization.

RONALD DEMARA is an Associate Professor at the School of Electrical Engineering and Computer Science at the University of Central Florida. He has been a reviewer for National Science Foundation, Journal of Parallel and Distributed Computing, IEEE Transactions on Parallel and Distributed Computing. His interests lie in the areas of Parallel and distributed processing, self-timed architectures.

AVELINO GONZALEZ is a Professor of the School of Electrical Engineering and Computer Science at the University of Central Florida. He has co-authored a book entitled, “The Engineering of Knowledge-Based Systems: Theory and Practice”. His research interests lie in the areas of artificial intelligence, context based behavior and representation, temporal reasoning, intelligent diagnostics and expert systems.

MARCELLA KYSILKA is a Professor and Assistant Chair of the Education Foundations Department at the University of Central Florida. She is active in her professional organizations and currently serves as Associate Editor of the "Journal of Curriculum and Supervision" (the scholarly journal of the Association for Supervision and Curriculum Development). Her research interests are in curriculum studies.

EROL GELENBE is a Professor and Director of the School of Electrical Engineering and Computer Science and an Associate Dean of the College of Engineering and Computer Science at the University of Central Florida. He is a Fellow of IEEE and a Fellow of ACM. His research interests cover packet network design, computer performance analysis, artificial neural networks and simulation with enhanced reality.

MANSOOREH MOLLAGHASEMI is an Associate Professor at the Industrial Engineering and Management Sciences (IEMS) Department at the University of Central Florida. She has co-authored three books in the area of Multiple Objective Decision Making. Her research interests lie in Simulation Modeling and Analysis, Optimization, Multiple Criteria Decision Making, Neural Networks and Scheduling.

Appendix I

Student Sample Code for the Pocket Algorithm Homework

```
function parity5();
% Runs the pocket algorithm for the parity5 problem.

% Initialize the values
weight = [0 0 0 0 0 0];
pi = [0 0 0 0 0 0];
runPi = 0;
runW = 0;
% Parity5 function as an array (truth table)
% where d(x) is the desired output.
%    [bias  x1  x2  x3  x4  x5  d(x)]
X = [1 -1 -1 -1 -1 -1  1
     1 -1 -1 -1 -1  1 -1
     1 -1 -1 -1  1 -1 -1
     1 -1 -1 -1  1  1  1
     1 -1 -1  1 -1 -1 -1
     1 -1 -1  1 -1  1  1
     1 -1 -1  1  1 -1  1
     1 -1 -1  1  1  1 -1
     1 -1  1 -1 -1 -1 -1
     1 -1  1 -1 -1  1  1
     1 -1  1 -1  1 -1  1
     1 -1  1 -1  1  1 -1
     1 -1  1  1 -1 -1  1
     1 -1  1  1 -1  1 -1
     1 -1  1  1  1 -1 -1
     1 -1  1  1  1  1  1
     1  1 -1 -1 -1 -1 -1
     1  1 -1 -1 -1  1  1
     1  1 -1 -1  1 -1  1
     1  1 -1 -1  1  1 -1
     1  1 -1  1 -1 -1  1
     1  1 -1  1 -1  1 -1
     1  1 -1  1  1 -1 -1
     1  1 -1  1  1  1  1
     1  1  1 -1 -1 -1  1
     1  1  1 -1 -1  1 -1
     1  1  1 -1  1 -1 -1
     1  1  1 -1  1  1  1
     1  1  1  1 -1 -1 -1
     1  1  1  1 -1  1  1
     1  1  1  1  1 -1  1
     1  1  1  1  1  1 -1];

iter = input('Choose how many iterations: ');
r_menu = input('Type "1" for Interactive, "2" for Computer only: ');

while (iter > 0)

    % Select rows interactively or randomly.
    if (r_menu == 1)
        row = input('Choose a row for input (1-32): ');
    else
        row = ceil(mod(randn(1)*1000, 32));
    end

    w_transpose = calc_weight(pi, X(row,1:6), 6);
    % Good classification, set our pocket values to the
    % current values.
    if ((w_transpose > 0) & (X(row,7) > 0)) | ((w_transpose < 0) & (X(row,7) < 0))
        disp('Good classification');
        runPi = runPi + 1;
        if (runPi > runW)
            weight = pi;
            runW = runPi;
        end
    % Bad classification, reset runPi to 0 and adjust pi
    % based on input values.
    else
        disp('Bad classification');
        if w_transpose > 0
            pi = pi - X(row, 1:6);
        else
            pi = pi + X(row, 1:6);
        end
        runPi = 0;
    end
    iter = iter - 1;
end
format compact;
```

```
function result = calc_weight(weight, x_input, size);
result = 0;
for i=1:size;
        result = result + (weight(1,i) * x_input(1,i));
end
if result >= 0
    result = 1;
else
    result = -1;
end
```

```
% Runs the pocket algorithm for the xor funciton.

weightfunction my_xor();
% Initialize the values
weight = [0 0 0];
pi = [0 0 0];
runPi = 0;
runW = 0;

% xor function as an array (truth table)
% where d(x) is the desired output.
%    [bias  x1  x2  d(x)]
X = [1 -1 -1 -1
     1 -1  1  1
     1  1 -1  1
     1  1  1 -1];

iter = input('Choose how many iterations: ');
r_menu = input('Type "1" for Interactive, "2" for Computer only: ');

while (iter > 0)

    % Select rows interactively or randomly.
    if (r_menu == 1)
        row = input('Choose a row for input (1-4): ');
    else
        row = ceil(mod(randn(1)*1000, 4));
    end

    w_transpose = calc_weight(pi, X(row,1:3), 3);

    % Good classification, set our pocket values to the
    % current values.
    if ((w_transpose > 0) & (X(row,4) > 0)) | ((w_transpose < 0) & (X(row,4) < 0))
        disp('Good classification');
        runPi = runPi + 1;
        if (runPi > runW)
            weight = pi;
            runW = runPi;
        end
    % Bad classification, reset runPi to 0 and adjust pi
    % based on input values.
    else
        disp('Bad classification');
        if w_transpose > 0
            pi = pi - X(row, 1:3);
        else
            pi = pi + X(row, 1:3);
        end
        runPi = 0;
    end
    iter = iter - 1;
end
format compact;
weight
```

Appendix II

Assessment and Evaluation Form for the Pocket Algorithm

Purpose:

One of the purposes of this machine learning module is for the students to understand the basics of neural networks, such as definition, neuronal models, popular neural network structures, and their utility. Another purpose is to understand the specifics of the single layer perceptron architecture, the perceptron learning algorithm and the pocket algorithm. Furthermore, they should be able to learn (if they do not know already) how to read vectors/matrices, compute inner products, use “for loops” and “if then else” statements in a programming language of their choice. Finally, they are supposed to run some experiments with the perceptron and pocket learning algorithms and compare their performance on appropriate data sets.

Learner Outcomes:

1. Understand the basics of neural networks.
2. Understand the perceptron learning algorithm.
3. Understand the pocket algorithm.
4. Understand that the pocket algorithm is a type of gradient descent algorithm.
5. Appreciate the differences of the perceptron learning and the pocket algorithm.

Activity:

a. How Hard did you find the Homework ?
Very Hard [] Hard [] Normal [] Easy [] Very Easy []
b. Did you enjoy working on this homework?
A Lot [] Yes [] A bit [] No [] Not at all[]
c. Was the assignment useful to exercise your programming abilities ?
Very Useful [] Useful [] A bit Useful [] Not Useful [] No, it was Detrimental []
d. On a scale of 1 to 5, 1 meaning not at all and 5 meaning a lot, how much to you think that the homework was related to machine learning?
5 [] 4 [] 3[] 2[] 1[]
e. Would you be interested in knowing more about Machine Learning?
Very Interested [] Interested [] Just Curious [] Not Interested []

Evaluation, related to Learner Outcomes:

1. Did you understand the basics of neural networks? If you found it difficult to understand the basics what was the reason? Please explain.
2. Did you understand the single layer perceptron (SLP) neural network ? If you found it difficult to understand the SLP-NN what was the reason? Please explain.
3. Did you understand the perceptron learning algorithm? If you found it difficult to understand the perceptron learning algorithm what was the reason? Please explain.
4. Did you understand the pocket algorithm and its difference from the perceptron learning algorithm? If you found it difficult to comprehend the pocket algorithm what was the reason? Please explain.

5. Did you understand what gradient descent is and how it relates to the pocket algorithm? Please explain.
6. Did you acquire some appreciation of the perceptron learning and pocket algorithms and their applicability to real-world problems? Please explain.

Demographics:

Gender: Male [] Female []
Level: Sophomore [] Junior [] Senior []
Have you had any previous programming experience recently? Please explain.
What is the length of the longest program that you wrote? In what programming language?