Neural Network Architecture Selection Analysis With Application to Cryptography Location

Jason L. Wright, *Student Member, IEEE* and Milos Manic, *Senior Member, IEEE*

***Abstract*—When training a neural network it is tempting to experiment with architectures until a low total error is achieved. The danger in doing so is the creation of a network that loses generality by over-learning the training data; lower total error does not necessarily translate into a low total error in validation. The resulting network may keenly detect the samples used to train it, without being able to detect subtle variations in new data. In this paper, a method is presented for choosing the best neural network architecture for a given data set based on observation of its accuracy, precision, and mean square error. The method, based on [1], relies on k-fold cross validation to evaluate each network architecture k times to improve the reliability of the choice of the optimal architecture. The need for four separate divisions of the data set is demonstrated (testing, training, and validation, as normal, and an comparison set). Instead of measuring simply the total error the resulting discrete measures of accuracy, precision, false positive, and false negative are used. This method is then applied to the problem of locating cryptographic algorithms in compiled object code for two different CPU architectures to demonstrate the suitability of the method.**

I. Introduction

The choice of neural network architecture to solve a particular problem drastically affects the effectiveness of the resulting solution. In this paper a method of choosing the appropriate architecture for a particular problem is described by examination of the mean square error of trained networks.

The problem of neural network architecture selection is a central problem in the application of neural network computation. A single neuron is capable of learning a simple line in an n dimensional space which bifurcates the space. More complex networks can circumscribe more complex shapes, but the problem is deciding which network most accurately and generally describes the shape of the problem. The goal is to create a network complex enough to learn the data, but small enough to avoid over-learning (in this paper, over-learning, over-fitting and over-training are used interchangeably).

A network that is too simple will only capture the most gross features of a data set. In fact, neural network learning models have been created that will tend to discover the statistical principal components of a data set [2]. There are also techniques that can automatically create networks that perform arbitrarily well on data sets like cascade correlation [3], hybrid evolutionary neural network construction, network pruning [4], or function complexity analysis [5]. One classic implementation of network pruning is Optimal Brain Damage which attempts to remove connections between neurons that have low weight [6].

Each of the previous methods requires a good understanding of the underlying tuning parameters. For instance, in cascade correlation, the choice of the minimum error provides the ability to create a network that performs extremely well on the training data, but is not well generalized enough to recognize subtleties in the data.

The process for neural network architecture selection described in this paper is then applied to the problem of locating cryptography algorithms within compiled machine code. The application for such a tool is primarily in the malware analysis community where many malware samples are taking advantage of the availability of strong cryptography primitives (e.g. Conficker [7], [8] and Rustock [9] worms). Other applications include export compliance verification and software component verification.

In previous work, a simplistic neural network has been proposed for the the problem of locating cryptographic algorithms in compiled object code [10]. This methodology has been generalized with automatic feature selection to determine the most relevant measurements to obtain, and this work builds upon that by taking the reduced dimensions and using them to train various neural networks [11]. The problem addressed in this paper is that of determining which neural network architecture best fits the problem.

Artificial neural networks have the ability to learn numeric solutions to problems, but they have several pitfalls. In particular, a network that is too small, either in number of neurons or number of hidden layers, will tend to learn the most gross behavior of the training data and ignore subtleties. However, a network that is too large will tend to over-specialize and learn the training data too well, resulting in a network that has does not solve the general problem.

Various techniques have been proposed for automatic neural architecture determination. Some techniques are constructive: neurons and/or connections are added until a stopping criteria is met. One classic example is the cascade correlation architecture which dynamically "grows" a network by adding a hidden layer and a single neuron in that layer until a stopping

Jason L. Wright is with the Modern Heuristics Research Group at the University of Idaho in Idaho Falls and the Idaho National Laboratory: 2525 Freemont Ave., MS2604, Idaho Falls, ID, USA 83415-2604; (phone: +1 (208)526-9476; email: jlwright@ieee.org).

Milos Manic is with the Modern Heuristics Research Group at the University of Idaho in Idaho Falls: Idaho Falls, ID, USA 83402 (email: misko@ieee.org).

978-1-4244-8126-2/10/$26.00 ©2010 IEEE

criterion is met [3] which results in a network with many hidden layers with a single neuron each. Another constructive method for building networks constructs a single hidden layer and add neurons until a stopping criterion is met [1]. This method relies on k-fold cross correlation where the input data is divided into k complementary sets. Some number of these sets, usually $k-1$, are used for training the candidate neural network, and the remaining set is used for validation. The role of the validation set is to determine whether the network is over-learning the input data set. It is a variation of this method that is described in this paper. The primary difference is the use of the discrete measures of accuracy, precision, false positive and negative rates (defined below).

The mean square error (MSE) of a classifier is a measure of correctness of the network to its input set. The formal definition is in Equation 1 where n is the number of points in the data set, d_i is the desired output for input i, o_i is the actual output for input i.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(d_i - o_i)^2 \tag{1}$$

Accuracy and precision are related concepts that measure the performance of a classifier [12]. Accuracy is the number of correctly classified items divided by the total number of items, and precision is a measure of correctly classified positive cases. Given a confusion matrix like the example in Table I, accuracy (AC) and precision (P) are calculated using Equations 2 and 3, respectively. False positive (FP) and false negative (FN) are measures of incorrectly classified cases relative to the total for that type of case using Equations 4 and 5, respectively.

TABLE I
EXAMPLE CONFUSION MATRIX.

classified as:		
non-crypto	**crypto**	
(a)	*(b)*	**non-crypto**
(c)	*(d)*	**crypto**

$$AC = \frac{a+d}{a+b+c+d} \tag{2}$$

$$P = \frac{d}{b+d} \tag{3}$$

$$FP = \frac{b}{a+b} \tag{4}$$

$$FN = \frac{c}{c+d} \tag{5}$$

The rest of this paper is organized as follows: Section II describes the architecture selection analysis process, Section III depicts the results of applying the architecture selection technique on the cryptography location problem, and Section IV provides the conclusion and future directions for this work.

II. ARCHITECTURE SELECTION ANALYSIS PROCESS

The basic architecture selection process is described in Figure 1. Each network architecture is trained k times (one fold is used as a validation set during each iteration), and the accuracy, precision, and mean square error for each architecture is recorded. The mean of each fold's MSE is defined as in Equation 6 where k is the number of folds and MSE_i is the mean square error of fold i. For each architecture and fold, MMSE is computed for the whole data set ($MMSE_a$), the training set ($MMSE_t$), and the validation set ($MMSE_v$). The goal is to find a neural network architecture that reacts consistently across $MMSE_a$, $MMSE_v$, and $MMSE_t$.

NETWORK-SELECTION-PROCESS
1: gather data
2: divide data into k stratified folds
3: **for all** candidate architectures **do**
4: **for** $i = 1$ to k **do**
5: train network with $k-1$ folds
6: validate with the kth fold
7: $MSE_t \leftarrow$ MSE of training set
8: $MSE_v \leftarrow$ MSE of validation set
9: $AC_t \leftarrow$ accuracy of training set
10: $AC_v \leftarrow$ accuracy of validation set
11: $P_t \leftarrow$ precision of training set
12: $P_v \leftarrow$ precision of validation set
13: **end for**
14: calculate mean MSE_t, MSE_v
15: calculate mean AC_t, AC_v, and P
16: calculate mean P_t, P_v
17: **end for**
18: rule out networks with low mean AC_t, AC_v, P_t, or P_v
19: choose network with consistent mean MSE_t, MSE_v

Fig. 1. Architecture selection process.

Similarly, the mean accuracy and precision are computed for each of the $k = 10$ folds (Equations 7 and 8, respectively). Strong performance on a single measure is not necessarily indicative of a good classifier, but an examination of all of the measures give an indication of the suitability of a candidate classifier to the problem. The selection of the "best" architecture is based on observation of the resulting data. A classifier with extremely low MSE square error may not be the best choice as it could indicate over-learning of the input data. For example, a classifier that gives low overall MSE, but has a high MSE on the validation set, is unlikely to be a good choice.

$$MMSE = \frac{1}{k}\sum_{i=1}^{k} MSE_i \tag{6}$$

$$MAC = \frac{1}{k}\sum_{i=1}^{k} AC_i \tag{7}$$

$$MP = \frac{1}{k}\sum_{i=1}^{k} P_i \tag{8}$$

For this method, a multilayer perceptron network architecture is used with a hyperbolic tangent activation function. A random weight set is used initially and the network is trained with the combination of the Levenberg-Marquardt (LM) and Error Back Propagation (EBP) algorithms. This combination combines the speed of LM with the convergence behavior of EBP. Training is stopped when the gradient is less than 10^{-10} or the maximum number of training epochs (1000) is reached. The number of neurons and their distribution among hidden layers is varied.

The notation used to describe each architecture is $a/b/c/d$ where a is the number of neurons to which the inputs are connected (first hidden layer), b is the number of neurons in the second hidden layer, c is the number of neurons in the third hidden layer, and d is the number of neurons in the output layer. For example, Figure 2 shows an example 3/2/1 network with 4 inputs. In this problem, there are always 8 inputs and exactly one output neuron as there is only a single output (the classification of the function as being cryptography or not). The bias input to each neuron is not depicted.

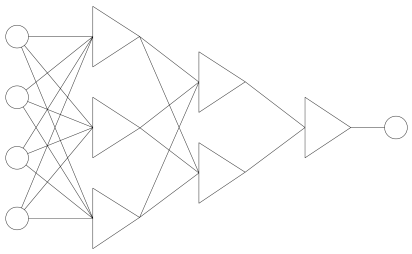

Fig. 2. Example 3/2/1 neural network architecture with 4 inputs and 1 output.

III. Test Results

In this section, the method of selecting a neural network architecture described in Section II is applied to the problem of locating cryptography in object code. To determine the correct architecture for the problem of locating cryptography in compiled object code, several neural networks were constructed and once trained, the resulting mean square error, accuracy and precision are examined to determine the best choice for the architecture that captures the behavior of the data set.

Section III-A describes how the data set is gathered. Section III-B shows the selection of a network for the Intel X86 CPU architecture, and Section III-C demonstrates the technique applied to the SPARC CPU architecture.

A. Gathering Data Set

The input data set for training the networks consists of the count of the number of each distinct processor opcode in each function in a library as well as the density of each opcode. An opcode here refers to an assembly language instruction mnemonic. The available opcodes differ for each instruction set architecture, e.g. X86 opcodes are largely distinct from SPARC opcodes, but there are some commonalities. In particular, simple mathematical operations such as addition, subtraction, multiplication, AND, OR, and eXclusive-OR are represented by familiar opcodes: *add*, *sub*, *and*, etc.

The number of occurrences of opcode i in function f is given by Equation 9. Density is defined as the number occurrences of a particular opcode in a function divided by the total number of opcodes in that function, Equation 10.

$$\sigma_{i,f} = \#\text{ of opcode } i \text{ in function } f \quad (9)$$

$$\rho_{i,f} = \frac{\sigma_{i,f}}{\text{total opcodes in function } f} \quad (10)$$

The library consists of the C library from the OpenBSD operating system, which contains a number of cryptographic algorithms (SKIPJACK, BLOWFISH, DES, 3DES, MD5, MD4, SHA1, SHA2, etc.) and a collection of other cryptographic algorithms: GOST, TEA, LUCIFER, RC4, RC5, and IDEA. This results in 53 cryptographic functions and 1723 non-cryptographic functions. The library is compiled with four different optimization levels making for a total more than 7104 data points (functions) with 8 dimensions (opcode counts and densities) measured for each. In addition to the total number and the density of each opcode in each function, an expertly determined cryptography indicator, $y \in \{-1, 1\}$, is added for each function.

The data set was then divided into $k = 10$ folds. The folds were stratified, meaning that ratio of cryptography to non-cryptography functions in each fold was kept constant. The reason for stratifying the folds was the small number of positive cases (cryptography) to negative cases (non-cryptography). Less than 4% of cases are cryptographic algorithms. Varying k to, for instance $k = 5$, might make the differences in architectures. This has the effect of making the testing and validation sets larger and should make them more consistent. Additional consistency in the testing and validation sets should, in turn, make the differences in architecture choice more apparent.

B. Neural Network Architecture Selection (x86)

For initial testing, the representative library is compiled for the X86 architecture which is also known as the Intel IA32 architecture [13]. This is an important architecture because it is found on almost every desktop computer and is the common target for malware.

The set of 7104 data points are divided into 10 stratified subsets. For each each tested architecture, nine of the sets are used for training and the remaining set is used as the validation set. Each architecture is trained 10 times leaving using a different validation set each time. Table II lists inputs used for each tested neural network architecture; ρx denotes the density of opcode x in each function, and σx denotes the total number of x in the function.

TABLE II
X86 OPCODES USED AS NETWORK INPUTS

ρje	ρshr	ρxor	ρmovz
σrol	σxor	σshl	ρor

Figure 3 shows the mean accuracy (MAC) of each tested architecture. Along the horizontal access are various architectures using the notation described in Section II. With a single hidden layer, accuracy improves as neurons are added until the number reaches 4; past this point, the accuracy declines. The overall peak of this graph corresponds to 4 neurons in a single hidden layer. No significant increase in accuracy occurs as second hidden layer neurons are added to the network. The 6/3/1 network is almost identical in accuracy to the 4/1 network. The standard deviation of the accuracy across folds is also included in the graph to demonstrate the consistency between tests. For both networks (4/1 and 6/3/1), the standard deviation is low meaning that across folds, the architectures produced similar accuracies.

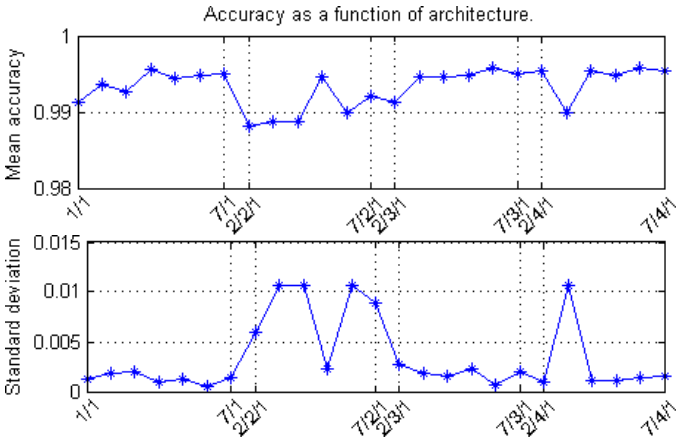

Fig. 3. Mean accuracy vs. network architecture.

In Figure 4, mean precision is plotted. There are peaks at 4 neurons (one hidden layer) and 6/3/1 (6 neurons in the first hidden layer, 3 in the second hidden layer). These two points correspond with peaks in accuracy as well and show low deviation across cross validation folds.

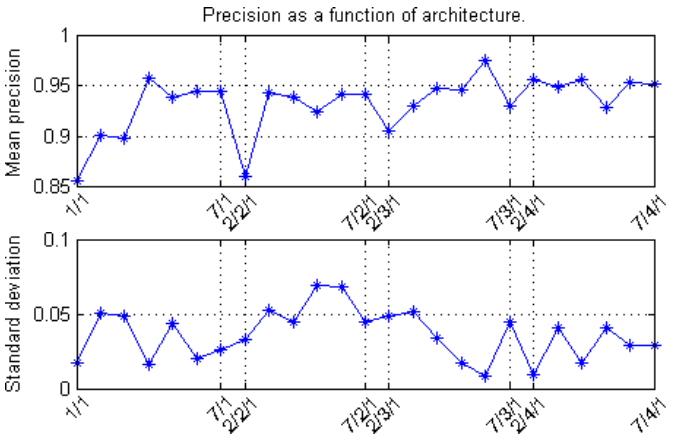

Fig. 4. Mean precision vs. network architecture.

Finally, in Figure 5, the MMSE of each network is plotted. The three lines represent the testing and training sets independently. Additionally, the MSE of the data set as a whole is plotted. Both of the architectures identified earlier (4/1 and 6/3/1) show a low training and overall error, but there is a significant divergence with testing error in the 6/3/1 architecture. Since four neurons in a single hidden layer captures high accuracy, high precision, consistent behavior for testing and training sets, and low standard deviation for $MMSE$, AC, and P, it is used as the architecture for differentiating cryptography and non-cryptography functions for the given dimensions.

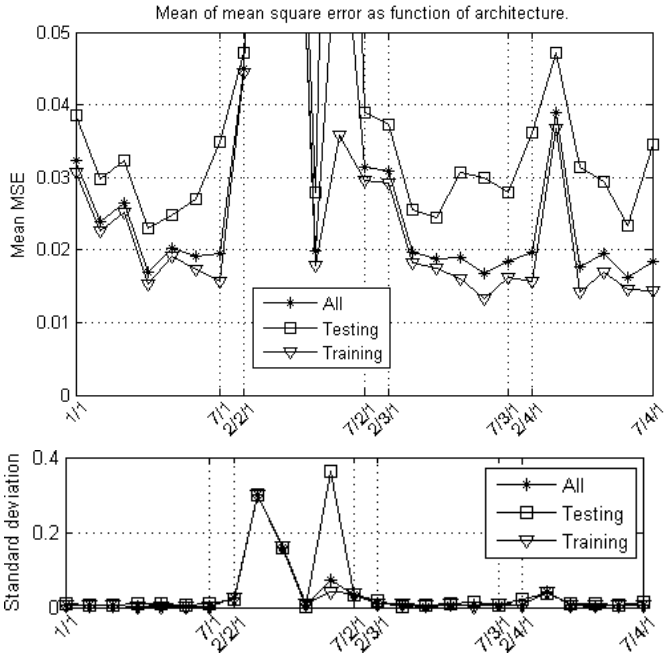

Fig. 5. Mean MSE vs. network architecture.

With the architecture chosen, the final network was chosen from the best fold of the best architecture: 8 inputs, 4 neurons in a hidden layer, and a single output neuron. The confusion matrix for this network is shown in Table III.

TABLE III
X86 NEURAL NETWORK PERFORMANCE (SINGLE).

classified as:		
non-crypto	**crypto**	
6829	7	**non-crypto**
30	182	**crypto**
Classification rate		
0.10%		**false positive**
14.5%		**false negative**
99.5%		**accuracy**
96.3%		**precision**

To summarize, the neural network selection process was run with 25 different neural networks. Each network was trained with 10 different data sets. One of the $k = 10$ folds was held out for validation. The MMSE, MAC, and MP was computed for each candidate architecture. By examining the MAC (Figure 3), all but two of the architectures were eliminated, 4/1 and 6/3/1. Examination of MP (Figure 4),

shows highest precision corresponding to the two remaining candidates. Finally, 4/1 is chosen as the final architecture because MMSE on the training and validation sets shows the lowest difference (Figure 5).

C. Neural Network Architecture Selection (SPARC)

The Scalable Processor Architecture (SPARC) is found on many server machines [14]. Unlike the Intel X86, the SPARC processor is a reduced instruction set computer (RISC). The distribution of instructions is likely to be different for these processors, and the reason for using it here is to demonstrate the applicability of the technique to alternate data sets. Table IV shows the opcodes used as inputs to the tested neural networks. ρx denotes the density of opcode x in each function, and σx denotes the total number of x in the function.

TABLE IV
SPARC OPCODES USED AS NETWORK INPUTS

$\rho$and	ρbpos	ρbrz	ρldu	ρret
ρsll	ρsra	ρst	ρxor	σsrl

Figure 6 shows the MMSE for the same candidate networks trained against data gathered for the SPARC architecture. Here, the highest accuracy occurs with a 7/3/1 network and also with a 7/2/1 network. For precision (Figure 7), a similar relationship exists, but 2/2/1 and 5/2/1 networks also have high precision. Simply based on the accuracy and precision, the 7/3/1 network is the best candidate (high precision combined with high accuracy). The 2/2/1 network is ruled out because while it has high accuracy and precision, it also has a high standard deviation of accuracy and precision.

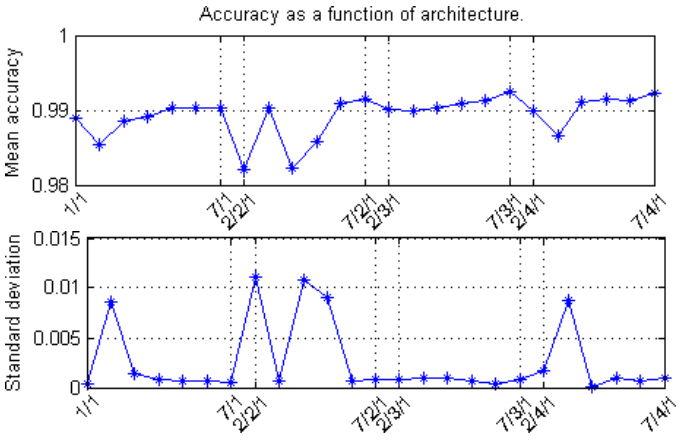

Fig. 6. Mean accuracy vs. network architecture (SPARC).

When MMSE is considered (Figure 8), the 2/2/1 and 5/2/1 networks which had high precision are ruled out again because of their high MMSE. The 7/3/1 network has a good combination of low MMSE, high accuracy, high precision, and low standard deviation on all measures.

The resulting confusion matrix for a single 7/3/1 neural network is shown in Table V. Comparing the results to the X86 neural network (Table III, the network used on SPARC

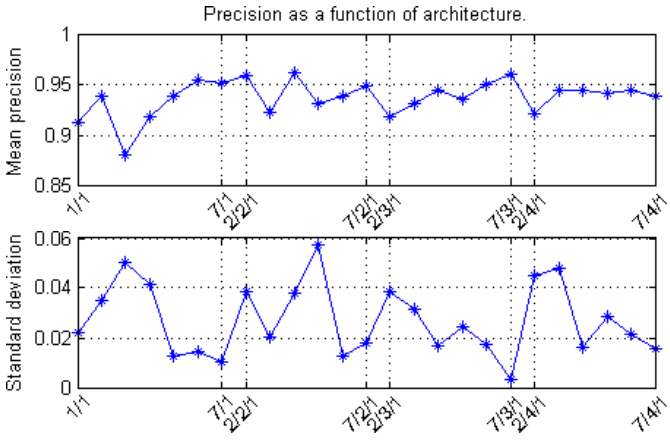

Fig. 7. Mean precision vs. network architecture (SPARC).

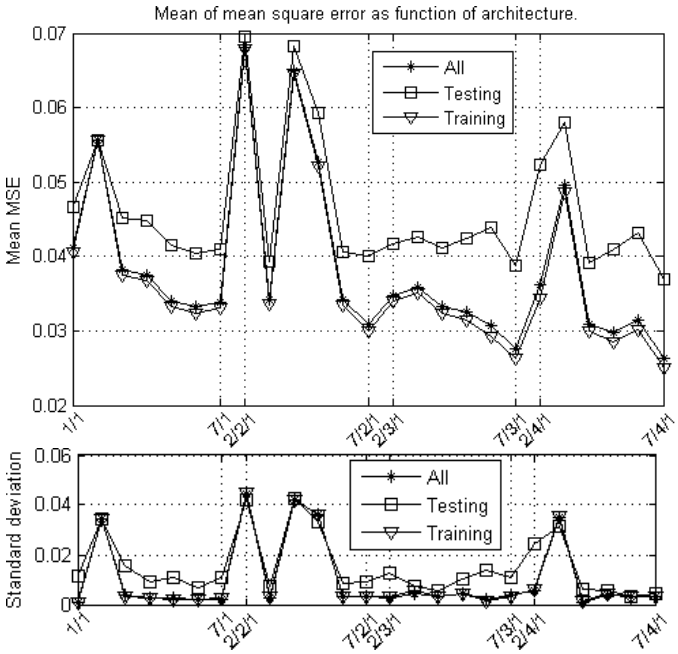

Fig. 8. Mean MSE vs. network architecture (SPARC).

performs similarly (within 0.1% on AC, FP, and FN and 5% on P). The two networks differ, however, in architecture (7/3/1 for SPARC, and 4/1 on X86); this underscores the difference in the instruction sets for the two processors (X86 is CISC and SPARC is RISC). The same neural network architecture cannot capture the behavior of the opcode counts and densities between CPU instruction set architectures.

In summary, the same neural network selection process was applied to the SPARC data set as was described in Section III-B. The MMSE, MAC, and MP was computed for each candidate architecture. By examining the MAC (Figure 6), all but two of the architectures were eliminated, 7/2/1 and 7/3/1. Highest precision occurs at 7/2/1, 7/3/1, 2/2/1, and 5/2/1 (Figure 7). Two networks, 2/2/1 and 5/2/1 are not considered because they do not correspond to high MAC. The final network chosen is 7/3/1 is chosen because it shows

TABLE V
SPARC NEURAL NETWORK PERFORMANCE (SINGLE).

classified as: non-crypto	classified as: crypto	
6829	7	**non-crypto**
41	171	**crypto**
Classification rate		
0.10%		**false positive**
19.3%		**false negative**
99.3%		**accuracy**
96.1%		**precision**

consistently low MMSE on both the test and validation sets (Figure 8).

IV. CONCLUSION

In this paper, a process was described that can be used to select the best neural network architecture for a given problem. The method relies on k-fold cross validation and training multiple networks for each candidate architecture. From there, the mean of the mean square error, accuracy, and precision are compared to determine the optimal architecture. In future work, this process will be refined and even more formalized.

The approach used in this work is known to have several requirements. Primarily, searching the entire space of possible networks exhaustively is computationally expensive. Secondly, the dataset should be divided into four, not three, parts. The first three divisions correspond to normal network training sets: training, testing, and validation sets. The fourth is then reserved for comparing different architectures.

The method was applied to the problem of locating cryptography algorithms in compiled object code for both the Intel X86 and SPARC instruction sets. The resulting networks show high accuracy, precision, and low mean square error. For this problem, the data must be stratefied because the number of positive (cryptographic) samples is much smaller than the number of negative samples. Future work on this problem will concentrate on locating additional measurement dimensions to increase the accuracy of cryptography detection and applying the described method to additional datasets.

REFERENCES

[1] R. Setiono, "Feedforward neural network construction using cross validation," *Neural Computation*, vol. 13, no. 12, pp. 2865–2877, 2001.

[2] E. Oja, "A simplified neuron model as a principal component analyzer," *Journal of Mathematical Biology*, vol. 15, no. 3, pp. 267–273, November 1982.

[3] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann, 1990, pp. 524–532.

[4] T. Q. Huynh and R. Setiono, "Effective neural network pruning using cross-validation," in *IEEE International Joint Conference on Neural Networks IJCNN '05*, vol. 2, July 2005, pp. 972–977.

[5] I. Gómez, L. Franco, and J. M. Jarez, "Neural network architecture: Can function complexity help?" *Neural Processing Letters*, vol. 30, no. 2, pp. 71–87, October 2009.

[6] Y. L. Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 1990, pp. 598–605.

[7] P. Porras, H. Saidi, and V. Yegneswaran, "An analysis of conficker's logic and rendezvous points," SRI International, Tech. Rep., March 2009. [Online]. Available: http://mtc.sri.com/Conficker/

[8] ——, "Conficker c p2p protocol and implementation," SRI International, Tech. Rep., September 2009. [Online]. Available: http://mtc.sri.com/Conficker/P2P/index.html

[9] K. Chiang and L. Lloyd, "A case study of the rustock rootkit and spam bot," in *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. Berkeley, CA, USA: USENIX Association, 2007, pp. 10–10.

[10] J. Wright and M. Manic, "Neural network approach to locating cryptography in object code," in *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, September 2009.

[11] J. L. Wright and M. Manic, "The analysis of dimensionality reduction techniques in cryptographic object code classification," in *IEEE Conference on Human System Interaction (HSI)*, May 2010.

[12] R. Kohavi and F. Provost, "Editorial: Glossary of terms," *Machine Learning: Special Issue on Applications of Machine Learning and the Knowledge Discovery Process*, vol. 30, no. 2-3, 1998.

[13] *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference A–M*, Intel Corporation, September 2009. [Online]. Available: http://www.intel.com/products/processor/manuals/index.htm

[14] D. L. Weaver and T. Germond, Eds., *The SPARC Architecture Manual (Version 9)*. SPARC International, Inc., 2000.