A Neural Net-Based Approach to Software Metrics

G. Boetticher, K. Srinivas, D. Eichmann
Software Reuse Repository Lab
Department of Statistics and Computer Science
West Virginia University

Abstract

Software metrics provide effective methods for characterizing software. Metrics have traditionally been composed through the definition of an equation, but this approach is limited by the fact that all the interrelationships among all the parameters be fully understood. Derivation of a polynomial providing the desired characteristics is a substantial challenge. This paper explores an alternative, neural network approach to generating metrics. Experiments performed on two widely known metrics, McCabe and Halstead, indicate that the approach is sound, thus serving as the groundwork for further exploration into the analysis and design of software metrics.

1. Introduction

As software engineering matures into a true engineering discipline, there is an increasing need for a corresponding maturity in repeatability, assessment, and measurement — both of the processes and of the artifacts associated with software. Repeatability of process is inherent to a true engineering discipline. Repeatability of artifact takes natural form in the notion of software reuse, whether of code or of some other artifact resulting from a development or maintenance process.

Accurate assessment of a component's quality and reusability are critical to a successful reuse effort. Components must be easily comprehensible, easily incorporated into new systems, and behave as anticipated in those new systems. Unfortunately, no consensus currently exists on how to go about measuring a component's reusability. One reason for this is a less than complete understanding of software reuse, yet obviously it is useful to measure something that is not completely understood. The number of potential parameters involved in a reusability metric implies that the derivation of such a metric will be a significant challenge.

This paper describes a preliminary set of experiments to determine whether neural networks can model known software metrics. If they can, then neural networks can also serve as a tool to create new metrics. Establishing a set of measures raises questions of coverage (whether the metric covers all features), weightings of the measures, accuracy of the measures, and applicability over various application domains. The appeal of a neural approach lies in a neural network's ability to model a function without the need to have knowledge of that function, thereby providing an opportunity to provide an assessment in some form, even if it is as simple as *this* component is reusable, and *that* component is not.

If all we seek is an assessment of a component, a properly trained neural network produces results comparable to a traditional algorithmic implementation of a multi-variable polynomial. In fact, while much of the research in metrics is concerned with the derivation of the polynomial, the result of evaluating that polynomial is frequently the true goal of the research. Our intent with this work was therefore not to show that we could model existing metrics just to avoid evaluating a known polynomial. Our intent was rather to show that our approach is capable of modeling both the linearity of the McCabe metric and the more complex Halstead measure with a reasonable amount of accuracy, and hence applicable to metrics of unknown simplicity or complexity, such as a reusability metric. Validating our approach against known benchmarks increases our assurance that the technique is sound and provides valuable feedback concerning the sensitivity to such factors as the architecture of the neural network.

We begin in section 2 by describing the two software metrics used, McCabe and Halstead, and then in section 3 briefly discuss various neural network architectures and their applicability. Section 4 presents our approach and scetion 5 the actual experiment. We draw conclusions in section 6, and present prospects for future work in section 7.

2. Software metrics

There are currently many different metrics for assessing software. Metrics may focus on lines of code, complexity [10, 11], volume[6], or cohesion [2, 3] to name a few. Among the many metrics (and their variants) that exist, the McCabe and Halstead metrics are probably the most widely known, frequently appearing in introductory material on the subject. More current and complete coverage of this area appear in work such as [12].

Traditionally, software metrics are generated by extracting values from a program and substituting them into an equation. In certain instances, equations may be merged together using some weighted average scheme. This approach works well for simple metrics, but as our models become more sophisticated, deriving metrics with equations becomes harder. The traditional process requires the developer to completely understand the relationship among all the variables in the proposed metric. This demand on a designer's understanding of a problem limits metric sophistication (i.e., complexity). One reason why it is so hard to develop reuse metrics, for example, is that no one completely understands "design for reuse" issues.

The goal then is to find alternative methods for generating software metrics. Deriving a metric using a neural network has several advantages. The developer need only to determine the endpoints (inputs and output) and can disregard (to an extent) the path taken. Unlike the traditional approach, where the developer is saddled with the burden of relating terms, a neural network automatically creates relationships among metric terms. Traditionalists might argue that you must fully understand the nuances among terms, but full understanding frequently takes a long time, particularly when there are numerous variables involved.

3. Neural Networks

Neural networks by their very nature support modeling. In particular, there are many applications of neural network algorithms in solving classification problems, even where the classification boundaries are not clearly defined and where multiple boundaries exist and we desire the best among the alternatives. It seems only natural then to use a neural network in classifying software.

There were two principle criteria determining which neural network to use for this experiment. First, we needed a supervised neural network, since for this experiment the answers are known. Second, the network needed to be able to classify.

The back-propagation algorithm [9] meets both of these criteria. It works by calculating a partial first derivative of the overall error with respect to each weight, taking small steps down a gradient [4]. However, a major problem with the back-propagation algorithm is that it is exceedingly slow to converge [7]. Fahlman developed the quickprop algorithm as a way of using the higher-order derivatives in order to take advantage of the curvature [4]. The quickprop algorithm uses second order derivatives in a fashion similar to Newton's method. From previous experiments we found the quickprop algorithm to clearly outperform a standard back-propagation neural network.

While an argument could be made for employing other types of neural models, due to the linear nature of several metrics, we chose quickprop to ensure stability and continuity in our experiments when we moved to more complex domains in future work.

4. Modeling Metrics with Neural Networks

As mentioned earlier, the goal of our research is to determine whether a neural network could be used as a tool to generate a software metric. In order to determine whether this is possible, the first step is determining whether a neural network can model existing metrics, in this case McCabe and Halstead. These two were chosen not from a belief that they are particularly good measures, but rather because they are widely known, public domain programs exist to generate the metric values, and the fact that the McCabe and Halstead metrics are representative of major metric domains (complexity and volume, respectively).

Since our long term goal is to determine whether a neural network can be used to derive software reusability metrics, Ada, with its support for reuse (generics, unconstrained arrays, etc.) seemed a reasonable choice for our domain language. Furthermore, the ample supply of public domain Ada software available from repositories (e.g., [1]) provides a rich testbed from which to draw programs for analysis.

Finally, programs from several distinct application domains (e.g., abstract data types, program editors, numeric utilities and system oriented programs) were included in the test suite to ensure variety.

We ran three distinct experiments. The first experiment modeled the McCabe metric on single procedures, effectively fixing the unit variable at 1. The second experiment extended the first to the full McCabe metric, including the unit count in the input vector, and using complete packages as test data. The third experiment used the same test data in modeling the Halstead metric, but a different set of training vectors. We present here only the results of the third experiment.

5. A Neural Halstead Metric for Packages

Based upon the results of the first two experiments, we assumed for this experiment that if the experiment worked for packages, then it also worked for procedures, and further, that the increasing the number of training set vectors improves upon the results. Therefore, the focus of this experiment was on varying neural network architectures over a fixed-size training set.

The experiment ranged over seven different neural network architectures broken into three groups: broad, shallow architectures (4-5-3, 4-7-3, and 4-10-3), narrow, deep architectures (4-7-7-3 and 4-7-7-7-3), and narrow, deep architectures with hidden layers that connected to all previous layers (4+7-7-3 and 4+7+7-7-3). We formed these

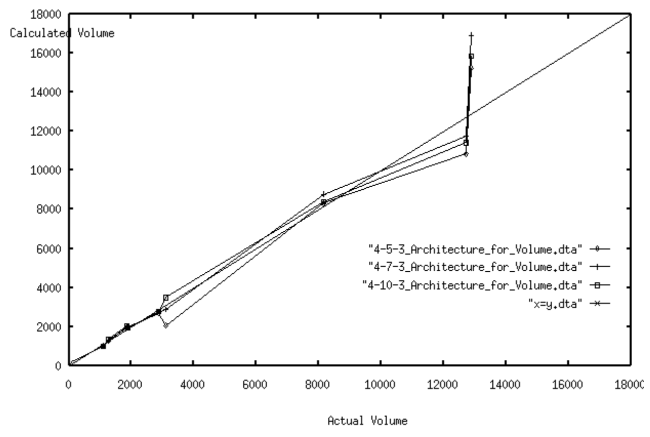

Figure 1. Volume Results, Broad Architectures

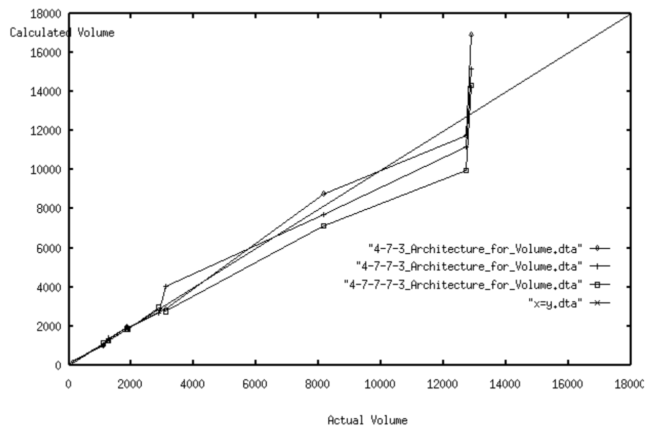

Figure 2. Volume Results, Deep Architectures

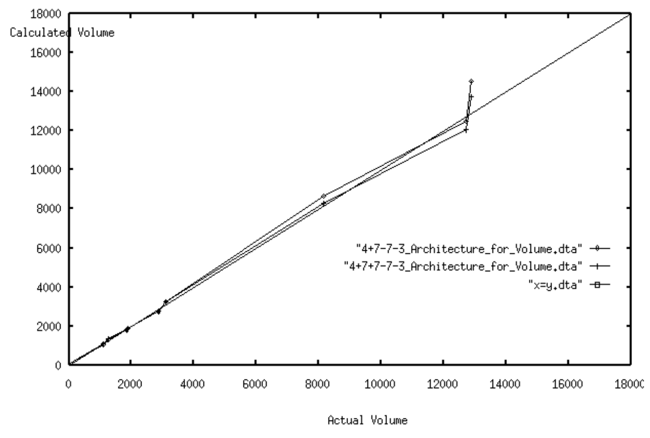

Figure 3. Volume Results, Connected Architectures

three groups in order to discover whether there was any connection between the complexity of an architecture and its ability to model a metric.

Figures 1, 2, and 3 present the results for the Halstead volume for broad, deep, and connected architectures,

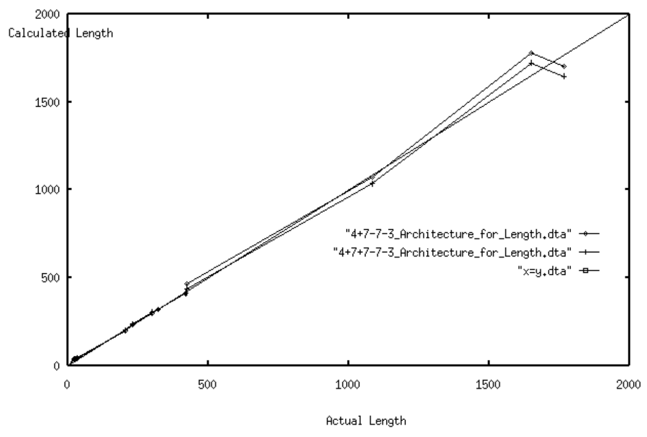

Figure 4. Length Results, Connected Architectures

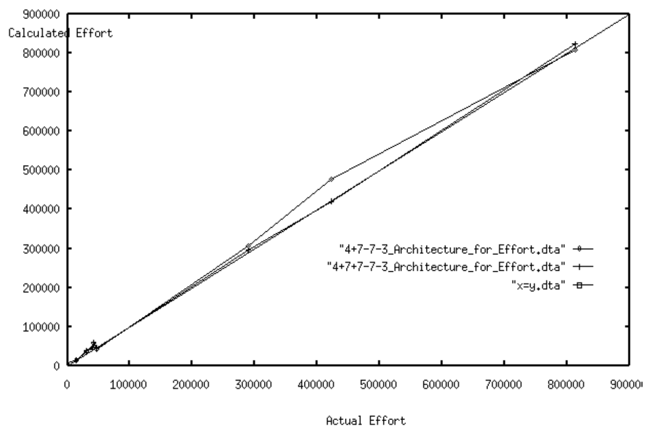

Figure 5. Effort Results, Connected Architectures

respectively. Note that both the broad and deep architectures do moderately well at matching the actual Halstead volume metric, but the connected architecture performs significantly better. Furthermore, there is no significant advantage for a five versus four layer connected architecture, indicating that connecting multiple layers may be a sufficient condition for adequately modeling the metric.

This pattern of performance also held for the Halstead length metric and the Halstead effort metric, so we show only the results for the connected architecture in Figure 4 and Figure 5, respectively.

6. Conclusions

The experimental results clearly indicate that a neural network approach for modeling metrics is feasible. In all experiments the results corresponded well with the actual values calculated by traditional methods. Both the data set and the neural network architecture reached performance saturation points in the McCabe metric. In the Halstead experiment, the fact that the results oscillated over the actual-calculated line indicate that the neural network was attempting to model the desired values. Adding more training vectors, especially ones containing larger values, would smooth out the oscillation.

7. Future work

Applying this work to other existing metrics is an obvious extension, but we feel that the development of new metrics by applying neural approaches is much more significant. In particular, expanding this work to the development of a reusability metric offers great promise. Effective reuse is only possible with effective assessment and classification. Since no easy algorithmic solutions currently exist, we've turned to neural networks to support the derivation of reusability metrics. Unsupervised learning provides interesting possibilities for this domain, letting the algorithm create its own clusters and avoiding the need for significant human intervention.

Coverage and accuracy are important aspects of developing a neural network to derive a software reuse metric. McCabe and Halstead metrics are interesting and useful, but they do not provide coverage regarding reusability. We need to expand the number of parameters in the data set in order to provide adequate coverage with respect to reusability of a component. We also would like to improve the accuracy of answers by enlarging our data sets to include possibly hundreds of training set vectors. This will need to be a requirement when exploring more complex metric scenarios, and the cost of such extended training is easily borne over the expected usage of the metric.

The possibility of extracting the function from a trained neural network is still an open research issue for neural researchers. The prospect is a very interesting one for us, however, since it would allow the use of a neural network in the generation and evolution of a metric, while still allowing the metric user and creator to evaluate the nature of the relationship of the metric inputs, as well as providing more efficient evaluation in a production context.

Finally, it is possible to explore alternative neural network models. For example, the cascade correlation model [5] dynamically builds the neural network architecture, potentially automating much of the process described here.

8. Acknowledgments

This work was supported in part by NASA as part of the Repository Based Software Engineering project, cooperative agreement NCC-9-16, project no. RICIS SE.43, subcontract no. 089. David Eichmann's current address is Software Engineering Program, University of Houston – Clear Lake, Houston, TX.

9. References

[1] Conn, R., "The Ada Software Repository and Software Reusability," *Proc. of the Fifth Annual Joint Conference on Ada Technology and Washington Ada Symposium*, 1987, pp. 45-53.

[2] Emerson, T. J., "A Discriminant Metric for Module Cohesion," *Proc. 7th International Conference on Software Engineering*, Los Alamitos, California, IEEE Computer Society, 1984, pp. 294-303.

[3] Emerson, T. J., "Program Testing, Path Coverage, and the Cohesion Metric," *Proc. of the 8th Annual Computer Software and Applications Conference*, IEEE Computer Society, pp. 421-431.

[4] Fahlman, S. E., *An Empirical Study of Learning Speed in Back-Propagation Networks*, Tech Report CMU-CS-88-162, Carnegie Mellon University, September, 1988.

[5] Fahlman, S. E. and Lebiere, M., *The Cascade-Correlation Learning Architecture*, Tech Report CMU-CS-90-100, Carnegie Mellon University, August 1991.

[6] Halstead, M.H., *Elements of Software Science*, North-Holland (Elsevier Computer Science Library), New York, 1977.

[7] Hertz, J., Krogh A., Palmer, R. G., *Introduction to the Theory of Neural Computation*, Addison Wesley, New York, 1991.

[8] Li, H.F. and Cheung, W.K., "An Empirical Study of Software Metrics," *IEEE Transactions on Software Engineering*, vol. 13, no. 6, pp. 697-708, June 1987.

[9] Lippmann, R. P. "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, pp. 4-22, April 1987.

[10] McCabe, T.J., "A complexity measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308-320, Dec. 1976.

[11] McCabe, T.J., "Design Complexity Measurement and Testing," *Communications of the ACM*, vol. 32, no. 12, pp. 1415-1425, December 1989.

[12] Zuse, H., *Software Complexity: Measures and Methods*, Walter de Gruyter, Berlin, 1991.