1993

The Use of Neural Networks to Support "Intelligent" Scientific Computing

Anupam Joshi

Csanjiva Weerawarana

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

Report Number:
93-089

THE USE OF NEURAL NETWORKS TO SUPPORT "INTELLIGENT" SCIENTIFIC COMPUTING

Anupam Joshi
Sanjiva Weerawarana
Elias N. Houstis

CSD-TR-93-089
December 1993

The Use of Neural Networks to Support "Intelligent" Scientific Computing

Anupam Joshi, Sanjiva Weerawarana and Elias N. Houstis[1]
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, USA.

Abstract—
In this paper we report on the use of back-propagation based neural networks to implement a phase of the computational intelligence process of the PYTHIA[3] expert system for supporting the numerical simulation of applications modelled by partial differential equations (PDEs). PYTHIA is an exemplar based reasoning system that provides advice on what method and parameters to use for the simulation of a specified PDE based application. When advice is requested, the characteristics of the given model are matched with the characteristics of previously seen classes of models. The performance of various solution methods on previously seen similar classes of models is then used as a basis for predicting what method to use. Thus, a major step of the reasoning process in PYTHIA involves the analysis and categorization of models into classes of models based on their characteristics. In this study we demonstrate the use of neural networks to identify the class of predefined models whose characteristics *match* the ones of the specified PDE based application.

I. Introduction

Partial differential equations are important mathematical models for describing various physical phenomena. However, PDEs cannot generally be solved analytically. Instead, a very large number of approximate methods has been proposed to solve them numerically. In all cases, the numerical solution depends on the mathematical properties of the PDE model and its solution. Thus, different methods exist for each class of PDE model. For some simplified models, special PDE solvers have been developed that produce the numerical solution faster than the more general ones. In addition, all PDE solvers depend on several input parameters that the user has to specify. Very often, the selection of these parameters requires expert knowledge about the mathematical behavior of the methods. Due to the importance of computational models in engineering and science, there exists a large base of mathematical software that implements a number of them. The simulation of very large scale applications requires enormous computational and memory resources. In these cases the corresponding computational models have been implemented in high performance machines. These implementations usually involve an additional number of parameters associated with characteristics of the high performance computing environment and the methodologies used to explore the specialized computational resources. One can safely conclude that it is unrealistic to expect the average application scientist to master the above computing technology together with the knowledge of a particular application area. In order to assist the "novel" practitioner of scientific computing in the area of PDE based computational models, we have proposed [2] to utilize expert system technology to support a "smart" computational environment called //ELLPACK[2][1].

The computational intelligence of this environment is realized by the PYTHIA expert system. PYTHIA is an exemplar based reasoning system that attempts to answer the question of which method to use to solve a given PDE problem. PYTHIA considers the problem of selecting an "optimal" numerical method, its parameters, and components from the domain of //ELLPACK solvers for solving the types of problems described above such that it utilizes minimum hardware resources and satisfies user specified computational objectives. In this study, we use a version of PYTHIA restricted to the case of boundary value problems consisting of one second order linear elliptic partial differential equation with mixed boundary conditions. Since the strategies described in this paper are independent of any specific PDE problem properties, this restriction is not a limitation.

An important step in the process of identifying

[1]This work was supported in part by NSF grant 9202536-CCR, NSF grant 9123502-CDA and AFOSR grant F49620-92-J-0069.

[2]"//ELLPACK" is read "parallel ELLPACK."

the optimal method to solve a given problem is the categorization of the problem in terms of its membership in previously defined classes of problems. As will be explained later, this categorization leads to a comparative ordering of solution methods that work well for those classes and hence to the final recommendation of a solution method. This paper reports on the use of backpropagation neural networks at this step.

The rest of this paper is organized as follows. In Section II, we first describe the general framework of PYTHIA and its reasoning process. Then, in Section III we explain the use of neural networks in identifying the classes of PDE problems to which a given PDE problem belongs. Throughout, we refer to it as the "matching" problem. In Section IV we present some simulation results that indicate the effectiveness of neural networks in resolving the matching problem. Finally, in Section V we summarize our observations and conclude that backpropagation neural networks are indeed extremely effective for implementing the matching phase of the PYTHIA reasoning system.

II. PYTHIA

PYTHIA attempts to solve the problem of determining an optimal strategy (i.e., a solution method and its parameters) for solving a given PDE problem within user specified resource (i.e., limits on execution time and memory usage) and accuracy requirements (i.e., level of accuracy). Throughout this paper we will use the term "PDE problem" and "problem" equivalently.

PYTHIA uses the performance behavior of solution methods on previously solved problems as a basis for predicting a solution strategy for solving a given problem. The basic premise in PYTHIA's reasoning strategy is that performance data gathered during the solution of a set of problems using different solution methods can be used to predict the performance of these methods on a new problem. Of course, for this strategy to be correct, the new problem must be "similar" (i.e., have similar characteristics) to the problems that have been solved before.

Thus, to recommend a strategy to solve a given PDE problem (p), PYTHIA needs:

- a database of previous solved problems along with data on the effectiveness of various solution methods on those problems,
- comparative data on the effectiveness of various methods on the problems in the database, and
- a mechanism to identify the problems from the database that are similar to p.

With this information, PYTHIA uses the following algorithm to predict what method should be used to solve a given problem:

1. Analyze the PDE problem and identify its characteristics. This stage involves applying symbolic analysis to extract some characteristics and asking the user about characteristics that cannot be determined automatically.
2. Identify the set $S \subset P$, where S is the subset of problems whose characteristics are similar to the ones of the given problem and P (the "population of PDE problems") is the database of problems that have been solved before.
3. Analyze the performance data for the problems in S and rank the applicable methods to select the "best" method for solving the problem within the given computational and performance objectives.
4. Use the performance data available for this method to predict the values for appropriate parameters to achieve the specified computational and performance objectives.

The above (method, parameters) selection procedure could be executed at every user session. However, to generate accurate predictions, one must have a reasonably large population of problems whose characteristics encompass a significant portion of the allowable spectrum of characteristics. In this case, the execution time of the above process can be prohibitive as one would have to search a large space of problems in order to find the set S.

What PYTHIA does to avoid this problem is group the population of problems into smaller collections or classes. These classes, rather than the problems themselves, are then used to identify similar problems in the algorithm described above. In addition, PYTHIA derives a priori information about these classes that describe the effectiveness of various methods on the members of that class.

The modified reasoning strategy is, then:

1. Analyze the PDE problem and extract its characteristics.
2. Find problem classes to which the problem belongs and identify the "closer" classes.
3. Find close problems within those classes to use as representatives of the class.
4. Using performance rules for classes and the user specified performance preferences, find the best methods for solving the user's problem.
5. Using problem performance profiles for the representative problems, predict the performance of the selected methods on the user's problem.

Thus, PYTHIA consists of several components: a population of PDE problems, a collection of inter-

Operator	Boundary Conditions	I/O Functions	Solution
Poisson	Dirichlet	Smooth	Singular
Laplace	Neumann	Oscillatory	Analytic
Helmholtz	Mixed	Wave Front	Oscillatory
Self-adjoint	Homogeneous	Singular	
Homogeneous		Peak	

Table 1: Examples of PDE problem characteristics for elliptic partial differential equations.

esting problem classes, a database of performance profiles for this population, a knowledge base of performance rules for interesting classes of problems, and of course the inferencing environment that navigates these components.

A. PDE Problem Characteristics and Their Extraction

In this section we identify some of the characteristics of a PDE problem that are important in the context of PYTHIA. Our interest in identifying these characteristics is to detect correlations between the presence or absence of a certain characteristic and the suitability or effectiveness of a certain solution method.

Throughout we assume that the PDE problem is defined in terms of the following components: The PDE operator, the initial and boundary conditions, and the spatial and time domains of definition.

There are two main types of characteristics of a PDE problem: Characteristics of the problem components and characteristics of the solution. The set of characteristics of PDE problem components includes some classification information for the components (for example, whether the operator is homogeneous or not) and some quantitative information about the behavior (i.e., smoothness and local variation) of the I/O functions (i.e., coefficients of the operators, right hand side of the operators, boundary and initial conditions, and of course the solution) of the components. The characteristics of the solution are mainly smoothness classifications, for example whether the solution is analytic or whether it has singularities.

Table 1 shows some of the characteristics that we use to characterize a PDE problem and its solution. In the case of second order, linear, elliptic boundary value problems on rectangular domains, a discussion of the codification of problem characteristics is presented in [4].

Each characteristic is also associated with a presence level α, where $\alpha \epsilon [0, 1]$. ($\alpha = 0$ means pure absence of that property while $\alpha = 1$ means pure presence.) For logical characteristics (for example, whether the boundary conditions are Dirichlet or not), we use the values 0 and 1 for false and true, respectively.

The set of characteristics of a PDE problem are represented as a *characteristic vector* v. PYTHIA uses this characteristic vector to identify a similar PDE problem or a class of PDE problems from an *a priori* defined population of PDE problems.

B. Population of Elliptic PDEs

We described earlier how the first step of PYTHIA's reasoning approach is the identification of an *a priori* defined PDE problem or class of problems whose characteristic vector is "close" to that of the problem specified by the user. The success of this approach relies greatly then on having available a reasonably large population of various PDE problems whose characteristics span most of the space the space of all characteristic vectors. For the class of linear second order elliptic PDEs, PYTHIA uses the population defined in [4].

This population was created for use in the evaluation of numerical methods and software for solving PDEs. It consists of fifty–six linear, two–dimensional elliptic PDEs defined on rectangular domains. Forty–two of the problems are parametrized which leads to an actual problem space of more than two–hundred and fifty problems. Many of the PDEs were artificially created so as to exhibit various mathematical behaviors of interest; the others are taken from "real world" problems in various ways. The population has been structured by introducing measures of complexity of the operator, boundary conditions, solution and problem.

The population also contains information about the properties of the problems. This information is encoded as a bit–string and is accessible via the ELLPACK Performance Evaluation System (PES) [6]. These properties are transferred to the PYTHIA characteristic vector representation by an automated procedure. Characteristics not identified in the ELLPACK PES are identified by the characteristic extraction component of PYTHIA.

C. Classes of PDE Problems

A class of problems is a collection of problems which have similar characteristics. There are basically two approaches for identifying various classes: problem characteristics and performance of various methods on problems. Defining classes based on problem characteristics uses mathematical properties of the PDE problems to define the classes. In the second approach, classes are defined based on the effectiveness of various solution methods on the problems. That is, if a set of problems have "similar" performance behavior with respect to a certain solution method, then we consider those problems as belong-

ing to one class.

Once a classification of problems is available, given a new problem, the first and most important step in PYTHIA's reasoning process is the identification of the set of classes to which this problem belongs. Since many of the problem characteristics are rather subjective and vague, there cannot be an algorithmic process that specifies exactly the class memberships of a given problem. Only a heuristic scheme that identifies potential memberships is possible.

PYTHIA uses a simple heuristic based on problem characteristics to perform this identification. The characteristic vector for a problem class is defined as the average of the characteristic vectors of all the class members with the average computed element–by–element. That is, if the characteristic vector of a problem or class is $CV(\cdot)$, then the i–th element of the characteristic vector of class is defined as:

$$(CV(C))_i = \frac{1}{|C|} \sum_{P \in C} (CV(P))_i.$$

The distance from a problem P to a class C is defined as the norm of the difference of the two characteristic vectors:

$$d(P, C) = \|CV(P) - CV(C)\|.$$

Then, we say that a problem P belongs to a class C if $d(P, C) < \epsilon$ where ϵ is some threshold value that can be adjusted depending on the reliability of the characteristic vectors.

The next section will describe how we use backpropagation neural networks to identify the class(es) to which a given problem belongs.

III. The Use of Neural Networks

It should be clear from the description in the foregoing sections that there are many places where something has to be classified. The original problem, for instance, says that given a problem, and the time and error bounds that the user wants, PYTHIA should advice him/her on which method to use. Also, in the scheme that we have outlined, we want to find the class to which a new problem belongs. These are all instances of the classification problem.

We are given sample vectors $v \in \mathcal{R}^n$. We have to use these to divide $\mathcal{R}^n$ into m class zones C_i, $i = 1$ to m so that given another vector $u \in \mathcal{R}^n$, we can decide which class it belongs to. The situation is complicated by the fact that a given problem can belong to more than one class. In other words, two (or more) classes may overlap. Traditional cluster analysis techniques very often assume that clusters are disjoint or hierarchical. The same assumption is made by some neural network based methods like LVQ[5] as well.

An alternate view of the problem is to consider it as a mapping problem. Let us suppose that we represent the m classes by a vector of size m. We treat each of its elements as a binary decision variable, with a 1 in the i^{th} position of the vector indicating membership in the i^{th} class. Our problem now becomes one of mapping the characteristic vector of size n into the output vector which shows class memberships.

To perform this mapping, we use a backpropagation based Neural Network. Backpropagation has been successfully used to solve similar pattern matching problems. In this case, our input consists of the characteristic vector, which had 32 elements. The output consists of a vector with 5 elements, corresponding to the number of classes that we used in our simulations and therefore to the number of classes that we wish to categorize the data into. Unlike the original non–neural heuristic, we are not imposing an arbitrary structure on classes by insisting that their central value lies at the mean of the samples that we have available. Rather we are letting the network discover the structure of the classes. More importantly, backpropagation based networks exhibit the capability to generalize, namely to give correct outputs on hitherto unseen data. This is of singular importance here, since it potentially enables the network to correctly classify novel problems.

Since the input and output of the network are fixed by the problem, the only layer whose size had to be determined is the hidden layer. We arbitrarily chose this to have 10 elements. Also, since we had no *a priori* information on how the various input characteristics affect the classification, we chose not to impose any structure on the connection patterns in the network. Our network was thus *fully connected*, that is, each element in layer i was connected to each element in layer $i + 1$. Considering the small size of our network, this meant the the number of connections was still small. To be precise, there were only $32 \times 10 + 10 \times 5 = 370$ connections in the network.

IV. Simulation Results

To study the effectiveness of our approach, we used the population of PDE problems described earlier to define several classes of problems based on problem characteristics. We defined the following non–exclusive classes (the number in parenthesis indicates the number of problems that belong to that class):

1. Solution–Singular: Problems whose solution has at least one singularity (6).

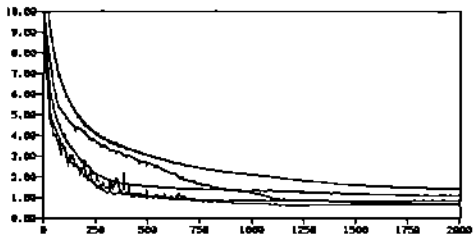

Figure 1: Plot of Sum of Squared Error vs. Iteration Number.

2. SOLUTION–ANALYTIC: Problems whose solution is analytic (35).
3. SOLUTION–OSCILLATORY: Problems whose solution oscillates (34).
4. SOLUTION–BOUNDARY–LAYER: Problems whose solutions depict a boundary layer (32).
5. BOUNDARY–CONDITIONS–MIXED: Problems with mixed boundary conditions (74).

There were a total of 167 problems in the population that belonged to at least one of these classes. We split this data into two parts– one part contained two thirds of the exemplars and was used to train the network. The other part was used to test the performance of the trained network.

All the simulations were performed using the Stuttgart Neural Network Simulator[7], a very useful public domain simulator with a convenient graphical interface.

The first training algorithm used was a "vanilla" backpropagation routine. The *learning rate* is an important free parameter here. It is the value by which the gradient term is multiplied before being used to update the weights of the network. In Figure 1, we show the Sum of Squared Error (SSE) for the network as it changes with the training epochs. As expected, the best performance is obtained by some reasonably small value for this parameter, in this case 0.2. While smaller values also converge to the same SSE value, they take much longer. Larger values show oscillatory behavior as they converge, and may get trapped in local minimum, as can be seen by their settling to a larger SSE in the graph.

A popular variation on the standard backpropagation is to use backpropagation with momentum and flat spot elimination. This involves using another term to modify the weights, which is the second derivative of the error function. Also, a small constant term is added to the derivative to avoid local minima. We next used this learning scheme. The value of the learning rate was fixed at 0.2 from the previous experiments, and the flat spot elimination constant was chosen as 0.05. The parameter multiplying the momentum term was varied and once

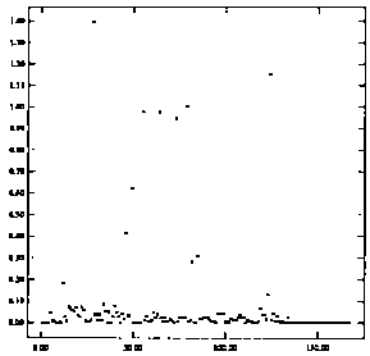

Figure 2: Scatter Plot of Error vs. Vector Number – Larger training set.

again plots of SSE vs. iteration number were made. The best performance seemed to be for the momentum term at about 0.8. It was also clear that the addition of the momentum and flat spot elimination lead as expected to lower SSE values than the standard backpropagation, and much faster convergence.

Clearly the network was learning to correctly classify all the problems it had been trained on. However, one of the reasons we wished to use neural networks was that they could generalize. The next step therefore is to verify that our network was generalizing correctly. To do this, we split the total of 167 vectors that we had into two sets, one set consisting of 111 vectors, and the other consisting of 56 vectors. First, the network was trained for 2000 epochs with the larger set. The learning rate was 0.2, the momentum 0.8 and the flat spot elimination constant was 0.05. After training, the network was presented with all the 167 vectors, and its output was recorded. To interpret and analyze the output, we computed the least square norm of the expected and actual outputs for each of the 167 cases. In Figure 2, we show a scatter plot for the results, with the X axis showing the vector number (from 1 through 167), and the Y axis showing the L_2 error norm. As can be clearly seen, the network correctly classified all but a few of the vectors, which show up as outliers. This shows the generalization powers of such networks, namely their ability to correctly classify hitherto unknown inputs.

To further demonstrate the generalization power of such networks, we next trained it with the smaller set. The parameters and the number of epochs remained the same as in the previous case. The testing was once again done with the complete set of vectors. Again, we noticed that the number of outliers are few. However, as expected, training with

Training Set	Threshold Value			
	0.2	0.1	0.05	0.005
Large	157	155	143	67
Small	142	132	113	37

Table 2: Number of correctly classified vectors with various training sets.

Training Set	Mean	Median
Large	0.0652	0.0095
Small	0.1367	0.0235

Table 3: Statistics for the error norms with various training sets.

fewer samples did lead to a degradation of performance with respect to the previous case.

Tables 2 and 3 present some further analysis of this data. We first chose an arbitrary threshold for the L2 error norm. Error norms above the threshold would imply that the corresponding input had been misclassified. In table 2, we show the number of correctly classified vectors (out of 167) for different values of the threshold. We can clearly see that the network is extremely successful in classifying correctly, especially with the larger training set. The same trend is reflected in Table 3, where we present the mean and median values for the error norms in the two cases. Notice that while the mean value of error for the smaller training set is slightly higher, the median value remains small. This clearly illustrates that while there are outliers, most of the problem do get classified correctly.

V. Discussion

We present here a simple backpropagation based architecture to address the issue of classifying a PDE problem presented to PYTHIA. This is required in order for PYTHIA to advise the user. Due to the inherent generalization capabilities of the neural net, it can correctly classify novel problems as well. Further work on the neural aspects of PYTHIA is in progress in several dimensions. The decisions involved in the process of selecting the best method given the problem and solution criterion are not crisp. We feel that in light of this, we can improve upon the performance of the current selection method. We are developing a method to directly map the original problem, that of selecting a method for a problem provided the user's estimates of the required time/grid and error criterion, to a Neural Network. While this leads to an increased learning time, the decision process would be virtually instantaneous, especially if we exploit the SIMD parallelism inherent in the network. However, it is not immediately clear if feedforward nets should be used for this problem, or whether other topologies and learning strategies might prove more efficient. We have conducted several experiments in this connection with encouraging results, but a paucity of space prevents us from reproducing them here. Also, we are investigating extensions to predict when one would need to use a parallel machine and when necessary, what machine to use and what its configuration should be.

References

[1] E. N. Houstis, J. R. Rice, N. P. Chrisochoides, H. C. Karathanasis, P. N. Papachiou, M. K. Samartzis, E. A. Vavalis, Ko-Yang Wang, and S. Weerawarana. //ELLPACK: A numerical simulation programming environment for parallel MIMD machines. In J. Sopka, editor, *Proceedings of Supercomputing '90*, pages 96–107. ACM Press, 1990.

[2] E. N. Houstis, J. R. Rice, C. E. Houstis, T. S. Papatheodorou, and P. Varodoglu. ATHENA: A knowledge based system for parallel ELLPACK. In E. Diday and Y. Lechevallier, editors, *Symbolic - Numeric Data Analysis and Learning*, pages 459–467. Nova Science, 1991.

[3] E. N. Houstis, J. R. Rice, S. Weerawarana, and C. E. Houstis. PYTHIA: An expert system for the optimal selection of PDE solvers based on the exemplar learning approach. to appear.

[4] Elias N. Houstis John R. Rice and Wayne R. Dyksen. A population of linear, second order, elliptic partial differential equations on rectangular domains, part I. *Mathematics of Computation*, 36(154):475–484, 1981.

[5] T. Kohonen, J. Kangas, J. Laaksonen, and K. Torkkola. LVQ_PAK: A program package for the correct application of Learning Vector Quantization algorithms. In *Proceedings of the International Joint Confrence on Neural Networks*, volume 1, pages 725–730. IEEE, 1992.

[6] John R. Rice Ronald F. Boisvert and Elias N. Houstis. A system for performance evaluation of partial differential equations software. *IEEE Transactions on Software Engineering*, SE-5(4):418–425, 1979.

[7] A. Zell, N. Mache, R. Hübner, G. Mamier, M. Vogt, K-U Herrmann, M. Schmalzl, T. Sommer, A. Hatzigeorgiou, S. Döring, and D. Posselt. Stuttgart Neural Network Simulator User Manual, Version 3.1. Technical Report 3, University of Stuttgart, 1993.