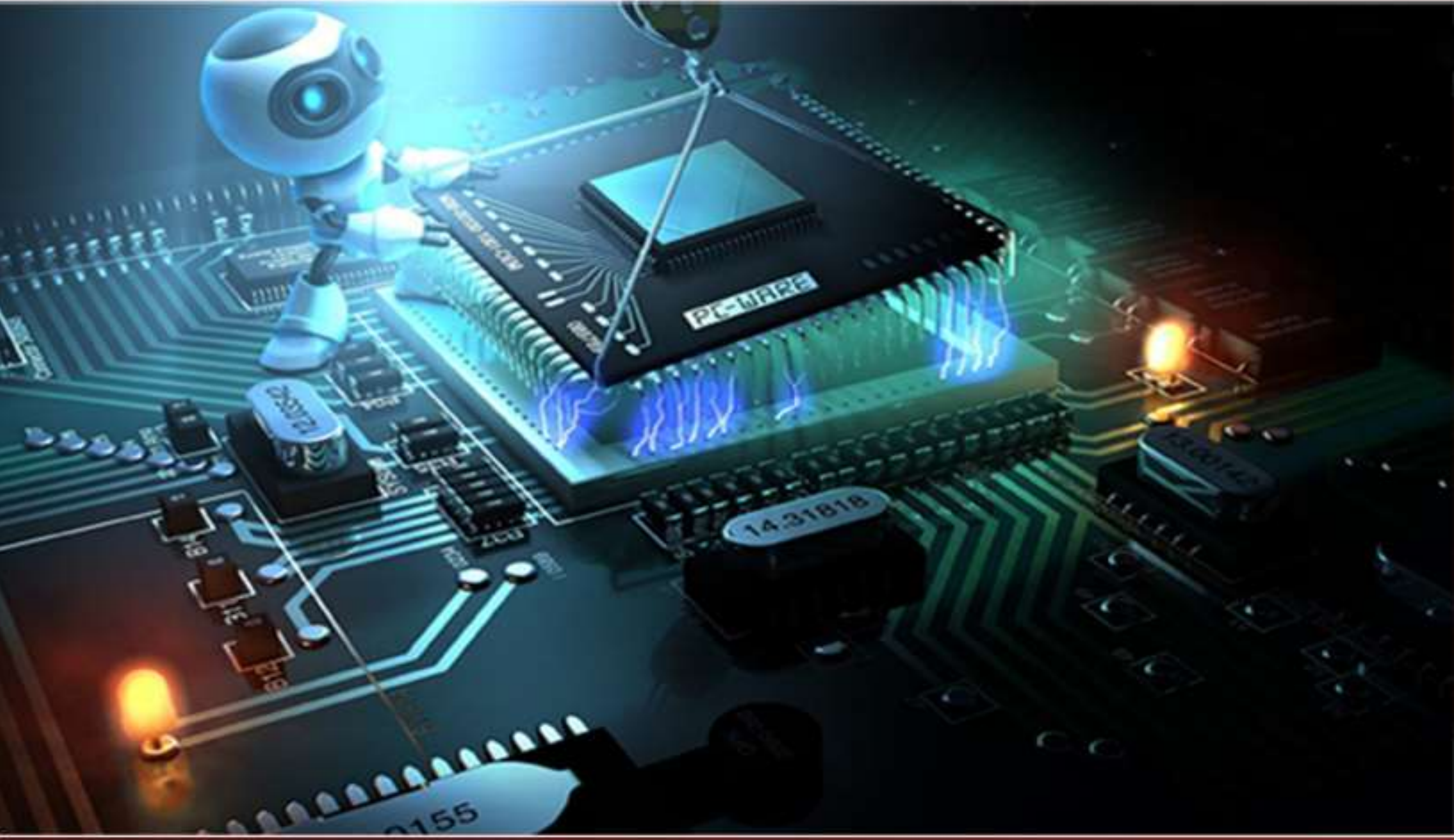
ISSN: 2054-7390

Volume 3 Issue 4, August 2015

Transactions on Machine Learning and Artificial Intelligence



Society for Science and Education
UNITED KINGDOM

TABLE OF CONTENTS

EDITORIAL ADVISORY BOARD

Society for Science and Education United Kingdom
TMLAI Transactions on Machine Learning and Artificial Intelligence
Volume 3 Issue 4
ISSN 2054-7390

Optimal Design of Continuous Reinforced Concrete Beams Using Neural Networks

[1]Jiin-Po Yeh and [2]Ren-Pei Yang
Department of Civil and Ecological Engineering, I-Shou University, Taiwan
[1]jpyeh@isu.edu.tw, [2]yunglnpei@gmail.com

ABSTRACT

This paper aims to build a neural network model to optimally design two-span continuous reinforced concrete beams. The training and checking data of the neural network are obtained by genetic algorithms, whose constraints are built according to the ACI Building Code and objective function is to find the minimum cost of longitudinal reinforcement, stirrups and concrete. The neural network adopted in this paper is the feed forward back propagation network, whose input vector consists of the span, width and effective depth of the beam, dead load, compressive strength of concrete as well as yield strength of steel and the output vector the positive and negative steel ratios and minimum total cost. The correlation coefficients between the target and network output of the testing data can reach as high as 0.9992, 0.9980 and 0.9999, respectively for the positive and negative steel ratios and minimum cost. Compared with the adaptive neuro-fuzzy inference system, the neural network shows almost the same accuracy but is much easier implemented.

Keywords: Continuous reinforced concrete beams, Genetic algorithms, Feedforward backpropagation networks, Correlation coefficients.

1 Introduction

Genetic algorithms are search procedures which mimic biological evolution. They can solve the optimization problems by the evolution theory of "survival of the fittest," whose constraints can be in the form of linear equality or inequality with bounds on the optimization variables. In 1970s, Professor John Holland first formally introduced its basic concept [1]. In 1989, Goldberg described in more detail the theory, terminology and its applications on machine learning and optimization [2]. From then on, the genetic algorithms became more attractive and widely used. Genetic algorithms have a variety of applications in fields like engineering, chemistry, economics, manufacturing and so on. Take civil engineering as an example. Many publications have been seen, such as multiobjective optimization of trusses [3], reliability analysis of structures [4], global optimization of grillages [5], global optimization of trusses with a modified genetic algorithm [6], optimization of pile groups using hybrid genetic algorithms [7], locating the critical slip surface in slope stability analyses [8-10], optimal design of reinforced concrete beams [11], etc.

Neural networks, as used in artificial intelligence, have traditionally been viewed as simplified models of neural processing in the brain. They are a computational tool based on the properties of biological neural systems, which are simplified models of neural processing in the brain and could capture and represent complex input/output relationships. The motivation for the development of neural network technology

DOI: 10.14738/tmlai.34.1303
Publication Date: 10th August, 2015
URL: http://dx.doi.org/10.14738/tmlai.34.1303

arose from the desire to have an artificial system to perform intelligent tasks similar to those done by humans. The artificial neural network was originated by McCulloch and Pitts in 1943 [12], which paved the way for neural network research. Rosenblatt [13] created the perceptron, an algorithm for pattern recognition based on a two-layer learning computer network using simple addition and subtraction. Also key later advance was the backpropagation algorithm by Werbos in 1975 [14]. In 1986, Rumelhart et al. [15] proposed the theory of parallel distributed processing that computed through the parallel cooperative and competitive interactions of a large number of simple neuron-like processing units in contrast to conventional programs that computed through the sequential application of stored commands. They developed the most famous learning algorithm to modify the weights on connections between these units so that the global error minimum could be achieved, which marked a milestone in the current artificial neural networks. Since then, a huge proliferation in the ANN methodologies and applications have been published, such as control of chaotic pendulum [16], flood forecasting [17], structural optimization [18-20], facilitating the accurate estimation of probabilistic constraints in optimization problem [21], frame optimization [22], traffic sign classification [23], modeling the financial market with multiple prices [24], etc.

2 Genetic Algorithms and Neural Networks

Because genetic algorithms could deal with high nonlinear constraints and neural networks could build complicated nonlinear relationships between inputs and outputs, this paper combines these two techniques to optimally design two-span continuous reinforced concrete beams and compares the results with the previous work [11], which uses the adaptive neuro-fuzzy inference system [25].

2.1 Genetic Algorithms

The genetic algorithm belongs to the field of artificial intelligence, which is a heuristic and multi-point search that imitates the course of natural selection. Without the need to find the derivatives of the objective function, it can solve optimization problems by the skills similar to natural evolution. To start the algorithm, the initial population of candidate solutions is generated at random across the search space, and then a sequence of new populations is produced. First, evaluate the fitness value of each member; Secondly, select members called parents according to their fitness, where fitter members are more likely to be chosen; Thirdly, retain a specified number of individuals, called elites, which have fittest values. They will pass to the next population unchanged; Fourthly, produce children by combining the vector entries of a pair of parents, i.e., crossover, where offspring under “crossover” will combine parental traits and will not be identical to its parents; Fifthly, make random changes to a single parent, i.e., mutation, which is more like walking randomly in the vicinity of a candidate solution; Finally, replace the current population with the crossover and mutation children and elites to form the next generation. The above steps are repeated until a terminating condition is met.

The optimization problem for two-span continuous reinforced concrete beams discussed in this paper can be formulated as follows:

Minimize $f(\boldsymbol{x})$

to find x

subject to

$$C_i(\boldsymbol{x}) \le 0,\ i=1,\ldots,m$$
$$C_j(\boldsymbol{x}) = 0,\ j=1,\ldots,n \qquad (1)$$
$$\boldsymbol{LB} \le \boldsymbol{x} \le \boldsymbol{UB}$$

where ***x*** is the vector of design variables, $C_i(\boldsymbol{x})$ and $C_j(\boldsymbol{x})$ represents the nonlinear inequality and equality constraints, respectively, $f(\boldsymbol{x})$ is the fitness function, which is the total cost of concrete, longitudinal reinforcement and stirrups, and ***LB*** and ***UB*** are the vectors of lower and upper bounds of design variables, respectively.

2.2 Artificial Neural Networks

The neural network used in this paper is the two-layer feedforward backpropagation network, whose structure is shown in Figure 1. The transfer function of the single hidden layer is the tan-sigmoid function

$$a_i = f(net_i) = \frac{2}{1+e^{-2net_i}} - 1 \quad , \quad i = 1,2,3,\ldots,k \qquad (2)$$

where k is the number of the neurons in the hidden layer, $net_i = w_{i1}P_1 + w_{i2}P_2 + \ldots + w_{iR}P_R + b_i$, $P_1, P_2, \ldots P_R$ are the inputs, R is the number of elements in the input veator, $w_{i1}, w_{i2}, \ldots, w_{iR}$ are the weights connecting the input vector and the ith neuron, and b_i is the bias of the ith neuron in the hidden layer. The output layer uses the linear transfer function

$$O_i = f(Net_i) = Net_i \quad , \quad i = 1,\ 2,\ldots,q \qquad (3)$$

where $Net_i = W_{i1}a_1 + W_{i2}a_2 + \ldots\ldots + W_{ik}a_k + B_i$, $W_{i1}, W_{i2}, \ldots, W_{ik}$ are the weights connecting the neurons in the hidden layer and the ith neuron of the output layer, B_i is the bias of the ith output neuron and q is the number of the network outputs. The square error between the network outputs and the targets can be expressed as

$$E_P = \sum_{j=1}^{q} (t_j - a_j)^2 \qquad (4)$$

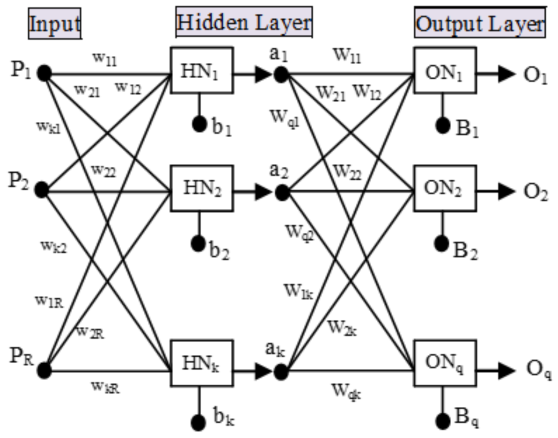


Figure 1 The structure of two-layer feed forward back propagation neural networks

where t_j and a_j are the jth target and network output, respectively, and p is a particular input-output pair. For the whole training set, the mean square error is given by

$$MSE = \frac{1}{m}\sum_{P=1}^{m} E_P \tag{5}$$

where *m* is the total number of training pairs. The Levenberg-Marquardt algorithm [26-28] is chosen as the training function to minimize the mean square error (i.e., the network performance function). If a tentative step increases the performance function, this algorithm will act like the gradient descent method, while it shifts toward Newton's method if the reduction of the performance function is successful. It interpolates between the quasi-Newton's algorithm and the gradient descent method; therefore, the mean square error in Eq. (5) will always be reduced at each iteration.

In order to generalize the network, the validation set is also presented to the network during the training process. When the network begins to overfit the training data, the error on the validation set typically begins to rise. Once the validation error increases for a specified number of iterations, e.g., 6 iterations, the training terminates and the weights and biases corresponding to the minimum of validation error are returned.

3 Design of Two-Span Beams and the Constraints

Based on the provisions of the ACI Building Code Requirements for Structural Concrete and Commentary [29], the constraints for the two-span continuous beam are formulated, taking into account the strength requirements of the maximum positive and negative moments, shear, the service requirement of deflection as well as the development length of flexural reinforcement. Design variables are the width b, effective depth d, positive steel ratio �1 and negative steel ratio �2 of two-span continuous reinforced concrete beams. The beams considered in this paper are subjected to a uniformly distributed load wu =1.2 wD+1.6 wL , as shown in Figure 2, where wD and wL are dead load and live load. The beams with span length L are singly reinforced and vertical stirrups are used. Top reinforcement in the negative moment region will be cut off, while there are no cutoffs for the bottom reinforcement to simplify the design, as shown in Figure 3. Force and length are measured in the units of kgf (=9.81N) and cm, respectively, for the following formulas.

3.1 Shear

Suppose that V_c is the shear capacity of the plain web concrete and V_u is the factored shear force. The shear diagram of the two-span continuous beam is shown in Figure 2, where V_A=0.375w_u L=V_E and V_C^+ =0.625w_u L=V_C^-. First, consider the segment AB in Figure 2. The design of web steel for shear may be considered by dividing the shear diagram into four regions: (1) Region I: If $V_u \le 0.5\phi V_c$, where ϕ=0.75 is the strength reduction factor, there is no need for shear reinforcement; (2) Region II: If $\phi V_c \ge V_u > 0.5\phi V_c$, a minimum web steel area

$$A_v \ge Max(0.2\sqrt{f_c'}\frac{bs}{f_y},\ \frac{3.5bs}{f_y}) \tag{6}$$

is required, where s is the spacing of vertical stirrups, fc� is the compressive strength of concrete and fy is the yield strength of steel. According to the ACI code, the spacing s must not be larger than Min (d/2, 60) cm. Because the size of stirrups is usually fixed along the span of the beam, the spacing

$$s \le \text{Min}\ (\frac{d}{2},\ 60,\ \frac{A_v f_y}{0.2\sqrt{f_c'}b},\ \frac{A_v f_y}{3.5b}) \qquad (7)$$

must be satisfied by combining Eq. (6) and the requirement of *Min* (*d*/2, 60) cm; (3) Region III: If $3\phi V_c \ge V_u > \phi V_c$, web reinforcement

$$V_s = \frac{A_v f_y d}{s} = \frac{V_u}{\phi} - V_c \qquad (8)$$

has to be provided to carry the difference and the spacing

$$s = \frac{\phi A_v f_y d}{V_u - \phi V_c} \qquad (9)$$

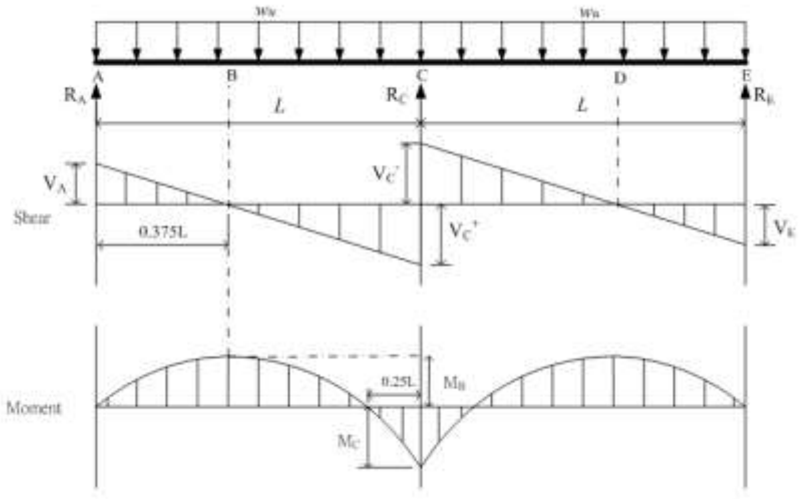


Figure 2 Two-span continuous reinforced concrete beam

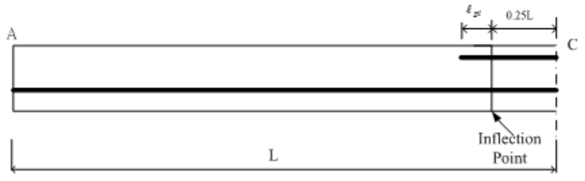


Figure 3 The top and bottom reinforcement in the left span

must not be larger than *Min* (*d*/2, 60, $\frac{A_v f_y}{0.2\sqrt{f_c'}b}$, $\frac{A_v f_y}{3.5b}$) cm; (4) Region IV: If $5\phi V_c \ge V_u > 3\phi V_c$, the web reinforcement in Eq. (8) similarly has to be provided to carry the difference, but the spacing *s* in Eq. (9) must not be larger than Min (d/4, 30, $\frac{A_v f_y}{0.2\sqrt{f_c'}b}$, $\frac{A_v f_y}{3.5b}$) cm.

Web reinforcement for segments BC, CD and DE shown in Figure 2 will be arranged in the same way as the segment AB. Because the reaction in the direction of applied shear introduces compression into the end regions of a member, the critical section is taken at a distance *d* from the face of support. If the factored shear force V_{ud} at a distance *d* from the face of the support is larger than $5\phi V_c$, the beam section has to be enlarged. Therefore, another constraint is given by

$$V_{ud} \le 5\phi V_c \tag{10}$$

After the spacing for each region is found, the total number of vertical stirrups can be computed right away.

3.2 Bending Moment

The moment diagram can be seen in Figure 2. The maximum positive moment is $M_B = 9w_uL^2/128$ located at 0.375*L* from *A* (or *E*) and the negative moment at the support *C* is M_c=0.125w_uL^2. For simplicity, the strain ε_t in the tension reinforcement is assumed to be equal to 0.005; therefore, the section is tension-controlled and the strength reduction factor for moment is fixed at 0.9, not a function of strain in the tension reinforcement any more. The constraint for both positive and negative moment then takes the form

$$M_u \le 0.9M_{n,0.005} \tag{11}$$

where M_u is the factored negative moment M_C or the factored maximum positive moment M_B, and

$$M_{n,0.005} = A_s f_y\left(d - \frac{1}{2}\times\frac{A_s f_y}{0.85 f_c' b}\right) \tag{12}$$

where the area of reinforcement

$$A_s = \frac{0.85 f_c'\beta_1}{f_y}\times\frac{3db}{8} \tag{13}$$

due to the net tensile strain of 0.005 in the extreme tensile reinforcement, and β_1 is the stress block depth factor. To prevent sudden failure with little or no warning when the beam cracks or fails in a brittle manner, the ACI code limits the reinforcement ratio to be between

$$\rho_{max} = \frac{0.85 f_c'\beta_1}{f_y}\left(\frac{3}{7}\right) \tag{14}$$

and

$$\rho_{min} = max\left(\frac{0.8\sqrt{f_c'}}{f_y}, \frac{14}{f_y}\right) \tag{15}$$

where ρ_{max} in Eq. (14) is to make sure that the tensile strain must be greater than or equal to 0.004. Therefore, the constraint for reinforcement ratio is given by

$$\rho_{min} \le \rho \le \rho_{max} \tag{16}$$

where ρ is the steel ratio ρ_1 for positive moment M_B or the steel ratio ρ_2 for negative moment M_c .

3.3 Development of Reinforcement

The ACI Code stipulates that at least one-third of the total tension reinforcement provided for negative bending moment at the support should extend beyond the inflection point (P.I) not less than the effective depth *d* of the member, 12d_b, or 1/16 of the clear span, as shown in Figure 3. The inflection point P.I. is located 0.25*L* from the support. For practical purposes, let span length $L \approx$ clear spam. Therefore, the length of top reinforcement in Figure 3

$$\ell_{top} = 0.25L + \ell_{pi} = 0.25L + Max\ (12d_b, d, \frac{L}{16}) \geq \ell_d \tag{17}$$

where ℓ_d is development length of tension reinforcement and d_b is the nominal diameter of the horizontal longitudinal bar.

3.4 Immediate and Long-Term Deflections

Because the deflection of the beam can be magnified by creep and shrinkage, both immediate and long-term deflection must be considered according to the ACI code. The creep and shrinkage deflection under sustained load can be evaluated using a multiplying factor

$$\lambda = \frac{\xi}{1 + 50\rho'} \tag{18}$$

where ρ' is the compression reinforcement ratio at midspan for simple and continuous beams and ξ is a time factor that is taken as 1.0 for loading time duration of 3 months, 1.2 for 6 months, 1.4 for 12months and 2.0 for 5 years or more, respectively. Suppose that the beam considered in this paper will support partitions and other construction likely to be damaged by large deflections. Hence, the long-term deflection

$$\Delta_{LT} = \Delta_{iL} + \lambda \Delta_{iD} \leq \frac{L}{480} \tag{19}$$

where Δ_{iL} is immediate live-load deflection and Δ_{iD} is immediate dead-load deflection under service loading.

4 Numerical Results

Given the span length, uniformly distributed dead and live loads, compressive strength of concrete and yield strength of steel of the two-span continuous singly reinforced concrete beams with rectangular cross-section, the optimal design is accomplished by using the genetic algorithm, whose design variables are the width *b*, effective depth *d*, positive steel ratio ρ_1 and negative steel ratio ρ_2, and the objective function is to find the minimum cost of concrete, longitudinal reinforcement and stirrups in New Taiwan Dollars. The unit prices are 1800 NT$/m^3 and 19.5 NT$/kgf for concrete and steel, respectively, in Taiwan. The concrete cover for the reinforcement is 4 cm and No. 3 vertical stirrups are used. On the basis of materials often used in Taiwan, this paper uses three kinds of yield strength f_y of the tension reinforcement: 2800 kgf/cm^2 (40 ksi), 3500 kgf/cm^2 (50 ksi) and 4200 kgf/cm^2 (60 ksi) as well as three kinds of compressive strength f'_c of the concrete: 210 kgf/cm^2 (3000 psi), 280 kgf/cm^2 (4000 psi) and 350 kgf/cm^2 (5000 psi). Besides, three kinds of span length: 6 m, 8 m and 10 m and four kinds of uniformly distributed dead load: 2100 kgf/ cm, 2300 kgf/cm, 2500 kgf/cm and 2700kgf/cm are adopted; uniformly distributed live load is fixed at 1800 kgf/cm. Hence, there are totally 108 cases to be designed, which will be used as training, validation and testing data for the neural networks.

4.1 Genetic Algorithms

The Global Optimization Toolbox based on MATLAB [30] is employed to execute the genetic algorithm, where some parameters are specified as follows: the population size 20, crossover rate 0.8, and elite number 2. In addition, all the individuals are real-number codes; "Rank" is taken as the scaling function to scale the fitness values; "Roulette" is the selection function to choose parents for crossover; "Two-point crossover" is the strategy to produce offspring; The mutation function "Adaptive Feasible Function" is applied to avoid being trapped into the local minimum. The genetic algorithm is executed 30 times for each case, from which the minimum cost is singled out. Then, the total 108 optimal sets of data are randomly divided into 3 groups: 64 training sets (60 %), 22 validation sets (20 %) and 22 testing sets (20 %).

4.2 Feedforward Backpropagation Neural Networks

This paper applies the Neural Network Toolbox based on MATLAB [31] to build the neural network, whose inputs of the neural network consist of six elements: f_y, f'_c, w_D, L, b as well as d, and targets have three components: the minimum cost, the steel ratios ρ_1 and ρ_2. Because the input vector has six elements, the number of neurons in the hidden layer is first set to be six by the past experience. The Levenberg-Marquardt algorithm is chosen to train the neural networks. The training process is shown in Figure 4. The weights and biases at epoch 64 are returned to the trained network. After the training of the network is completed, the testing set is then used to examine the network performance. The graphs of the network outputs and targets are plotted in Figs. 5-7. The inputs, targets and network outputs of the 22 sets of the testing data are listed Table 1. To further evaluate the performance of the trained network, this paper makes use of a linear regression analysis between the network outputs and the corresponding targets. The scatter plots are shown in Figs. 8-10. The regression results of the steel ratios $\rho 1$ and $\rho 2$ and the minimum cost (103 NT\$) are shown in Table 2, where the symbols m, b and r stand for the slope, the y-intercept and correlation coefficient, respectively. The correlation coefficients between the network output and target are 0.9992, 0.9980 and 0.9999, respectively for the positive and negative steel ratios and minimum cost. Besides, the slope m is close to 1 and y-intercept b approximately equals 0. Based on Figs. 5-10 and Table 2, the performance of the feedforward backpropagation networks can be considered excellent, which is as good as the adaptive neuro-fuzzy inference system. The networks with 12, 18 and 24 neurons in the hidden layer are also explored for comparison with 6 neurons. Their linear regression results can be seen in Table 2, which indicates that even the number of neurons used in the hidden layer increases, the network output accuracy doesn't improve significantly.

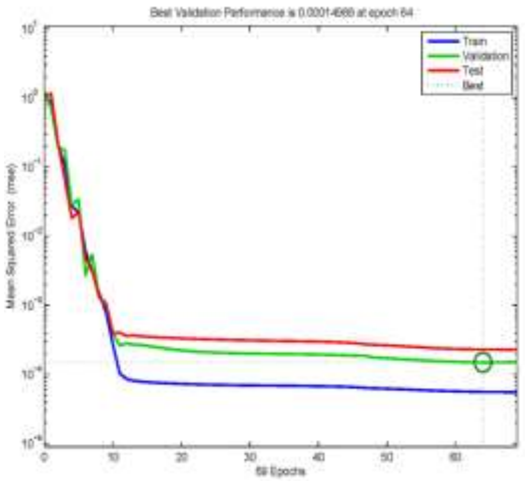


Figure 4 The training process for the neural network with 6 neurons in the hidden layer

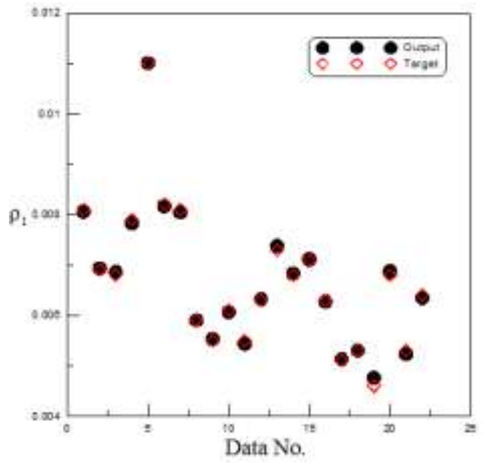


Figure 5 The network outputs and targets of the steel ratio of the positive moment for the testing sets with 6 neurons in the hidden layer

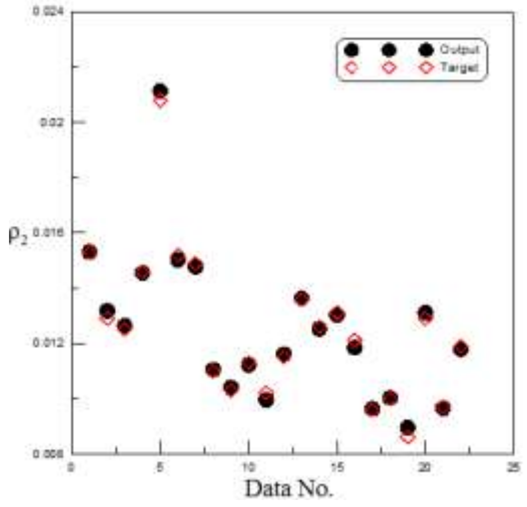


Figure 6 The network outputs and targets of the steel ratio of the negative moment for the testing sets with 6 neurons in the hidden layer

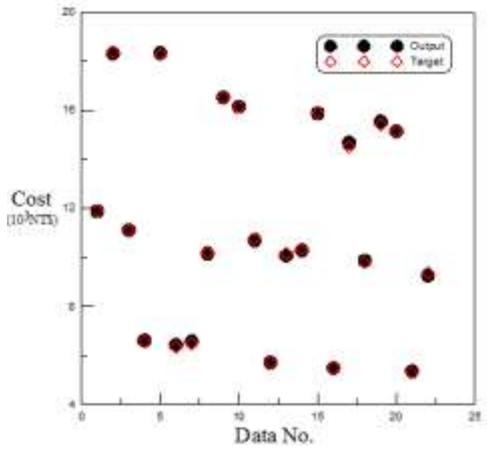


Figure 7 The network outputs and targets of the minimum cost for the testing sets with 6 neurons in the hidden layer

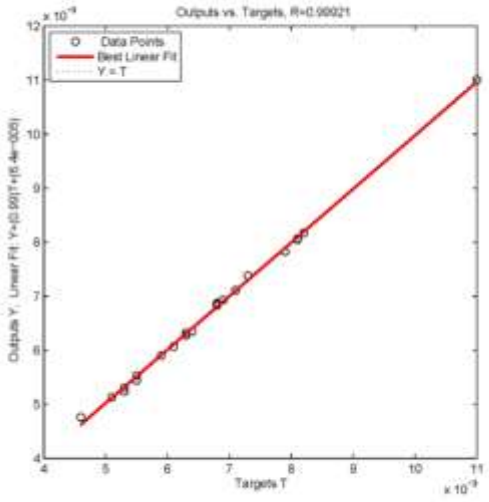


Figure 8 The scatter plot of the steel ratio of positive moment for the testing set with 6 neurons in the hidden layer

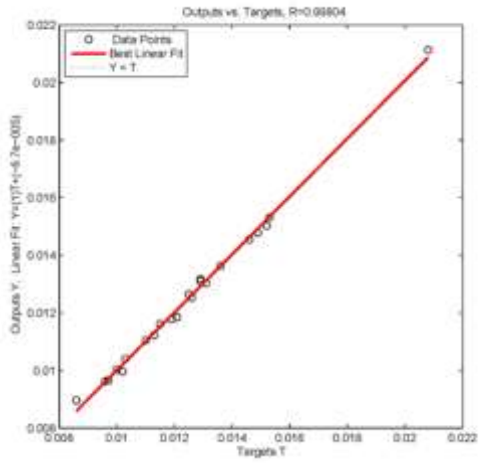


Figure 9 The scatter plot of the steel ratio of negative moment for the testing set with 6 neurons in the hidden layer

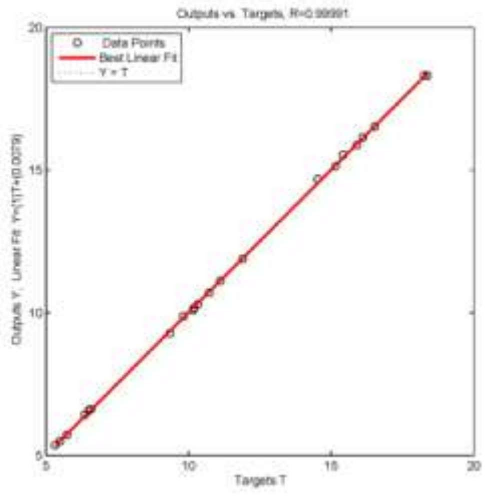


Figure 10 The scatter plot of the minimum cost (103 NT$) for the testing set with 6 neurons in the hidden layer

Table 1 Optimal results and outputs of the 22 sets of testing data for the neural network with 6 neurons in the hidden layer

Inputs and Targets of the network									Network Outputs		
Inputs						Targets					
f_y (ton/cm^2)	f'_c (ton/cm^2)	w_d (ton/m)	L (m)	b (m)	d (m)	ρ_1	ρ_2	Cost ($10^3 NT\$$)	ρ_1	ρ_2	Cost ($10^3 NT\$$)
2.8	0.21	2.5	8	0.2005	0.8312	0.0081	0.0153	11.894	0.0081	0.0153	11.885
2.8	0.21	2.5	10	0.2069	1.1012	0.0069	0.0129	18.376	0.0069	0.0132	18.311
2.8	0.28	2.1	8	0.2037	0.8513	0.0068	0.0125	11.109	0.0069	0.0126	11.114
2.8	0.28	2.3	6	0.2000	0.6147	0.0079	0.0146	6.5762	0.0078	0.0146	6.609
2.8	0.28	2.3	10	0.2021	0.8703	0.011	0.0208	18.26	0.0110	0.0211	18.329
2.8	0.35	2.1	6	0.2001	0.5848	0.0082	0.0152	6.3606	0.0082	0.0150	6.429
2.8	0.35	2.3	6	0.2001	0.6024	0.0081	0.0149	6.4999	0.0080	0.0148	6.580
3.5	0.21	2.1	8	0.2000	0.8349	0.0059	0.0110	10.189	0.0059	0.0110	10.165
3.5	0.21	2.5	10	0.2063	1.1032	0.0055	0.0103	16.532	0.0055	0.0104	16.526
3.5	0.28	2.5	10	0.2034	1.0536	0.0061	0.0113	16.109	0.0061	0.0112	16.146
3.5	0.28	2.7	8	0.2008	0.9053	0.0055	0.0102	10.724	0.0054	0.0100	10.697
3.5	0.35	2.1	6	0.2000	0.5986	0.0063	0.0115	5.7371	0.0063	0.0116	5.722
3.5	0.35	2.3	8	0.2002	0.7567	0.0073	0.0136	10.159	0.0074	0.0136	10.083
3.5	0.35	2.5	8	0.2000	0.8003	0.0068	0.0126	10.32	0.0068	0.0125	10.291
3.5	0.35	2.5	10	0.2013	0.9775	0.0071	0.0131	15.901	0.0071	0.0130	15.874
4.2	0.21	2.1	6	0.2001	0.5570	0.0063	0.0121	5.4984	0.0063	0.0118	5.497
4.2	0.21	2.1	10	0.2036	1.0145	0.0051	0.0096	14.527	0.0051	0.0096	14.681
4.2	0.21	2.5	8	0.2000	0.8393	0.0053	0.0100	9.8189	0.0053	0.0100	9.870
4.2	0.21	2.7	10	0.2004	1.1405	0.0046	0.0086	15.417	0.0048	0.0090	15.525
4.2	0.28	2.7	10	0.2006	0.9408	0.0068	0.0129	15.152	0.0069	0.0131	15.139
4.2	0.35	2.1	6	0.2000	0.5978	0.0053	0.0097	5.3274	0.0052	0.0097	5.360
4.2	0.35	2.3	8	0.2000	0.7411	0.0064	0.0119	9.3465	0.0063	0.0118	9.273

Table 2 The regression analysis of the targets with network outputs of the testing data for different number of neurons in the hidden layer

No. of neurons in the hidden layer	Targets vs. network outputs	*m*	*b*	*r*
6	ρ_1	0.9910	0.0001	0.9992
	ρ_2	1.0071	-0.0001	0.9980
	Cost	1.0004	0.0079	0.9999
12	ρ_1	1.0036	0.0000	0.9993
	ρ_2	1.0023	0.0000	0.9982
	Cost	0.9986	0.0351	0.9998
18	ρ_1	1.0185	-0.0001	0.9952
	ρ_2	1.0119	-0.0002	0.9977
	Cost	0.9709	0.2512	0.9972
24	ρ_1	1.0241	-0.0002	0.9981
	ρ_2	1.0341	-0.0005	0.9959
	Cost	0.9962	0.0046	0.9995

5 Conclusions

This paper uses the feedforward backpropagation neural network to form a design model for two-span continuous reinforced concrete beams with rectangular cross-section and compares the results with the adaptive neuro-fuzzy inference system. The training, validation and testing data are obtained from the optimal results of the genetic algorithm. The inputs of these models are the yield strength of steel, compressive strength of concrete, dead load (live load is fixed) and span length, width and effective depth of the beam, while the targets are the minimum cost, the steel ratios for the positive and negative moment. The reason for the inputs of the neural networks to be a little bit different from the given conditions of the genetic algorithm is to make the design model more practical. Numerical results show that the performance of the feedforward backpropagation neural network is excellent with all correlation coefficients being greater than 0.998, the slope and y-intercept of the regression line close to 1 and 0, respectively, which is as good as the adaptive neuro-fuzzy inference system. Furthermore, if more neurons in the hidden layer are used, the effectiveness of the neural network doesn't improve significantly. In addition to having the same accuracy, the feedforward backpropagation neural network is much easier to be implemented than the adaptive neuro-fuzzy inference system.

REFERENCES

[1] Holland, J. H. 1975. Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor, MI, USA.

[2] Goldberg, D. E. 1989. Genetic algorithms in search, optimization and machine learning, Addison Wesley, Reading, MA, USA.

[3] Coello, C. A. and Christiansen, A. D. 2000. Multiobjective optimization of trusses using genetic algorithms, Computers and Structures. Vol. 75, pp. 647-660.

[4] Cheng, J and Li, Q. S. 2008. Reliability analysis of structures using artificial neural network based genetic algorithms. Computer Methods in Applied Mechanics and Engineering, Vol. 197, No. 45. pp. 3742-3750.

[5] Belevičius, R. and Šešok, D. 2008. Global optimization of grillages using genetic algorithms, Mechanika. Nr., Vol. 6, No. 74, pp. 38-44.

[6] Šešok, D. and Belevičius, R. 2008. Global optimization of trusses with a modified genetic algorithm, Journal of Civil Engineering Management. Vol. 14, No. 3, pp. 147-154.

[7] Chan, C. M., Zhang, L. M. and Jenny, T. N. 2009. Optimization of pile groups using hybrid genetic algorithms, Journal of Geotechnical and Geoenvironmental Engineering. Vol. 135, Issue 4, pp. 497-505.

[8] Goh, A. T. C. 2000. Search for critical slip circle using genetic algorithms. Civil Engineering and Environmental Systems, Vol. 17, No. 3, pp. 181–211.

[9] Zolfaghari, A.R., Heath, A.C., McCombie, P.F. 2005. Simple genetic algorithm search for critical non-circular failure surface in slope stability analysis. Computer and Geotechnics, Vol. 32, No. 3, pp. 139-152.

[10] Li, Y.-C., Chen, Y.-M., Zhan, T. L. T., Ling, D.-S. and Cleall, P. J. 2010. An efficient approach for locating the critical slip surface in slope stability analyses using a real-coded genetic algorithm. Canadian Geotechnical Journal, Vol. 47, No. 7, pp. 806–820.

[11] Yeh, J-P and Yang R-P, 2014, "Application of the Adaptive Neuro-Fuzzy Inference System for Optimal Design of Reinforced Beams," Journal of Intelligent Learning Systems and Applications, Vo. 6, pp. 162-175.

[12] McCulloch, W.S. and Pitts, W. 1943. A logical calculus of ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, Vol. 5, pp. 115-133.

[13] Rosenblatt, F. 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. Psychological Review, Vol. 65, No.6, pp. 386–408.

[14] Werbos, P. J. 1975. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Ph.D. Dissertation, Harvard University, Boston, MA, USA.

[15] Rumelhart, D. E., McClelland, J. L. and the PDP Research Group. 1986. Parallel distributed processing: explorations in the microstructure of cognition. volume 1: foundations, MIT Press, Cambridge, MA, USA.

[16] Bakker, R., Schouten, J. C., Takens, F. and van den Bleek, C. M. 1996. Neural network model to control an experimental chaotic pendulum. Physical Review E, 54A, pp. 3545-3552.

[17] Mukerji, A., Chatterjee, C., and Raghuwanshi, N. S. 2009. Flood forecasting using ANN, Neuro-Fuzzy and Neuro-GA models. Journal of Hydrologic Engineering, Vol. 14, No. 6, pp. 647-652.

[18] Cheng, J. and Li, Q. S. 2009. A hybrid artificial neural network method with uniform design for structural optimization. Computational Mechanics, Vol. 44, No. 1, pp. 61-71.

[19] Möller, O., Foschi, R. O., Quiroz, L. M., and Rubinstein, M. 2009. Structural optimization for performance-based design in earthquake engineering: applications of neural networks. Structural Safety, Vol. 31, No. 6, pp. 490–499.

[20] Gholizadeh, S. and Salajegheh, E. 2010. Optimal design of structures for earthquake loading by self organizing radial basis function neural networks. Advances in Structural Engineering, Vol. 13, No. 2, pp. 339-356.

[21] Patel, J. and Choi, S. K. 2012. Classification approach for reliability-based topology optimization using probabilistic neural networks. Structural and Multidisciplinary Optimization, Vol.45, Issue 4, pp. 529–543.

[22] Meon, M. S., Anuar, M. A., Ramli, M. H. M., Kuntjoro, W., and Muhammad, Z. 2012. Frame optimization using neural network. International Journal Advanced Science Engineering Information Technology, Vol. 2, No. 1, pp. 28-33.

[23] Ciresan, D. C., Meier, U., Masci, J. and Schmidhuber, J. 2012. Multi-column deep neural network for traffic sign classification. Neural Networks, Vol. 32, pp. 333-338.

[24] Danting, Z., Tingfeng, T., Zhongwei, J., Huichao, X., Xian, L. and Qing-Guo, W., 2014. Modeling the financial market with multiple prices. Transactions on Machine Learning and Artificial Intelligence, Vol. 2, No. 5, pp. 41-51.

[25] Jang, J.-S. R. 1993. “ANFIS: Adaptive-Network-Based Fuzzy Inference System”, IEEE Transactions on Systems, Man and Cybernetics, Vol. 23, No.3, pp.665-685.

[26] Hagan, M. T. and Menhaj, M. 1994. Training feedforward networks with the Marquardt algorithm. IEEE Transactions on Neural Networks, Vol.5, No.6, pp.989-993.

[27] Hagan, M. T., Demuth, H. B. and Beale, M. H. 1996. Neural network design. PWS Publishing, Boston, MA, USA.

[28] Pujol, J. 2007. The solution of nonlinear inverse problems and the Levenberg-Marquardt method. Geophysics, Vol. 72, No. 4, W1–W16.

[29] ACI 2008. Building code requirements for structural concrete (ACI 318-08) and commentary (ACI 318R-08). American Concrete Institute, Farminton Hills, MI, USA.

[30] The MathWorks. 2012. Global optimization toolbox: user's guide. The MathWorks, Inc., Natick, MA, USA.

[31] Demuth, H., Beale, M., and Hagan, M. 2010. Neural network toolbox user’s guide. The MathWorks Inc., Natick, Massachusetts, USA.

Society for Science and Education United Kingdom
TMLAI Transactions on Machine Learning and Artificial Intelligence
Volume 3 Issue 4
ISSN 2054-7390

Application of Hybrid Machine Learning to Detect and Remove Malware

[1]Richard R. Yang, [2]Victor Kang, [3]Sami Albouq and [3]Mohamed A. Zohdy
[1]College of Engineering and Applied Sciences, University of Wyoming, United States;
[2]College of Literature, Science and the Arts, University of Michigan Ann Arbor, United States;
[3]School of Engineering and Computer Science, Oakland University, United States;
ryang3@uwyo.edu; vkang@umich.edu; salbouq@oakland.edu; zohdyma@oakland.edu

ABSTRACT

Anti-malware software traditionally employ methods of signature-based and heuristic-based detection. These detection systems need to be manually updated with new behaviors to detect new, unknown, or adapted malware. Our goal is to create a new malware detection solution that will serve three purposes: to automatically identify and classify unknown files on a spectrum of malware severity; to introduce a hybrid machine learning approach to detect modified malware traces; and to increase the accuracy of detection results.

Our solution is accomplished through the use of data mining and machine learning concepts and algorithms. We perform two types of data mining on samples, extracting n-grams and PE features that are used for our machine learning environment. We also introduce a new hybrid learning approach that utilizes both supervised and unsupervised machine learning in a two-layer protocol. A supervised algorithm is applied to classify if a file is considered malware or benign. The files classified as malware will then be categorized and then assigned on a severity spectrum using the SOFM unsupervised algorithm.

Keywords: machine learning, data mining, self-organizing feature map algorithm, malware detection.

1 Introduction

Due to the advancements and increased use in technology, malware is becoming more prevalent in today's society. Malware are widely categorized, but the main types we investigate are: Adware, Trojans, Keyloggers, and Rootkits. The purpose of Adware is to obtain users internet behavior for commercial purposes such as targeted ads. A Trojan disguises itself as legitimate or useful software but actually infiltrates a user's system. Keyloggers record keystrokes and other user input to a system. Finally, a Rootkit attempts to gain elevated control of a user's system while hiding itself and other system components. Each category of malware seeks to accomplish a different purpose, but their overarching goal is to cause malicious intent and to steal information from users without their consent.

Currently, anti-malware solutions are able to fight back new malware through two approaches: signature-based detection and heuristic-based detection. In signature-based detection, a suspicious file is obtained by the anti-malware firm and analyzed. If the file is deemed to be malware, a unique signature is generated for the file and that signature is added to an online database. The anti-malware solution is able to quarantine files that match malicious signatures on clients systems. Although this method has been

DOI: 10.14738/tmlai.34.1436
Publication Date: 26th August, 2015
URL: http://dx.doi.org/10.14738/tmlai.34.1436

reliable and commonly used, the downside is that anti-malware firms must readily obtain samples and manually update the database each time new or modified malware surface. Compared to signature-based detection, heuristic-based detection generalizes malware by using a single signature for any of the malware's variations or mutations. However, systems that use heuristic-based detection often has a high number of false positives [3]. Though these methods have been seen to be effective, we choose to investigate a machine learning alternative to these current detection methods.

The two primary categories of machine learning are supervised learning and unsupervised learning. In supervised learning, the system is given a set of training data and how that data is classified. The algorithm is able to classify new input by learning from the training set that it was provided. The results from supervised learning are more specific than its counterpart. In unsupervised learning, the system is provided only with data and no classification keys. The idea of unsupervised learning is to classify relatively similar inputs together without any prior classification knowledge about the data set.

In our research, we expand the application of data mining and machine learning concepts for malware detection and introduce a new hybrid machine learning approach that incorporates both supervised and unsupervised learning in a two-step classification process.

2 Related Work

Alternatives to signature-based detection and heuristic-based detection in security software have been proposed and examined in the past [2] [5] [6] [7] [10].

In [7], the authors use data mining and natural language processing on End-User License Agreements (EULA) during software installation. The authors' objective were to detect spyware that may have been installed with the user's consent through the EULA, where the EULA may have been written in a way that is hard to comprehend for users. The study compared 17 learning algorithms with a baseline algorithm on a bag-of-words and meta-data model of representing the EULA.

Another method of detection was introduced by [10], where a Support Vector Machine is used in conjunction with static and dynamic analysis of file samples. Static analysis of samples were performed by dumping the program's import table from the Portable Executable structure, and recording all of the DLLs used and API function calls. For dynamic analysis, the authors ran each sample in a VMWare sandbox environment and monitored changes in registry, system folders/files, and network states. The results of each type of analysis were used as features in the Support Vector Machine, and information gain was calculated to observe the influence of feature selection.

A system known as Early Detection, Alert and Response (eDare) was designed to detect malicious code in network traffic in [6]. The eDare system uses network traffic scanners and machine learning algorithms to "pinpoint unknown malicious code exhibiting suspicious morphological patterns." [6] In this investigation, static code analysis is performed using decision trees, neural networks, and Bayesian networks.

Unsupervised learning was introduced to detecting malware in wireless devices by [2]. In [2], k-Means Clustering, γ-Algorithm, Divisive Hierarchical Clustering, Agglomerative Hierarchical Clustering, and Quartersphere Support Vector Machine unsupervised learning algorithms were applied to features data mined from Android applications.

The approach of using data mining and supervised machine learning was investigated in [5]. The authors extract n-grams from file samples and apply supervised learning algorithms to classify files as purely spyware or benign. N-grams of various sizes (centered on n = 5) and several supervised learning algorithms were compared to create a better performing solution than existing anti-virus software.

3 Experimental Procedure

Our experimental procedure begins with collecting benign and malware samples from several trusted websites. Next, we data mine and extract features from the samples. A feature is an identifying characteristic of the file used in machine learning algorithms. In our research, we use two general types of features: N-grams and Portable Executable structure. After data mining and obtaining features from our samples, we proceed with our hybrid machine learning approach. First, the features are organized in a database and we run it through supervised learning algorithms that will strictly classify if a file is malware or benign. From there, we use all of the files that were identified as malware and use them as input to an unsupervised learning algorithm to classify the malware on a severity spectrum. Finally, we evaluate our approach by computing the true positive rate, false positive rate, overall accuracy, and area under the receiver operating characteristics curve for each trial.

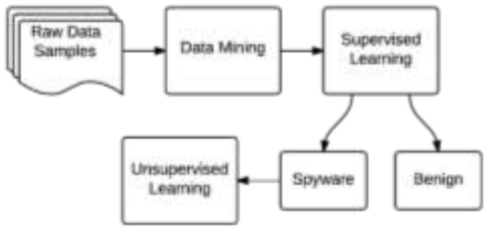


Figure 1: A graphical representation of our experimental procedure

3.1 Sample Collection

Initially, we collected 300 benign files and 120 files containing malware. The benign files were downloaded from two web sources, Download.com and Softpedia.com. These websites both certify that their files are malware free. The benign files we downloaded contain a wide variety of software, including business applications, games, web browsers, and developer tools. The overall size of our benign sample collection is 5.05 GB.

The malicious files were downloaded through collections from VXHeaven and KernelMode.info, both are online communities focusing on malware reversal, development, and research. The malware in the collections are categorized by their behaviors. We primarily downloaded malware that were categorized under Backdoor, Rootkit, Trojan, and Worm. The overall size of our malicious sample collection is 1.28 GB.

3.2 Data Mining/Feature Extraction

With the samples we collected, we now data mine all of the files for features. A feature is an identifying characteristic of the file, and we extract two general types of features from each file: N-grams and Portable Executable structure.

3.2.1 N-grams

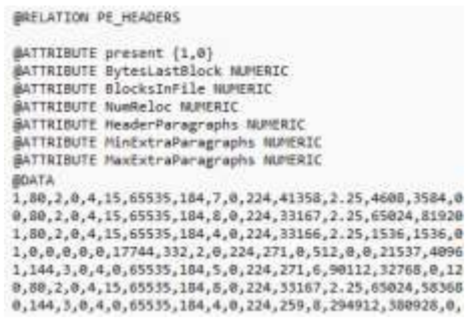
N-grams are sequences of n bytes extracted from a sample's machine code. N-grams are the most naive form of features from a data mining perspective, and can be viewed as DNA strands for virtual files. We first generate a hexadecimal dump - the raw computer data as seen in memory - from each sample. After the hexadecimal dumps are obtained, we select a size of n to sequence the dump into n-grams. Past research from [5] [8] has shown that n-grams of size 5 (each sequence is exactly 5 bytes) produced the most overall accurate results. The dumping and sequencing is performed using a Linux utility called "xxd" and can be used on any Linux distribution.

3.2.2 Portable Executable Features

The Portable Executable structure contains information about how the operating system manages a resources allocated to a program [4]. Features from the PE Header, MZ Header, and Data-Directory were extracted and compared from each sample's structure. In the results section, we compare the performance of our algorithms using selected features from each section of the PE structure. In addition, we extracted the DLL imports and API function calls of each file from the Import Address Table [9]. The PE feature mining was done by through a package from Ruby called "pedump".

3.3 Supervised Machine Learning

The goal of the supervised machine learning implementation is to categorize file inputs as strictly benign or malware. We use the Waikato Environment for Knowledge Analysis (WEKA) as our platform for running supervised algorithms and analyzing performance evaluations. WEKA uses a unique file format as input, an Attribute-Relation File Format (ARFF) database. The format allows us to list all of our features and their type (numeric, nominal, string, etc.), an example can be seen in figure 1. ARFF files for n-grams consist of two attribute fields: the n-gram sequence (numeric) and a "present" field, which indicates if an n-gram is found in malware traces or not (nominal). ARFF files for PE features also contain a present attribute field (nominal), all of the PE fields (numeric), and API function calls (nominal). The ARFF file generation is done through a parser that we created in C++.



```
@RELATION PE_HEADERS

@ATTRIBUTE present {1,0}
@ATTRIBUTE BytesLastBlock NUMERIC
@ATTRIBUTE BlocksInFile NUMERIC
@ATTRIBUTE NumReloc NUMERIC
@ATTRIBUTE HeaderParagraphs NUMERIC
@ATTRIBUTE MinExtraParagraphs NUMERIC
@ATTRIBUTE MaxExtraParagraphs NUMERIC
@DATA
1,80,2,0,4,15,65535,184,7,0,224,41358,2.25,4608,3584,0
0,80,2,0,4,15,65535,184,8,0,224,33167,2.25,65024,81920
1,80,2,0,4,15,65535,184,4,0,224,33166,2.25,1536,1536,0
1,0,0,0,0,0,17744,332,2,0,224,271,0,512,0,0,21537,4096
1,144,3,0,4,0,65535,184,5,0,224,271,6,90112,32768,0,12
0,80,2,0,4,15,65535,184,8,0,224,33167,2.25,65024,58368
0,144,3,0,4,0,65535,184,4,0,224,259,8,294912,380928,0,
```

Figure 2: An example ARFF database generated for a sample's PE features

For n-grams, we first dump all benign and malware files into separate databases. The two databases are then merged into a new database with duplicate n-grams removed, which we will call the common database. N-grams from the common database is compared with the n-grams in the malware database, and if the n-gram is present in the malware database, we indicate so by assigning the n-gram's present field in the ARFF file with a "1."

All of the PE features are numeric, so their values and directly saved into the ARFF database. Due to the large quantity of API function calls, we selected the top 100 most frequent calls among malware and used them as nominal fields in the ARFF database. If a sample performs the specified function call, we assign a "1" to that field and a "0" otherwise.

After the ARFF files have been prepared, we use them as input to our WEKA simulator. WEKA uses a method known as k-fold cross validation to test the supervised algorithms, where the entire data set is partitioned into k sections and one section is used as testing data while the rest are used as the training data. Through this process WEKA is able to compute the true positive rate, false positive rate, and other statistic quantifiers of the algorithm. The three supervised algorithms that we investigated are:

3.3.1 J48 Algorithm

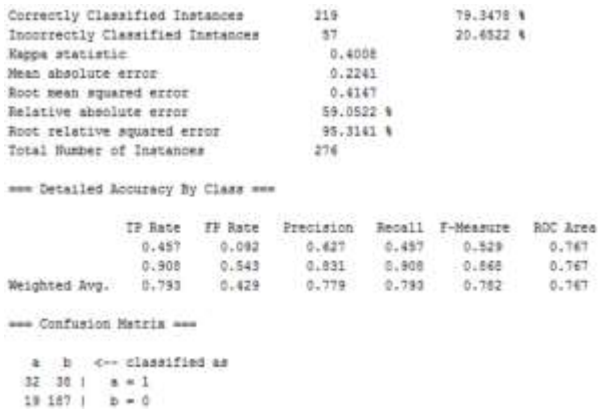
An implementation of the C4.5 decision tree algorithm in Java. This algorithm calculates the normalized information gain ratio from splitting on a feature for all features. The decision tree will create a node that splits on the feature with the highest information gain.

3.3.2 Random Forest

The Random Forest algorithm is an ensemble learning method. It creates several decision trees from the random parts of the data with replacement. The combination of these trees, which contain an approximate of the underlying data, creates a "forest" which gives a much more accurate approximation of the data.

3.3.3 Naïve Bayes Classifier

The Naïve Bayes classifier assumes that all features are independent of each other, and then applies Bayes' Theorem to calculate which node the decision tree will split on.



```
Correctly Classified Instances         219               79.3478 %
Incorrectly Classified Instances        57               20.6522 %
Kappa statistic                          0.4008
Mean absolute error                      0.2241
Root mean squared error                  0.4147
Relative absolute error                 59.0522 %
Root relative squared error             95.3141 %
Total Number of Instances              276

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area
                 0.457     0.092      0.627     0.457     0.529      0.767
                 0.908     0.543      0.831     0.908     0.868      0.767
Weighted Avg.    0.793     0.429      0.779     0.793     0.782      0.767

=== Confusion Matrix ===

   a   b   <-- classified as
  32  38 |   a = 1
  19 187 |   b = 0
```

Figure 3. Output information from WEKA after running a supervised machine learning algorithm on an input sample

3.4 Unsupervised Machine Learning

After the supervised learning algorithms label the samples as malware and benign, the samples classified as malware are then used as input for the unsupervised machine learning. The goals of using unsupervised learning after the supervised algorithm are to classify malware on a severity spectrum and to act as a second buffer to catch any additional false positives. The primary algorithm that was used for unsupervised machine learning was the Self Organizing Feature Map (SOFM).

The SOFM is an artificial neural network that uses neurons that respond to input [1]. The algorithm takes in a data file that is organized similarly to the ARFF format used by WEKA, and outputs a grid of colors where each circle is a neuron.

In order to classify the malware based on their behavior, the features used as input into the SOFM needed to be representative of the functionality of a program. As a result, API function calls, which are found in the DLL imports of the PE structure were used, as they provided the best insight into the behavior of a sample. Our malware samples contained a total of 172,641 different API calls.

Each class of malware has different API function calls which are needed to perform their malicious activity. In our collection of malicious files, there were 4 different classes of malware, including: Backdoor, Rootkit, Trojan, and Worm. In each class, we recorded the top 100 most frequent API calls, ending up with a database of the top identifying API calls of each type of malware.

Using our database of the most frequent API calls of each class, we created a feature vector of 1's and 0's (1 if a certain API call was present, 0 if it was not) for each sample that was labeled "malware" by the supervised learning.

The SOFM we were working with used colors to represent different clusters. The colors are represented numerically with 3 inputs, Red, Green and Blue. To reduce our feature vector to just 3 numbers, we split the vector into groups of 3, and converted the binary sequence into a decimal number. The purpose of this was to give us a one-to-one mapping of the different binary sequences to a unique decimal value. We then took the decimal values and normalized them into a value between 0 and 1, which was finally used as input for the SOFM.

4 Results and Analysis

4.1 Performance Metrics

The performance of each algorithm was compared through the following evaluation metrics:

True Positives (TP): The number of malware correctly identified as malware

False Positives (FP): The number of malware incorrectly identified as benign

True Negatives (TN): The number of benign correctly identified as benign

False Negatives (FN): The number of benign incorrectly identified as malware

True Positive Rate (TPR): Also known as sensitivity,

$$TPR = \frac{TP}{TP + FN}$$

True Negative Rate (TNR): Also known as specificity,

$$TNR = \frac{TN}{TN + FP}$$

Overall Accuracy (ACC):

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

In addition to these metrics, we also use a Receiver Operating Characteristics (ROC) curve, which is used in many applications outside of Computer Science to evaluate detection performance. The ROC is a comparison graph of the False Positive Rate (1 - specificity) vs. True Positive Rate (sensitivity). In addition to the curve itself, the *area under the ROC curve* (AUC) can also be used as an evaluation metric in comparison to the overall accuracy. An ideal ROC curve is a unit-step function, where 100% TPR corresponds to 0% FPR and the AUC is exactly 1.0.

4.2 Parameters

For the supervised learning algorithms, we compared the results of each algorithm under different parameters. These included: malicious file percentage (MFP), combinations of different features, feature extraction methods.

It was found that changing the malicious file percentage (MFP) in the training data had a moderate impact on our results. As we increased the number of either benign or malicious files significantly over the other, a negative impact was seen on the performance of our solution. This was due to the fact that with huge imbalances in our sample, the algorithm would generalize files more often, as it was trained with more of one type the other. However, we also found that an exact split in the number of files did not work as well as a slightly skewed split.

Another parameter we changed were the different combinations of features used as input for WEKA. Within the Portable Executable structure, there are many sections of data that can be interpreted. These include the MZ header, PE File Header, Data Directory, and DLL imports (contain API function calls) for a total of 42 unique features. The top 100 most frequent API calls were also recorded, for a total of 142 features. Individually, these sections performed worse than when used in combination with each other. It is interesting to note that when using the MZ header features by themselves, the result was entirely meaningless as it provided an AUC of 0.5.

For feature extraction, we used both information gain and information gain ratio to select our attributes. These extraction methods reduce the amount of "noise" created by useless features that do not tell us anything regarding whether the program is malicious or benign. As a result, all features contained within the MZ header were removed. The number of features were reduced from 142 to 73. In most cases, applying information gain was the most ideal, as it lowered the FPR while the TPR remained relatively the same. On the other hand, using information gain ratio lowered the FPR approximately 2-3 percent, however the TPR suffered a significant reduction. As a result, we determined that information gain was the better feature selection as it maximized the distance between the TPR and the FPR, as well as having a slightly higher AUC.

A comparison of the algorithms reveals that the Random Forest algorithm is the best on average in every metric. It consistently outperformed both the J48 algorithm and the NBTree algorithm, most noticeably in regards to the AUC. The average AUC of Random Forest was 0.9467, while both J48 and NBTree AUC's falling under 0.9. With higher AUC values, the Random Forest algorithm maximized the difference between the TPR and FPR, thus producing the best results.

4.3 Comparison and Optimal Results

As mentioned earlier, the PE structure contains several categories of features in different locations of the structure. The three categories we looked at were the PE header (PE), Data-Directory (DD), and MZ header (MZ). In addition, we extracted the top 100 API function calls for a given set of samples and used them as features. Table 1 is a comparison of our top performing solutions. We found that the Random Forest algorithm almost always produced better results when run under the same conditions as the other algorithms. Also, it was observed that when only the MZ header features are used, all supervised learning algorithms returned inconclusive results.

Table 1. Performance Comparison of Varying Parameters

Features	Algorithm	ACC	TPR	FPR	AUC
PE + DD + API	Random Forest	93.5%	90%	5%	0.960
PE + DD + MZ + API	Random Forest	93%	90%	5.7%	0.959
DD + API	Random Forest	91%	86.7%	7.1%	0.963
PE + API	J48	90.5%	85%	7.1%	0.875
PE + DD + MZ + API	J48	90%	81.7%	6.4%	0.895
MZ + PE + API	Naïve Bayes	91%	88.3%	7.9%	0.912
API	Naïve Bayes	89%	88.3%	10.7%	0.918

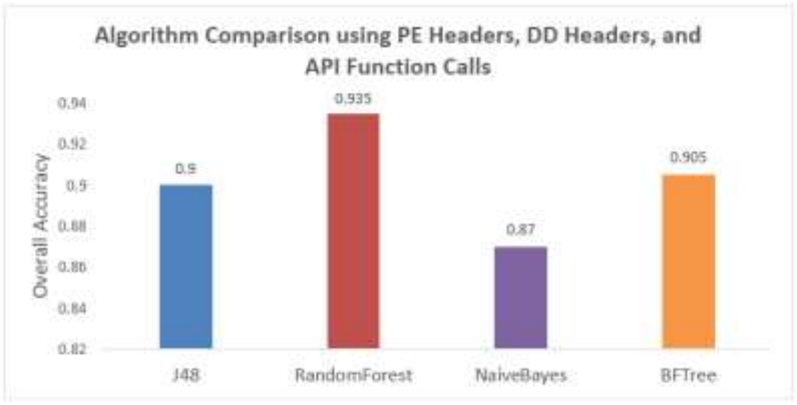


Figure 4. Comparison of algorithms' overall accuracy using the same parameters

In figure 4, we compare the overall accuracy of each algorithm using the same parameters. The features used were PE headers, DD headers, and API function calls. We saw the highest performing algorithm was RandomForest, which we use to further test the effectiveness of feature selection.

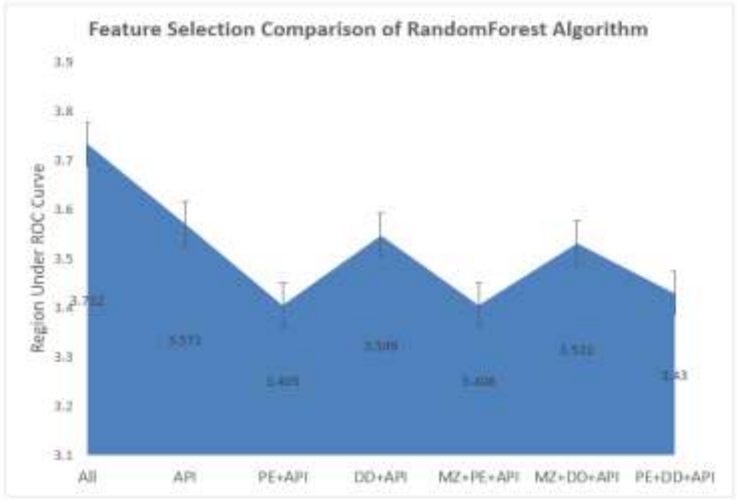


Figure 5. A comparison of area under the ROC curve using different features with the Random Forest algorithm

After running multiple trials, using different combinations of each parameter, we achieved our best result. This was done with the following parameters: 70% benign files to 30% malicious files, done under the Random Forest Algorithm, using the PE Header in conjunction with the Data Directory and top 100 most frequent API calls, all applied with information gain. The result was a feature vector containing 73 identifying features for malware. **This gave us an overall accuracy of 94%, a TPR of 91.7%, an FPR of 5% and an AUC of 0.962.** While we did achieve higher TPR and lower FPR, we defined the "best" result as the trial that maximized the difference between the TPR and the FPR.

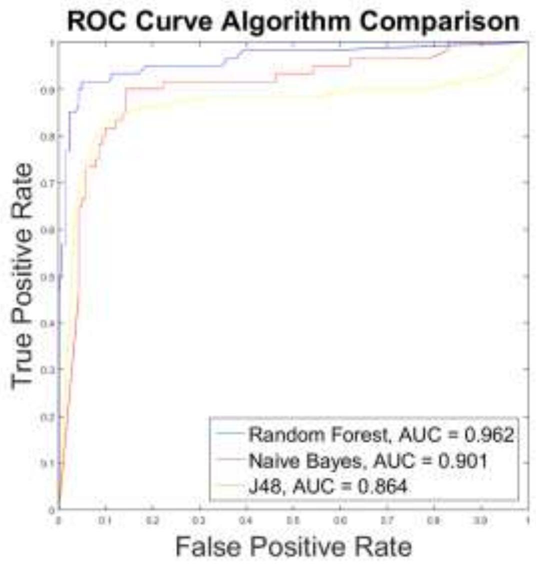


Figure 6. The ROC curve comparison for the top performing parameters of each supervised algorithm. Our overall best solution is shown in blue.

4.4 Self-Organizing Feature Map Classification

The SOFM algorithm we implemented represents neurons as colors by having three fields for RGB values ranging between 0 and 1. Our input data, the API function calls, need to be reduced to three fields and then normalized. The normalized value of each input is calculated by:

$$X_{i,norm} = \frac{X_i - X_{min}}{X_{max} - X_{min}}$$

In addition, we calculated the 6-norm distance from each input to the neuron rather than the Euclidean distance used in general SOFMs. The equation to calculate the p-norm distance from input to neuron is as shown below. Using the 6-norm provided a clearer output spectrum and each category of malware can be easily seen.

$$L_p = \left(\sum_{i=1}^{n} |q_i - p_i|^p \right)^{1/p}$$

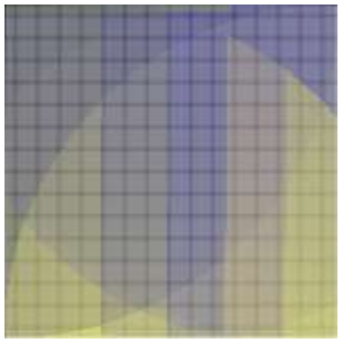
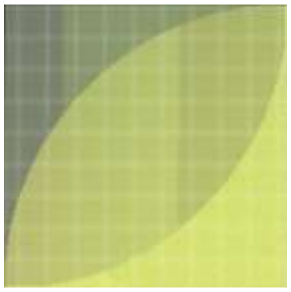


Figure 7. Snapshot of the SOFM in the process of clustering related malware into a severity spectrum

For an epoch of 3000 iterations and an initial learning rate of 0.3, were able to generate the severity spectrums shown in Figure 4 and 5. For the sake of clarity on the color clusters, we used a total of 22,500 (150x15) neurons in the network.

Figure 8. A completed SOFM output severity spectrum after 3000 iterations, each category of malware can be distinguished from the different colors of neurons



5 Conclusion and Future Work

In our investigation, we introduced a new method of hybrid machine learning to detect spyware and classify spyware on a severity spectrum. We explored the variations of parameters and their effect on the accuracy of detection. It was discovered that the best performing solution uses features mined from the PE structure and API function calls selected through information gain applied with the Random Forest algorithm. We achieved a 94% overall accuracy with 91.7% TPR and 5% FPR.

For future work, we suggest the use of more features in conjunction with each other. Our research combined the use of PE features with API function calls, using information gain to reduce our features. Additional features that could be examined are: opcode n-grams, and string analysis to find keywords representing maliciousness. The combination of these features used with the correct feature extraction would minimize the counts of false positives.

For the unsupervised machine learning, more algorithms in clustering and neural networks can be explored. Algorithms such as K-means clustering and hierarchical clustering could provide more accurate groupings of malware. In addition, a non-color based representation of the SOFM can be used. Disregarding color will give additional depth and dimensions to the SOFM, allowing it to cluster samples in more than just a two dimensional grid.

6 ACKNOWLEDGEMENTS

This research work was conducted at Oakland University in the UnCoRe program - REU site supported by the National Science Foundation under Grant No. CNS-1460897. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

The authors would like to thank Dr. Huirong Fu, PI and Program Coordinator of the UnCoRe REU program at Oakland University, for her support and guidance.

REFERENCES

(1) Abdel-Aty-Zohdy, H. S., & Zohdy, M. A. (2001). Self-organizing feature maps. Wiley encyclopedia of electrical and electronics engineering () John Wiley & Sons, Inc. doi:10.1002/047134608X.W5114

(2) Akpojaro, J., Aigbe, P., Onwudebelu,U. (2014). Unsupervised machine learning techniques for detecting malware applications in wireless devices. Transactions on Machine Learning and Artificial Intelligence, 2(3)

(3) Bahraminikoo, P., Samiei yeganeh, M., & Babu, G. P. (2012). Utilization data mining to detect spyware. IOSR Journal of Computer Engineering, 4(3)

(4) Baldangombo, U., Jambaljav, N., & Horng, S. (2013). A static malware detection system using data mining methods. Corr, abs/1308.2831

(5) Chavan, M. k., & Zende, D. A. (2013). Spyware solution: Detection of spyware by data mining and machine learning technique. International Conference on Advanced Research in Engineering and Technology, Vijayawada, India.

(6) Elovici, Y., Shabtai, A., Moskovitch, R., Tahan, G., & Glezer, C. (2007). Applying machine learning techniques for detection of malicious code in network traffic.4667, 44-50. doi:10.1007/978-3-540-74565-5_5

(7) Lavesson, N., Boldt, M., Davidsson, P., & Jacobsson, A. (2011). Learning to detect spyware using end user license agreements. Knowledge and Information Systems, 26(2), 285-307.

(8) Shahzad, R. K., Haider, S. I., & Lavesson, N. (2010). Detection of spyware by mining executable files. Availability, Reliability, and Security, 2010. ARES '10 International Conference on, Krakow. 295-302.

(9) Singhal, P., & Raul, N.Malware detection module using machine learning algorithms to assistin centralized security in enterprise networks.

(10) Wang, T., Horng, S., Su, M., Wu, C., Wang, P., & Su, W. (2006). A surveillance spyware detection system based on data mining methods. IEEE Congress on Evolutionary Computation, Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada. 3236-3241.

Society for Science and Education
United Kingdom
TMLAI Transactions on Machine Learning and Artificial Intelligence
Volume 3 Issue 4
ISSN 2054-7390

Bio-Inspired Temporal-Decoding Network Topologies for the Accurate Recognition of Spike Patterns

Gianluca Susi[1,2]
[1]Department of Electronic Engineering,
[2]Department of Civil Engineering and Computer Science
University of Rome "Tor Vergata", Italy.
gianluca.susi@uniroma2.it

ABSTRACT

In this paper will be presented simple and effective temporal-decoding network topologies, based on a neuron model similar to the classic Leaky Integrate-and-Fire, but including the spike latency effect, a neuron property able to take into account that the firing of a given neuron is not instantaneous, but it occurs after a continuous-time delay depending on the inner state. These structures are able to detect spike sequences composed of pulses belonging to neuron ensembles, exploiting basic biological neuron mechanisms. According to the biological counterpart, with these structures is possible to achieve a high temporal accuracy, but also deal with the natural variability present in spike trains. In addition, the connection of these neural structures at a higher level make possible to afford some pattern recognition problems, operating a distributed and parallel input data processing.

Keywords: Temporal coding; Neuronal modelling; Spiking Neural Network (SNN); Latency; Pattern recognition; Classification; Coincidence detection.

1 Introduction

In sensory systems the recognition of relevant stimuli is made possible by the detection of spike patterns during the processing of peripheral inputs. However, specific neural mechanisms that enable recognition of such sequences are currently unknown [1].

'Decoding' refers to the problem of how to 'read out' the information contained in a set of neural spike trains and has both theoretical and practical implications for the study of neural coding [2, 3]. Some works highlight that neurons represent information through their mean rates of action potential firing [4], on the other hand data suggest that the temporal structure of spike trains take advantage of reliable and precise latencies [5, 6]. In [7] is shown how time coding and rate coding tend to become confounded, at very short time scales (10 ms or less), and any form of temporal code can be described in terms of very rapid changes in firing rate [8].

Temporal spike coding, together with temporal spike learning, has been revealed in the auditory- and visual information processing in the brain as well as in motor control [9], and its use in neuro-prosthetics is essential along with applications for a fast, real-time recognition and control of sequence of related processes [10]. In [7] is highlighted the relevance of considering the timing of spikes across a population of afferent neurons.

DOI: 10.14738/tmlai.34.1438
Publication Date: 29th August, 2015
URL: http://dx.doi.org/10.14738/tmlai.34.1438

In this paper, very simple and effective temporal-decoding network topologies will be presented, based on the spike latency effect, an intrinsically continuous neuron property able to take into account that the firing of a given neuron is not instantaneous, but occurs after a continuous-time delay [11], depending on the inner state. With these structures is possible to process spike trains arisen by neuron ensembles. Also considering other bio-plausible phenomena, in the next sections a method for the tuning of such structures will be presented.

2 Neuron and network model

The structures presented in this paper are based on the Leaky Integrate and Fire with Latency neuron model [12, 13]. This kind of neuron is characterized by a real non-negative *inner state* (S), and is able to work in two modes of operation, depending on whether S is above or below a certain value; this value, called *firing threshold* (S_{th}), is equal to $1 + d$, where d is a positive value called *threshold constant.*

Passive mode is the operating mode of the neuron when its inner state is less than S_{th} ($S < S_{th}$); otherwise, the neuron is said to be in *active mode* ($S \geq S_{th}$). In the passive mode the neuron is a simple integrator, characterized by a linear *subthreshold decay* with constant Ld, such that in a time interval between two consecutive input spikes, Δt , the inner state decreases of a quantity $T_l = Ld\Delta t$. Conversely, in the active mode the neuron is ready to generate a spike. However, the output spike is not produced immediately, but after the *latency* time, called in the model *time-to-fire* (t_f). The inner state and the time-to-fire are related through the following bijective relation, called *firing equation*

$$t_f = \frac{1}{S-1} \ . \qquad (1)$$

As time passes, the time to fire decreases, and inner state grows up, but it remains sensitive to possible excitatory or inhibitory input spikes. For $S = S_{th}$ (i.e., the mimimum inner state possible in active mode) the time-to-fire is equal to $t_{f,max} = 1/d$ (i.e., the maximum time-to-fire).

Once the $t_f = 0$, the neuron definitively generates a spike. The amplitude of each spike generated by the network is characterized by the product $P_j = P_r \cdot P_w(x,j) = P(x,j)$. The quantity $P_w(x,j)$ (or equivalently P_{wj} if the target neuron is implicitly indicated) is the *postsynaptic weight*, that characterizes the connection strength from the neuron j to a certain neuron x. The quantity P_r , is the *presynaptic weight*, that represents the value of the spike amplitude as it has been generated by the firing neuron.

Once the neuron generates a spike, its state is reset to its *resting potential* (conventionally, $S = 0$) for a time defined as the *absolute refractory period* (t_{arp}).

The inner state evolution of a neuron in response to a spike sequence is illustrated in Fig. 1, where two incoming spikes modify the initial inner state S_{p0} (i.e., *previous-state-0*) of a target neuron.

For the purpose of emulating a continuous-time behavior, an event-driven approach is required, as described in [13], where the network model is characterized by the following global parameters: P_r *,* Ld*,* d (and then, S_{th}); in the present work the same global parameters are considered.

For the sake of simplicity, in the structures presented in this paper zero-delays axonal connections are considered, and synaptic plasticity phenomena are not taken in account. However, the relations that will be obtained below remain compatible with such cases.

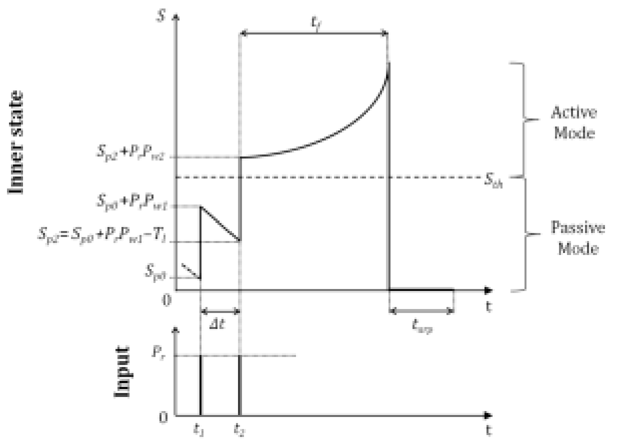


Figure 1. An example of the inner state behavior of a neuron in both passive and active modes is illustrated. An incoming excitatory pulse at t_1 causes an instantaneous increase of the state from S_{p0} to $S_{p0}+P_rP_{w1}$. When, at t_2 , a second excitatory pulse is applied, the state increase his value from S_{p2} to $S_{p2} + P_rP_{w2}$. The firing is not instantaneous but occurs after t_f. Finally, after the firing, the neuron is reset to its resting potential for a time equal to t_{arp} [13].

3 Dynamic summation in bio-plausible neuronal structures

A spike sequence can be quite exhaustively defined by amplitudes and arrival times of its components. Many attempts described in scientific literature are pointed to perform the recognition of specific spike sequences through bio-plausible neural structures (e.g., [14-16]). Typically, the recognition process is signaled by the activation of an indicator unit (e.g., a single neuron, a particular combination of neurons, or a whole network); with the aim of simplifying the analysis, in this paper a single neuron (called target neuron, TN) is used as indicator unit, so that its firing identifies that the recognition of the sought input spike sequence has occurred.

The purpose of this section is to introduce a key-mechanism of the neuron, here indicated as dynamic summation. Such mechanism can be exploited for the realization of many simple structures whose operation takes in account precise spike times of the input sequence. In particular, basic aspects of such mechanism will be explained in the next subsection.

3.1 Target neuron and working mode activity level

The task of a *TN* is to generate an output spike when a number of contributions is received in a certain time interval (Fig. 2). Actually, the target neuron becomes active when the following two conditions are simultaneously satisfied:

- the sum of the incoming spike amplitudes is greater than S_{th};
- the input spikes are properly synchronised.

For an exhaustive explanation of the subthreshold decay effect and the interplay between pulse amplitudes and arrival times characterizing the spike sequence, the reader may consult [17].

With reference to Fig. 2, in order for the *TN* to generate a spike, the following relation has to be satisfied:

$$P_r\ (P_{wa} + P_{wb} + P_{wc}) - Ld\ (\Delta t_1 + \Delta t_2) > S_{th} \qquad (2)$$

where P_r is the pre-synaptic weight, P_{wa}, P_{wb} and P_{wc} are the post-synaptic weights of the branches afferent to *TN* and Δt_1 , Δt_2 are the *interspike intervals* (*ISIs*) characterizing the specific spike sequence.

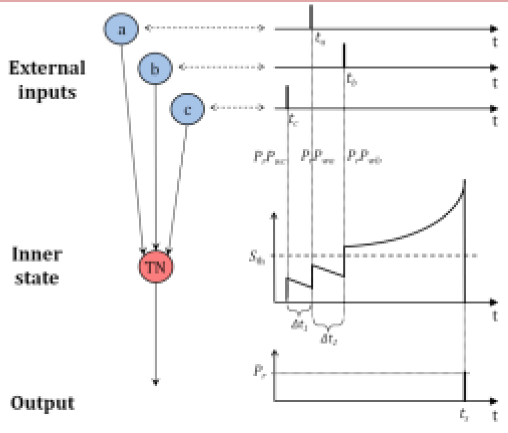


Figure 2. A spike sequence composed of three pulses coming from different neurons is sent to a *TN*, which operates the dynamic summation. The crossing of *S_{th}* by the inner state of *TN* let the neuron produce a spike; the intervals *Δt_1* and *Δt_2* represent the *ISIs* [18] of the sequence. The role played by the subthreshold decay phenomenon on the need of synchronisation of the input pulses is clearly notable [17].

The *working mode activity level* (*WMAL*) is a parameter that explicates the number of input spikes, simultaneous and of equal amplitude, necessary to lead the inner state of *TN* over the threshold, starting from the resting potential (Fig. 3). Of course, the concept is related to a condition upon the weights of the afferent branches; it is particularly useful to consider the *WMAL* for the synthesis of structures oriented to the recognition of spike sequences.

In the design phase, hypothesizing the same value for the weights of the afferent branches, the *WMAL* is obtained taking into account the weight $P_j = P_rP_{wj}$ (identical for the neurons afferent to the *TN*) and the threshold value S_{th}:

to obtain *WMAL* = 2, it must be imposed

$$\frac{S_0}{2} < P_j < S_0 , \tag{3}$$

to obtain *WMAL* = 3, it must be imposed

$$\frac{S_0}{3} < P_j < \frac{S_0}{2} , \tag{4}$$

to obtain *WMAL* = n, it must be imposed

$$\frac{S_0}{n} < P_j < \frac{S_0}{n-1} . \tag{5}$$

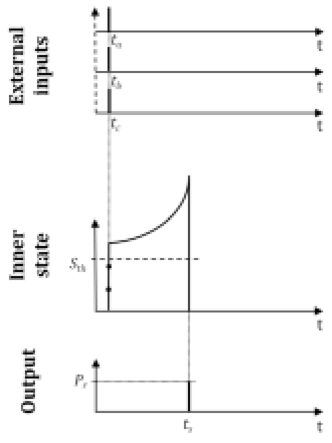


Fig. 3. Case of *WMAL* = 3. In reference to the architecture shown in Fig. 2, three identical and simultaneous pulses ($P_{wa} = P_{wb} = P_{wc}$; $t_a = t_b = t_c$) are necessary to let *TN* spike. A smaller amplitude or number of input spikes, or a smaller synchronism among them, could make the *TN* not able to generate the output spike.

3.2 Time-amplitude uncertainty

The inner state of a neuron at a time t0 does not provide information on the previous neuron activity: even though the initial state of the neuron is known, after a certain interval of activity it can reach a given inner state in infinite manners.

In this paper will be presented bio-inspired structures for the recognition of spike patterns. In the tuning phase it must be considered that the components of the input spike-sequences can present timing errors, or amplitudes not equal to the expected ones. With the aim of analyzing in detail this issue, in this paragraph will be shown the time-amplitude uncertainty phenomenon.

Useful relations for the design phase of detector structures will be given considering the internal state of the neuron as a function of its previous state, both for passive and active modes (i.e., respectively, subthreshold and overthreshold), hypothesyzing a temporal neighbourhood in which the neuron does not switch operation mode (i.e., no threshold crossing).

3.2.1 Subthreshold uncertainty

Considering a neuron in passive mode is straightforward to express the inner state as a function of its previous state. Said

$$S_1 = S(t_1) \tag{6}$$

and

$$S_2 = S(t_2) \tag{7}$$

it can be written

$$S_2 = S_1 - K_d (t_2 - t_1) \tag{8}$$

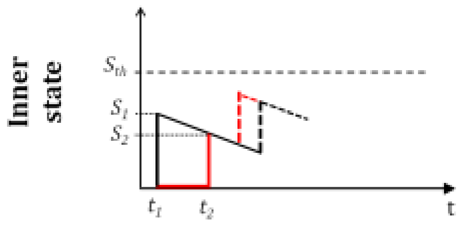


Figure 4: Subthreshold uncertainty phenomenon: assuming that at time t_2 the internal state of the neuron is S_2, there is uncertainty on the activity that has lead the system on this state. With the aim of highlighting such uncertainty, a compatible *subthreshold curve* is plotted (in red).

3.2.2 Overthreshold uncertainty

Also considering a neuron in active mode it is possible to express its inner state as a function of its previous state. Said

$$S_1 = S(t_1) \tag{9}$$

$$S_2 = S(t_2) \tag{10}$$

we obtain

$$t_{f_1} = \frac{1}{S_1 - 1} \tag{11}$$

$$t_{f_2} = \frac{1}{S_2 - 1} \tag{12}$$

where

$$t_{f_2} = t_{f_1} - (t_2 - t_1) = t_{f_1} - t_2 + t_1 = \frac{1}{S_1 - 1} - t_2 + t_1 \tag{13}$$

therefore

$$S_2 = 1 + \frac{1}{t_{f_2}} = 1 + \frac{1}{\frac{1}{S_1 - 1} - t_2 + t_1} \tag{14}$$

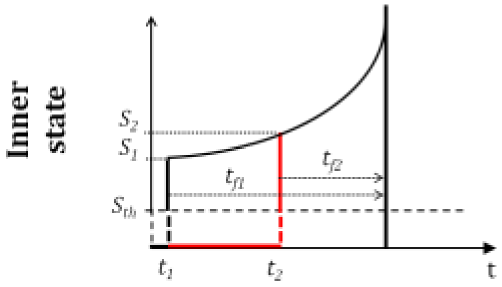


Figure 5: Overthreshold uncertainty phenomenon: assuming that at time t_2 the internal state of the neuron is S_2, there is uncertainty on the activity that has generated such a situation. With the aim of highlighting such uncertainty, a compatible *overthreshold curve* is plotted (in red).

During the design of the structures proposed in the following sections, it will not be always needed to specify both all the amplitudes of the input pulses, and all the *ISIs* of the spike sequence. Nevertheless, with the aim of unambiguously identify a sequence, it may be useful to refer to the relations obtained above.

4 Spike-timing sequence detectors

Spike-timing sequence detectors (STSD), are classes of structures able to reveal specific ISIs present in input sequences. These structures provide also the opportunity to set the tolerance of the detection in respect to the spike timings of the sequences to recognize.

Different kinds of realization are possible for these structures: they may differ in the number of spike trains (i.e., input lines) covered, number and type of neurons involved, or other aspects, like the number of neuronal phenomena exploited in the computation. In the course of this discussion, an order of presentation based on their complexity of operation will be followed.

In the structures presented in this paper, spike sequences composed of pulses belonging to different neurons will be considered, as parallel spike trains coming from a neural ensemble [19, 20, 16]. By operating on some of the parameters presented above it is possible to tune the structures on particular intervals, in order to detect specific sequences.

The relations between the times of the input spikes arrival and the effective firing of the target neuron will be shown by specifying the activation domain, and the operation of the presented structures will be shown by means of temporal diagrams.

4.1 Activation domains

In the following sections are presented different classes of structures able to detect specific *ISIs* between couples of pulses of an input sequence. Brain regions involved in object recognition in general must deal with the natural variability present in spike trains [1]. To take account of this necessity of flexibility, with the aim of maintain a bio-plausible behaviour, in the class of structures presented in this paper, a certain tolerance upon the arrival times of the input pulses can be set. Fig. 6 illustrate different kind of activation domains, namely the variation ranges of the input spikes, for which the structure is still able to detect the input sequence and to activate the TN.

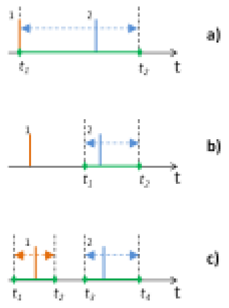


Figure 6: Different *activation domain* classes: *single* (a,b) and *multiple* (c). Green lines represent the *tolerance ranges* on the timings of the input spikes (orange and blue).

Note that, with the aim of facilitating the approach, in this paper the arrival time of the pulse from EI_1 will always considered preceding that of EI_2 (i.e., $t_2 > t_1$), as shown schematically in Fig. 6.

4.2 Direct STSD

Direct STSD is the simplest class of structures able to reveal specific *ISIs* present in input sequences.

The properties of these structures strictly depend on the global parameters (P_r , Ld, d), and on the value of P_t , defined as the common post-synaptic weight of the synapses afferent to the *TN.*

Although a greater number of branches permits the detection of spike sequences of higher cardinality, as a basic example it will be analyzed the *Simple direct STSD*, that consists of two branches connected to the external inputs (EI_1 , EI_2) and afferent to a *TN*. This structure is able to operate with spike sequences of cardinality 2. For the sake of simplicity, for this structure unitary amplitudes will be considered for the input spikes for a better understanding of the operation principle.

The topology of this particular architecture is shown in Fig. 7

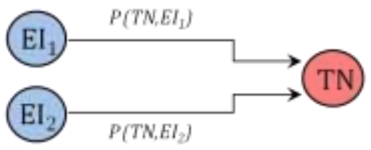


Figure 7: Topology of the Simple direct STSD structure. The detection is positive when the neuron TN produces a spike; otherwise the detection is negative.

In Fig. 7, $P(TN, EI_1)$ represents the product between pre- and post-synaptic weights concerning the synapse EI_1-*TN*, and $P(TN, EI_2)$ represents the same kind of product concerning the synapse EI_2-*TN*. With a proper choice of the parameters, the system is able to work with *WMAL* = 2. This means that *TN* could become active in presence of two input pulses from EI_1 ed EI_2. For example, if $P(TN, EI_1) = P(TN, EI_2) = 0.8$, and the spikes from EI_1 and EI_2 arrive in a very close succession, *TN* becomes active with state equals to:

$$S\,(TN)\;\approx\;1.6\,(>\,S_{th}) \tag{15}$$

such that an output spike is produced, after a latency interval t_f defined by the firing equation (1).

In this case, the output spike generated by *TN* reveals that the structure has received in its input a sequence composed of two spikes close enough. On the other hand, if *the arrival times of* EI_1 and EI_2 (respectively t_1 and t_2) are far, only the value $S\,(TN) = 0.8$ is present at t_1. In facts, for the *subthreshold decay* phenomenon, the inner state of *TN* at time t_2 falls to

$$S\,(TN)\;=\;0.8\;-\;(t_2\,-\,t_1\,)\,Ld \tag{16}$$

and the new incoming spike from EI_2 is not able to make *TN* active; in this case, the detection is negative. Whereas, the detection can be positive if the difference (t_2 - t_1) is smaller.

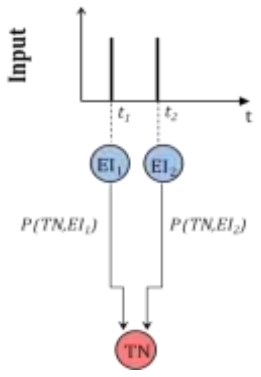


Figure 8: A two-input direct *STSD* (i.e. Simple direct *STSD*); external inputs are affected sequentially by the spikes of the proposed sequence.

Temporal diagram (Fig. 9, *a* and *b*) and activation domain (fig. 10) of the Simple direct *STSD* are plotted below, considering $P_r = 1$, and $P_w(TN, EI_1) = P_w(TN, EI_2) = 0.8$.

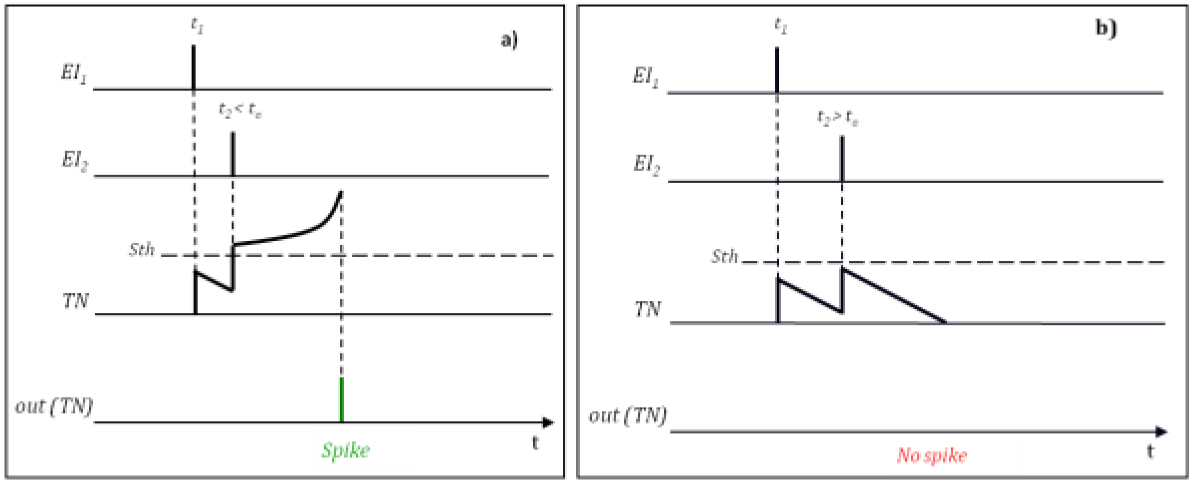


Figure 9: Temporal diagram of the Simple direct *STSD*. As depicted in a), if the second spike arrives in the temporal range between the first spike and the value t_a, the inner state of *TN* exceeds the threshold, then an output spike is generated. If the time of the second spike exceeds t_a, as depicted in Fig. 9b, the detection becomes negative due to the subthreshold decay phenomenon.

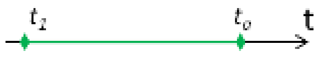


Figure 10: *Activation domain* of the Simple direct STSD: for sequences characterized by interspike intervals comprised between 0 and a certain value t_a, the detection is positive; otherwise the detection is negative.

It is possible to tune the structure with the aim of detecting a particular *ISI* range. The following two relations represent a possibility to properly set the structure parameters to make it sensitive to a particular interval (t_2 - t_1), modifying the values *P* (*TN, EI*$_1$) and *P* (*TN, EI*$_2$) .

The first condition allow the TN to exceed the threshold

$$P_r P_t - (t_2 - t_1)Ld > 1 + d \tag{17}$$

and the second condition imposes a *WMAL* = 2

$$\frac{1}{2}\,\frac{1+d}{P_r} < P_t < \frac{1+d}{P_r} \quad . \tag{18}$$

Of course it is possible to modify the structure in order to recognize sequences of cardinality greater than 2. For this end, in the following sections some considerations will be presented.

4.3 Delayed STSD

This class of structures is characterized by the presence of one or more delay neurons (DNs), i.e. simple excitatory neurons in the middle of a branch, able to create a transmission delay on the connection they are inserted, and capable of obtain useful effects for the computation.

The operation is based on the fact that the pulses coming from the two branches are able to generate a spike in TN only if they are properly distant in time; in this scenario, the delay neuron provides to slow down the first spike so that it can converge to the TN temporally closer to the second input.

The properties of these structures strictly depend on the value of Pwt (i.e. the common post-synaptic weight of the synapses afferent to the TN) and on the values of Pwd (i.e. the post-synaptic weights of the synapses afferent to the DNs), in addition to the global parameters.

4.3.1 Simple delayed STSD

The previous discussion on the *Simple direct STSD*, oriented to spike sequences of cardinality 2, can be easily extended to define structures that support more complex activation domains. If the topology is modified as in fig. 11, that is, characterized by the presence of one *DN*, the activation domain may be shifted of a time equal to *dt* (i.e. the latency time provided by *DN*) with respect to the arrival time of the first spike. In this section, the upper branch will be indicated as the *delay branch*.

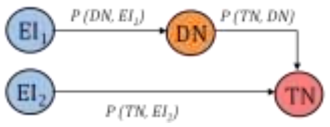


Figure 11: Topology of the Simple delayed STSD.

The value of *dt* can be modified by varying the values of the weights. Of course, this neuron must have a *WMAL = 1*.

Temporal diagram and activation domain of the Simple delayed *STSD* are reported in Fig. 12 and Fig. 13.

By analyzing Fig. 11, it is easy to deduce the requirements for the correct operation of the structure: the activation of *DN* requires *WMAL* = 1, whereas *WMAL* = 2 is needed for the activation of *TN*.

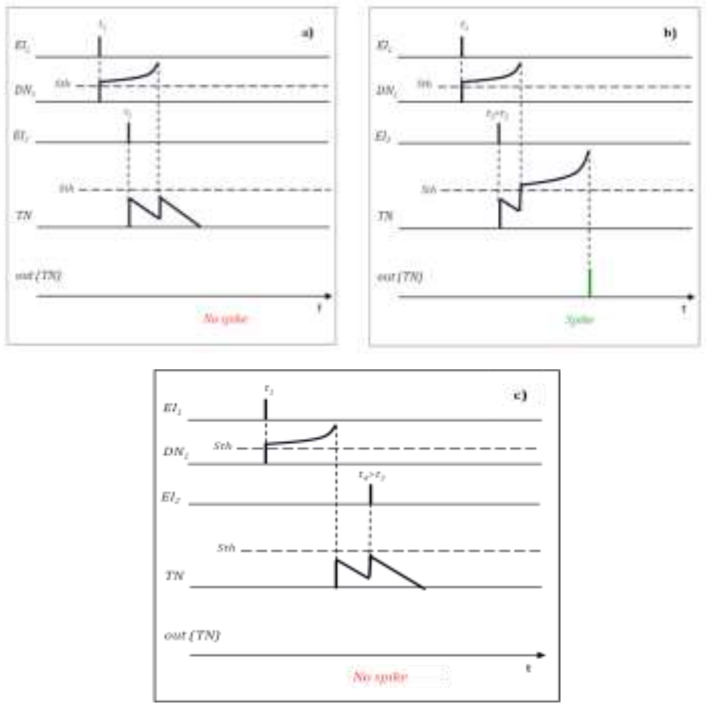


Figure 12: Temporal diagram of the Simple delayed *STSD*. The structure is able to exhibit an output spike only for a certain range of *ISI*, characterizing the *cardinality-2* sequence.

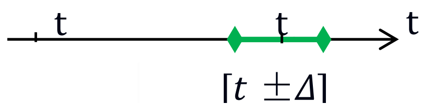


Figure 13: Activation domain of the Simple delayed STSD.

In addition, with the aim of guaranteeing the activation of the neurons of the input layer, the following relations have to be satisfied

$$EI_1 \,\&\, EI_2 \; > \; 1 + d \; . \tag{19}$$

For DN we have

$$P(DN, EI_1) > \; 1 + d \tag{20}$$

namely

$$P_r \; P_w(DN, EI_1) > \; 1 + d \tag{21}$$

that is to say

$$P_w(DN, EI_1) > \frac{1+d}{P_r} \; . \tag{22}$$

On the other hand, to guarantee the output spike, assuming the favorable simultaneity condition on the target, the following relation have to be satisfied

$$P_r \; [P_w(TN, DN) + \; P_w(TN, EI_2)] \; > \; 1 + d \tag{23}$$

then

$$P_w(TN, DN) + \; P_w(TN, EI_2) > \frac{1+\,d}{P_r} \; . \tag{24}$$

Indicating with *Δt_{in}* the time difference between input pulses and *Δt_{out}* the time difference between the pulses afferent to the target, it is possible to merge the previous conditions, as follows:

$$P_r[P_w(TN, EI_2) + P_w(TN, DN)] - Ld \left| \frac{1}{(Iext_2 - 1)} - \frac{1}{Iext_1 - 1} - \frac{1}{[P_r \, P_w(DN, EI_1) - 1]} + \Delta t_{in} \right| > (1 + d) \; . \tag{25}$$

This inequality permits to identify relations that allow the design of such structures for specific purposes.

However, this form is incompatible with an univocal design of the structure because an excessive number of degrees of freedom is present. In order to achieve the simultaneity condition of the two contributes on *TN*, the following equality must be verified:

$$t_f(EI_1) \; + \; t_f(DN) \;\; = \;\; \Delta t_{in} \; + \; \Delta t_{out} + \; t_f(EI_2) \tag{26}$$

with $\Delta t_{out} = 0$. Then

$$t_f(EI_1) \; + t_f(EI_2) \;\; = \;\; \Delta t_{in} \; + \; t_f(EI_2) \;\; . \tag{27}$$

It can be noted that for spike sequences in which pulses are far away, such a structure would be useless. In that case is necessary to use more than one *DN* in cascade, in order to satisfy the following relation:

$$t_f(EI_1) + t_f(DN_1) + t_f(DN_2) + \cdots + t_f(DN_n) = \Delta t_{in} + t_f(EI_2) \, . \tag{28}$$

In addition, in order to recognize sequences of greater cardinality, it is necessary to increase the number of delay branches, such that the contributions can arrive simultaneously to the target neuron, which must be of type *WMAL* n.

In relation to the Simple delayed *STSD*, let us analyze the tuning procedure.

Indicating *Tol* the permissible error on the arrival time of the second pulse still capable to let the *TN* spike, and *MaxTol* the maximum value of *Tol* that can be used, the tuning of the structure can begin with the computation of the maximum tolerance (*MaxTol*) supported by the *Ld* considered; under proper considerations, it can be written

$$Ld \; = \frac{(2P_r\, P_T - 1 - d)}{Tol} \;. \tag{29}$$

Placing P_t to the upper end, we obtain

$$Ld = \frac{\left\{2P_r \frac{1+d}{P_r} - 1 - d\right\}}{MaxTol} \tag{30}$$

where

$$MaxTol \; = \frac{[2\,(1+d) - 1 - d]}{Ld} = \frac{(1+d)}{Ld} \;. \tag{31}$$

At this point, the requirements necessary for the operation of the structure imposes the relations (19) and (20). With the aim of confining the degrees of freedom of the structure it is possible to proceed as follows. The *WMAL* of *TN* can be subdivided evenly with respect to the number of afferences (i.e., make equal the post-weight of the neurons afferent to *TN*)

$$P_r\, P_w(TN, EI_2) \; = \; P_r\, P_w(TN, DN) \; = \; P_r\, P_t \;. \tag{32}$$

In order to assure the output spike, from (4) we obtain

$$2P_r\, P_t \;\; > \;\; (1+d) \tag{33}$$

whereas, in order to guarantee that the target does not work in *WMAL* = 1, must be

$$P_r\, P_t \;\; < \;\; (1+d) \;. \tag{34}$$

Merging the last two conditions, for a two-input Simple delayed *STSD* we obtain

$$1/2 \;\; [(1+d)/P_r] < P_t < [(1+d)/P_r] \tag{35}$$

In this way, imposing the simultaneity condition on *TN*, an output spike will be certainly produced and the target will work at a *WMAL = 2*.

In reference to the interpulse interval between the two input spikes, the more P_t tends to $[(1+d)/P_r]$, the more the structure is tolerant to temporal errors of the second input pulse; conversely, the more P_t tends to $[(1+d)/2P_r]$, the less the structure is tolerant to temporal errors of the second input pulse.

5 STSD-based temporal coding classifiers

Using the elementary structures presented in the previous sections, some pattern recognition problems can be easily afforded.

In an-dimensional space, a single object can be represented as a set of n-reference features. Each axis is referred to a time interval that permits to quantify a reference feature, so that objects belonging to different classes will be mapped in a n-dimensional "timing space" [17]. An operation scheme using Simple delayed STSD is depicted in Fig. 14.

6 Conclusion

In this work have been presented some bio-inspired temporal-decoding network topologies based on a bio-plausible neuron model, for the accurate recognition of spike patterns. These structures are able to detect specific input spike sequences in a continuous time domain. The connection of these neural structures at a higher level, make possible to process the information operating a distributed and parallel input data processing.

Future developments will include the application of the spike-timing dependent plasticity (*STDP*) algorithm with the purpose of auto-setting tuning parameters as the postsynaptic weights.

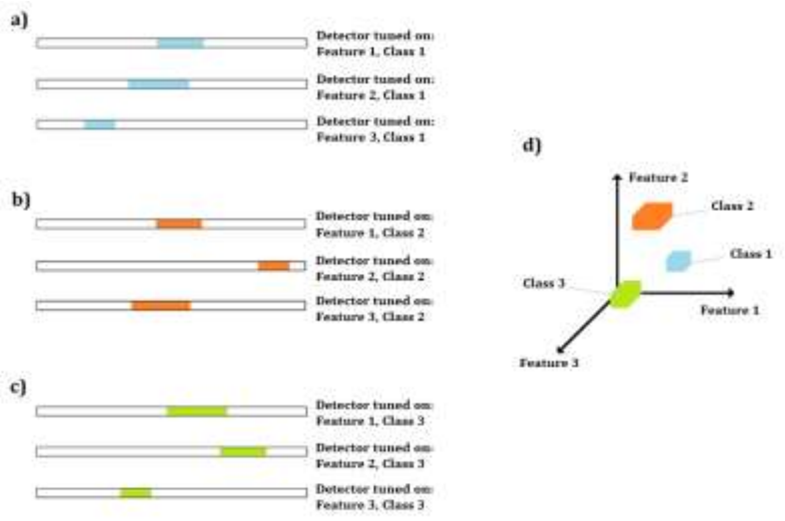

a)
Detector tuned on: Feature 1, Class 1
Detector tuned on: Feature 2, Class 1
Detector tuned on: Feature 3, Class 1
b)
Detector tuned on: Feature 1, Class 2
Detector tuned on: Feature 2, Class 2
Detector tuned on: Feature 3, Class 2
c)
Detector tuned on: Feature 1, Class 3
Detector tuned on: Feature 2, Class 3
Detector tuned on: Feature 3, Class 3
d)
Feature 2
Class 2
Class 1
Class 3
Feature 1
Feature 3

Figure 14. Scheme of a STSD-based temporal coding classifier. Case of three features, each one characterized by means of three classes: a) class 1 recognizer set; b) class 2 recognizer set; c) class 3 recognizer set; d) single class domains, in the three dimensional feature space [17].

REFERENCES

(1) Larson E, Perrone BP, Sen K, Billimoria CP, *A robust and biologically plausible spike pattern recognition network*. The Journal of Neuroscience, 2010. 30(46): p.15566 –15572.

(2) Paninski L , Pillow J , Lewi J, *Statistical models for neural encoding, decoding, and optimal stimulus design*. Progress in Brain Research 2007;165:493-507.

(3) Donoghue J, *Connecting cortex to machines: recent advances in brain interfaces*. Nature Neuroscience 5, pp. 1085–1088.

(4) Gütig, R, Sompolinsky H, *The tempotron: a neuron that learns spike timing-based decisons*. Neuroscience (Nature), 2006.

(5) Panzeri S, Ince RAA, Diamond ME, Kayser C, *Reading spike timing without a clock: intrinsic decoding of spike trains*. Phylosophical transactions B, , The royal society publishing, 2014.

(6) Dhoble K, Nuntalid N, Indiveri G and Kasabov N. *Online Spatio-Temporal Pattern Recognition with Evolving Spiking Neural Networks utilising Address Event Representation, Rank Order, and Temporal Spike Learning*. Neural Networks (IJCNN), International Joint Conference on. IEEE, 2012.

(7) Gautrais J, Thorpe S. *Rate coding versus temporal order coding: a theoretical approach*. Biosystems, Vol 48, Nov 1998, Pages 57–65.

(8) Rieke F, Warland D, van Steveninck RdR. *Spikes: Exploring the neural code*. MIT Press Cambridge, 1999.

(9) Morrison A, Diesmann M, Gerstner W, *Phenomenological models of synaptic plasticity based on spike timing*. Biological Cybernetics, vol. 98, no. 6, pp. 459–478, 2008.

(10)Brader J, Senn W, Fusi S, *Learning real-world stimuli in a neural network with spike-driven synaptic dynamics*. Neural computation, vol. 19, no. 11, pp. 2881–2912, 2007.

(11)Salerno M, Susi G, Cristini A, *Accurate latency characterization for very large asynchronous spiking neural networks*. 4th Int. Conf. on Bioinformatics Models, Methods and Algorithms, 2011.

(12)Cardarilli GC, Cristini A, Di Nunzio L, Re M, Salerno M, Susi G, *Spiking Neural Networks based on LIF with Latency: Simulation and Synchronization Effects*. 2013 IEEE Asilomar Conference on Signals, Systems, and Computers, 3-6 Nov 2013, Pacific Grove, CA, USA, pp. 1838-1842.

(13)Cristini A, Salerno M, Susi G, *A Continuous-time Spiking Neural Network Paradigm*. Advances in Neural Networks: Computational and Theoretical Issues - Smart Innovation, Systems and Technologies, vol. 37, pp. 49-60. Springer, 2015.

(14)Sutskever I, Vinyals O, VL Quoc, *Sequence to Sequence Learning with Neural Networks*. NIPS conference, 2014.

(15)Panchev C, Wermter S*, Temporal sequence detection with spiking neurons: towards recognizing robot language instructions*. Connection science. Taylor Francis, 2006.

(16)Gansel KS, Singer W, *Detecting multineuronal temporal patterns in parallel spike trains*. Frontiers in Neuroinformatics (2012); 6: 18.

(17)*Salerno M, Susi G, Cristini A, Sanfelice Y, D'Annessa A, Spiking neural networks as continuous-time dynamical systems: fundamentals, elementary structures and simple applications*. ACEEE Int. J. on Information Technology, Vol. 3, No. 1, Mar 2013, pp. 80-89.

(18)Gerstner W and Kistler WM, *Spiking Neuron Models - Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

(19)Ventura, V, *Spike Train Decoding without Spike Sorting*. Neural Computation , Vol 20,4. MIT press, 2008.

(20)Gerstner W, Kistler WM, Naud R Paninski L, *Neuronal Dynamics, From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

Society for Science and Education
United Kingdom
TMLAI Transactions on Machine Learning and Artificial Intelligence
Volume 3 Issue 4
ISSN 2054-7390

Phishing Websites Detection Using Data Mining Classification Model

[1]Riad Jabri and [2]Boran Ibrahim
Computer Science Department, University of Jordan, Amman, Jordan
jabri@ju.eu.jo; boranabed@yahoo.com

ABSTRACT

Phishing is a significant security threat to the Internet; it is an electronic online identity theft in which the attackers use spoofing techniques like fake websites that mimic legal websites to trick users into revealing their private information. Many of successful phishing attacks do exist and subsequently a considerable number of anti-phishing methods have been proposed. However, they vary in terms of their accuracy and error rate. This paper proposes an algorithm for phishing websites detection using data mining classification model. It is implemented and experimented using a dataset composed of 20 different webpage features and 1,000 instances. The experimental results showed that the proposed algorithm outperforms the original one in terms of the number of classification rules, accuracy (87%) and less error rate (0.1 %).

Keywords: Phishing detection; PRISM; Machine learning; Classification; Data mining.

1 Introduction

Recently, online services are widely deployed. For example, online banking over the web has become essential for customers as well as for banks. For more than a decade, the naive users have been targeted by phishing attacks to acquire their sensitive information such as username, passwords, and pass code through the fraudulent websites which is under the control of the attackers. Phishing attacks are increasing rapidly. According to the Anti-Phishing Working Group (APWG) [1], the phishing problem has grown significantly over the last years that affect economics, and the financial losses from phishing attacks have risen considerably. Although the most phishing targets are financial institutions like banks, the APWG reports that an increasing number of attacks against other organizations like government agencies and automotive associations [1]. The increasing number of scams (phishing) sites demands new methods to classify the websites with a high degree of confidence. Hence, in this paper, we will use data mining techniques and exploit the websites features to establish a large labeled training data, and then yield a classifier integrating these features in such a way that new coming websites could be classified correctly. We propose an algorithm for phishing websites detection using data mining classification model. The proposed algorithm is an enhanced version of a well-known data mining algorithm, called PRISM [2]. PRISM algorithm has been selected to learn the relationships between the selected phishing features. We have chosen this algorithm since the learnt classifiers are easily understood and used by a human. In addition, the learnt can easily be expanded to include features of newly emerging phishing web sites.

DOI: 10.14738/tmlai.34.1381
Publication Date: 3th September, 2015
URL: http://dx.doi.org/10.14738/tmlai.34.1381

This paper is organized as follows: in section 2 related works of Phishing Detection are presented, section 3 describes the classification model and the proposed algorithm, section 4 shows experimental evaluation. A discussion and conclusion are given in section 5 section 6 respectively.

2 Related Works

Although phishing is not new and well-known by Internet users, many people are still tricked into providing their confidential information. To counter the phishing threat, a number of anti-phishing solutions have been proposed. Representative ones are as follows:

- A class of anti-phishing approaches aims to solve the phishing problem at the email level; the main idea is that when phishing emails reach its victims, they could fall for the scam. This line of research is closely related to anti-spam research. One reason for the abundance of spam and phishing emails is that the Simple Mail Transport Protocol (SMTP) does not contain any authentication mechanisms. The sender information in an email message can easily be spoofed [3]. In this context, in [4] a novel approach is proposed to detect phishing attacks by implementing a prototype web browser to processes the arriving email for phishing attacks, they integrate this method with the approach of using link based features. Such as a hyperlink to get personal details about users, such as usernames, passwords, account numbers, etc. Thus, this approach considers features, which have numerical values but with different ranges, these features will be extracted from the email body and according to a certain values, the user gets notified of phishing attacks to avoid opening the suspicious websites.
- To capture the patterns in phishing URLs, Afroz and Greenstadt [5] proposed a real time approach for web phishing detection, called PhishZoo. This approach uses profiles of trusted website's appearances and content, to detect targeted phishing attacks. They make use of computer vision techniques such as matching images, scene analysis (segmenting images into objects) and the URLs and (HTML) contents of a website to identify imitations.
- There exists a number of anti-phishing toolbars based on different techniques, many of which exploit blacklists to achieve close-to-zero false positive rate [6]. The most popular and widely-deployed anti-phishing techniques are based on the use of blacklists. These blacklists store a set of phishing domains that the browser blocks their visit [3]. On the other hand, other approaches exploit whitelists [5]. These approaches seek to detect known good sites. Some whitelisting approaches use server side validation to add additional authentication metrics to client browsers as a proof of its benign nature, for examples, dynamic security skins [7]; trust bar [8] and SRD ("Synchronized Random Dynamic Boundaries") [9].
- In an attempt to exploit visual similarity, Wenyin [10] proposed an approach to detect the phishing web pages based on visual similarity, if the visual similarity of these pages is higher than a predetermined threshold, this website is recorded as a suspicious and the owner is alerted.
- A content-based approach, called CANTINA, is proposed in [11] to detect phishing websites. CANTINA examines the content of a web page to determine whether it is legitimate based on the term frequency/inverse document frequency.

- In [12] authors proposed a server-side detection model; an automatic, and proactive identity theft detection model in online games utilizing rich features. The proposed model classifies a hacker's login from a genuine user's login using well-balanced features from connection information, user behavior, and economic variables.
- An interesting solution has been proposed in [13] to detect e-banking phishing websites using an artificial intelligent technique, authors propose a model based on using association and classification data mining algorithms and tools, which were used to classify the fishing websites and the relationship that correlate them with each other, six classification algorithm were implemented to extract the phishing training datasets criteria to classify their legitimacy.

Although there are many solutions available and act as a model for automatic phishing detection, the promising ones are based on machine learning and data mining techniques. Recently, an associative classification expert system was proposed in [14]. It has been shown that the rule induction and decision tree methods outperformed other ones [12, 14]. Hence, this paper constitutes an improvement of such techniques.

3 Classification Model

The Using machine learning techniques, a data mining classification model based on modified version of PRISM algorithm is proposed to categorize websites as phishing versus non-phishing. Such categorization is achieved by applying induction rules on selected websites features. The construction of such a model proceeds as follows:

- A website is represented as a set of features and their respective values. For example, {<subjreply {0, 1}>,…, <urlnoLinks{0,1,..}}
- A dataset is then constructed as a set of tuples of the values of the selected features of phishing and non-phishing websites. Hence, the dataset describes two classes (phishing versus non-phishing) and subsequently each tuple is assumed to belong to a predefined class, as determined by the class label. For example, the rows of Table 1 represent such tuples, where the first row represents site 1 that is classified as a phishing one. On the other hand, Site n is classified as non-phishing. The complete dataset is given in Table 2.

Table 1: Dataset example

	subjreply …………	attachedfile	class
site			
1	0	0	1
n	1	1	0

- Using distribution probabilities, the dataset is then divided into training and testing one. A classification model is then constructed as a set of induction rules. For example, {If invisiblelinks=1 and If subjforward=1 Then yes,…, If urlnoLinks=6 Then yes}. Such a set is obtained by applying the proposed algorithm on the training dataset. It is then tested and evaluated using the test set to achieve a high accuracy rate as well as less error rate, where the accuracy rate is percentage of test set samples that are correctly classified by the model, while the error rate is the opposite.

3.1 The Proposed Algorithm

PRISM is attribute-value-oriented; it measures the attribute-value pair in determination of the classification and selects the attribute that contributes the maximum information gain. If this attribute is perfect (the attribute accuracy is equal 1), the corresponding rule will be generated and added to the rule set to represent the classification model for the prediction purposes. PRISM generates one rule at a time, in a particular iteration if more than one attribute is perfect this algorithm chooses the attribute with the highest coverage to generate a rule, otherwise a random attribute will be chosen to generate a specific rule. In contrast to PRISM algorithm, the proposed one derives all the perfect rules at the same time instead of learning one rule at a time. As a result, the number of derived rules and their coverage rate were increased. Hence, the accuracy and the expressiveness of the proposed model and subsequently the prediction process were improved. The pseudo code of the proposed algorithm is given below as Algorithm 1.

Algorithm1

```
Input: labeled training dataset D (%)
Output:  rule set R that covers all instances in D
Method:
Initialize R as the empty set
for each class C {
        while D is nonempty {
              Construct all perfect rules r that correctly classifies some
           instances in D that belong to class C and do not incorrectly
           classify any non-C instances;
              Add rules r to rule set R
              Remove from D all instances which are correctly classified by
           all r
        }
return R
```

4 Experimental Evaluation

In this section, we present the experiments designed to build and evaluate the proposed model. This includes a description of the constructed dataset, metrics to evaluate the effectiveness of our approach. The experiments were conducted over 1000 different websites (70% are phishing ones and 30% are non-phishing) were used in the conducted experiments.

4.1 Dataset

For the purpose of this research, a large number of phishing pages with their respective features were explored. As a result, 20 feature- value pairs were selected based on the frequency analysis as suggested in [15] and as shown by the following set

{The set of features and it their possible values:

Attribute subjreply {0, 1},

Attribute subjforward {0, 1},

Attribute subjnoWords {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16},

Attribute subjverify {0, 1},

Attribute subjbank {0, 1},

Attribute bodyhtml {0, 1},

Attribute bodyFunctionWords {0, 1},

Attribute bodysuspension {0, 1},

Attribute bodyconfirm/verifyYourAccount {0, 1},

Attribute urlnoIpAddresses {0, 1},

Attribute urlatSymbol {0, 1},

Attributeur lnoLinks {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,..},

Attribute urlnoDomains {0,1,2,3,4,5,6,7,9,11,12,14,15,17,29,73},

Attribute urlmaxNoPeriods {0,1,2,3,4,5,6,7,8,10,11,12,13,16},

Attribute urllinkText {0, 1, 2},

Attribute OnClickEvents {0, 1, 10},

Attribute Invisiblelinks {0, 1},

Attribute Unmatchingurl {0, 1},

Attribute Longurladdresses {0, 1},

Attribute attachedfile {0, 1}**}**

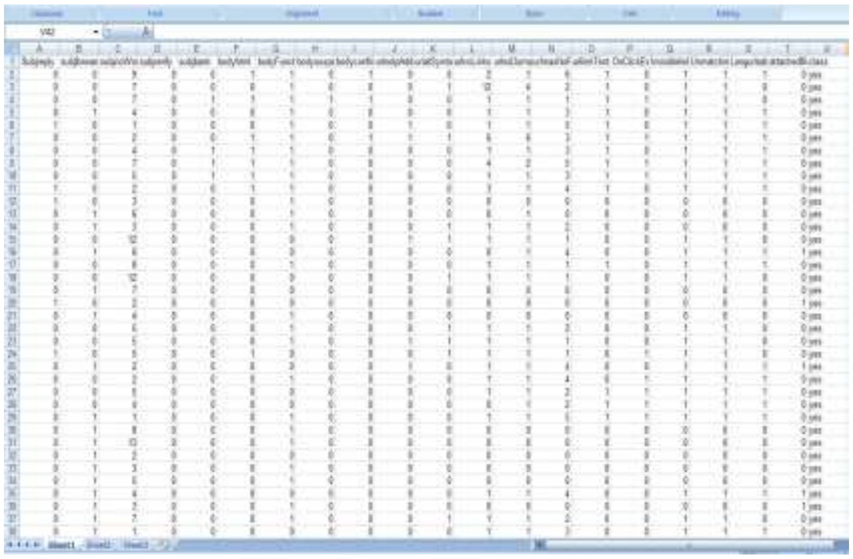
The dataset is then constructed by extracting feature – values pairs of phishing cases from one source, and non-phishing web pages from another source. We chose PhishTank [15] as a source of phishing websites. The information from this site is freely available and the amount of reported phishing sites is very large (approximately five hundred new phishing reports every day). The attributes of the non-phishing web sites were collected by a web data extractor. Such extractor is a user defined software that is plugged with a browser to automatically extract features from a user connection [16]. Thus, the constructed dataset contains a total of 700 phishing websites and 300 legitimate ones, as shown in table 2.

Preparatory experiments were conducted on this dataset using the well-known data mining software WEKA to show the classification effectiveness of the machine learning algorithms and to justify the proposed algorithm. The performance of three algorithms (JRip, J48, Naïve Bayes (NB)) has been compared to the one of PRISM, as shown Table 3. Such comparison has been performed using 70% of the dataset for training and the remainder for testing. In addition, the following evaluation metrics have been applied:

- Accuracy and error rate
- Number of derived rules

It is clear that the J48 algorithm is the best one, because it has the highest accuracy in the websites dataset, while NB algorithm is a weak algorithm since it has the lowest accuracy with the higher error rate, and it doesn’t produce rules at all. PRISM algorithm has a minimum accuracy with some unclassified instances in both datasets.

Table 2: The constructed dataset


[illegible]

Hence, we set an objective to improve this algorithm PRISM algorithm in terms of discovered knowledge, accuracy, error rate and other criteria which appear useful for phishing detection. PRISM is “weak” algorithm; the aim is to change it into an accurate “strong” typical algorithm for our detecting purposes. We selected the PRISM classifier for the following reasons. First, we believe that a PRISM provides intuitive insight into which features are important in classifying a dataset. Second, the PRISM algorithm is known to work well for a wide range of classification problems.

Table 3: Results of WEKAs' algorithms

	No. of Instances	WEKA Algorithms	No. of Rules	Accuracy Rate %	Error Rate %	Unclassified Rate %
Websites Data	1000	JRip	7	91	9	-
		J48	26	94	5	-
		NB	-	88	11	-
		PRISM	189	86	5	8

4.2 Experimental results

Experiments were conducted to evaluate the performance of PRISM and the proposed algorithm. Figure 1 and Figure 2 show samples of their run using the collected dataset.

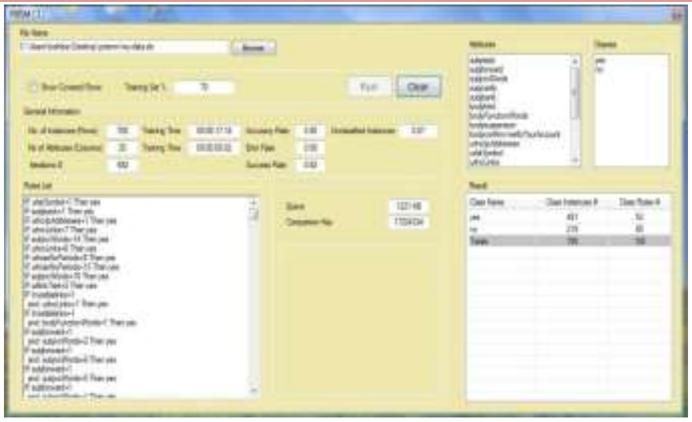


Figure 1: A sample run of PRISM

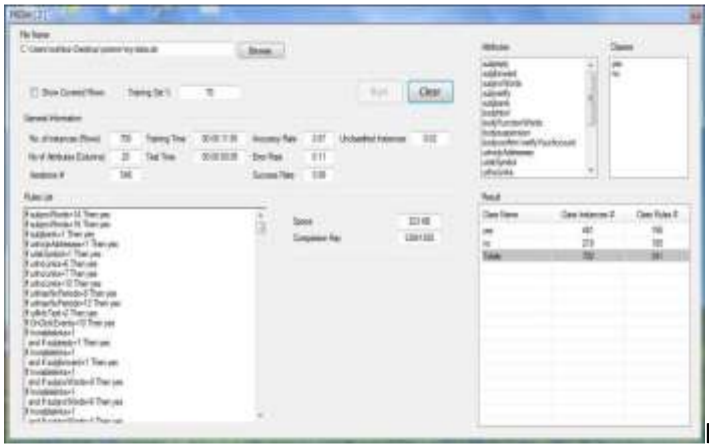


Figure 2: A sample run of the proposed algorithm

Several runs of PRISM and the proposed algorithm have performed using different ratios between training, and testing datasets show the results of these runs as follows

- Table 4 shows the achieved accuracy and error rate by both algorithms
- Table 5 shows the phishing and legitimate rules generated by both algorithms.
- Table 6 shows the number of iterations performed during the run of both algorithms

Table 4: Accuracy and error rates for the considered models

	PRISM				Modified PRISM			
Training Data % %	40	60	70	80	40	60	70	80
No. of Phishing Instances	267	407	481	555	267	407	481	555
No. of Legitimate Instances	133	193	219	245	133	193	219	245
Accuracy Rate %	84	84	85	88	84	85	87	86
Error Rate	12	9	8	6	15	13	11	12

Table 5: Total number of rules genrated by the considered models

	PRISM				Modified PRISM			
Training data %	40	60	70	80	40	60	70	80
No of phishing instances	267	407	481	555	267	407	481	555
No of legitimate instances	133	193	219	245	133	193	219	245
No. of Rules	67	117	138	150	121	270	341	373
no. of phishing rules	25	49	53	51	45	123	156	161
no. of legitimate rules	42	68	85	99	76	147	185	212

The obtained results demonstrate that the constructed classifiers have 341 rules, with accuracy of (87%). Thus, the proposed algorithm achieves higher accuracy and generates more rules. Furthermore, it is more efficient, as demonstrated by the number of iterations performed during its run.

Table 6: Total number of iterations performed by the considered models

	PRISM				Modified PRISM			
Training data Percentage %	40	60	70	80	40	60	70	80
No of instances	400	600	700	800	400	600	700	800
No of phishing instances	133	407	481	555	267	407	481	555
No of legitimate instances	267	193	219	245	133	193	219	245
No of Iterations	513	652	692	713	444	515	546	563

5 Discussion

Considering the results achieved by the proposed algorithm, we can conclude that this approach is quite successful in protecting users, especially when predicting the phishing websites. The proposed algorithm has correctly recognized almost 87% of the malicious websites. It appears that this algorithm outperforms the original PRISM and has a comparable performance to the one of several algorithms (CBA, PART, C4.5, MCAC, JRip, and MCAR) as published in recent research [14], where the accuracy of these algorithms ranges from 92% to 94.5% , while and the number of rules ranges from 10 to 180. Hence, the results the proposed algorithm is encouraging. Although the results for the larger datasets are rich, it takes more time to compute. It is worth mentioning that the proposed phishing detection system represents a good indicator for the phishing websites since this model produced a large number of significant rules. In addition, it is flexible and capable of capturing new emerging features of the phishing web sites.

6 Conclusion

In this paper, we have proposed a new phishing websites detection model based on PRISM algorithm, the prediction of phishing websites is essential, and this can be done using data mining classification algorithms, our classification system automatically classify phishing pages. We have tested the phishing detection model in terms of accuracy and error rate. The experimental results confirm the increasing number of rules of the proposed method for phishing detection and show that we maintain a false positive and false negative rate about (0.1%) which is reasonably low. The experiments have demonstrated the feasibility, and effectiveness of using rule induction classification techniques in real applications involving large databases. It has better performance as compared to other traditional classification algorithms with accuracy rate of (87%). Due to the variations in the existing approaches, a generalized and formalized one constitutes a future work.

REFERENCES

[1]. APWG. *Phishing Activity Trends*: Technical report, Anti Phishing Working Group, [online], http://www.antiphishing.org/reports/apwg_trends, 2013.

[2]. Cendrowska, Jadzia, *PRISM: An algorithm for inducing modular rules*. International Journal of Man-Machine Studies, 1987. 27(4): P. 349-370.

[3]. Medvet, E., Kirda, E., and Kruegel, C., *Visual-Similarity-Based Phishing Detection*. In Proceedings of the 4th international conference on Security and privacy in communication networks, ACM 22, Istanbul, Turkey, 22 – 25, September, 2008.

[4]. Jain, A., and Richariya, V., *Implementing a Web Browser with Phishing Detection Techniques*. World of Computer Science and Information Technology Journal (WCSIT), 2011. 1 (7):p. 289-291.

[5]. Afroz, S. and Greenstadt, R., *PhishZoo: Detecting Phishing Websites By Looking at Them*. In Proceedings of the Semantic Computing (ICSC), Fifth IEEE International Conference , 18-21 September, 2011, 368 – 375.

[6]. Xiang, G. and Hong, J.I., *A Hybrid Phish Detection Approach by Identity Discovery and Keywords Retrieval,* In Proceedings of the 18th International World Wide Web Conference (IW3C2), ACM, Madrid, Spain, 2009, p. 571- 580.

[7]. Kumaraguru, P. Rhee, Y. Acquisti, A. Cranor, L. F. Hong, J. and Nunge, E., *Protecting people from phishing: The design and evaluation of an embedded training email system*. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07), ACM, New York, NY, USA, 2007, p. 905-914.

[8]. Herzberg A. and Gbara, *A Security and Identification Indicators for Browsers against Spoofing and Phishing Attacks*, Journal of ACM Transactions on Internet Technology (TOIT), 8(4): p. 1-36.

[9]. Ye Z. and Smith, S. *Trusted paths for browsers*, In Proceedings of the 11th Usenix Security Symposium, ACM NY, USA, 2005, p. 263-279.

[10]. Wenyin, L. Huang, G. Xiaoyue, L. Min, Z. and Deng, X., *Detection of Phishing Webpages based on Visual Similarity*, In Proceedings WWW '05 Special interest tracks and posters of the 14th international conference on World Wide Web, ACM, May, 2005, p. 1060-1061.

[11]. Zhang, Y. Hong, J. and Cranor, L., *CANTINA: A Content-Based Approach to Detecting Phishing Web Sites*. In Proceedings of the 16th international conference on World Wide Web, , **ACM**, Banff, Alberta, Canada, 8-12, 2007, p. 639-648.

[12]. Woo, J. Choi, H.J. and Kim, H.K., *An automatic and proactive identity theft detection model in MMORPGs*, .Applied Mathematics and Information Sciences, 2012, 6(1S) : p. 291S-302S.

[13]. Aburrous, M. Hossain, M. A. Thabtah, F. and Dahal, K., *Intelligent Detection System for e-banking Phishing websites using Fuzzy Data Mining*, International Conference on CyberWorlds, IEEE Conference Publications **,** 2009**,** 37(12)**:** p. 265-272.

[14]. Abdelhamid,N., Ayash, A., Tabatah, F., *Phishing Detection Based Associative Classification Data Mining*, Expert System with Application, 2014, 41: p., 5948-5959.

[15]. PhishTank, *Out of the Net, into the Tank*, http://www.phishtank.com/developer_info.php, 2012

[16]. Aburrous, M. Hossain, M. A. Dahal, K. Thabtah, F., *Predicting Phishing Websites using Classification Mining Techniques with Experimental Case Studies*, In Proceedings of the 2010 Seventh International Conference on Information Technology: New Generations (ITNG '10), **IEEE**, Las Vegas, Nevada, USA, April 2010, p. 176-181