Ant Colony Optimization Applied to the Regularization Process of a High Order Neural Network

Ashraf M. Abdelbar
Dept. of Mathematics & Computer Science
Brandon University
Brandon, Manitoba, Canada
Email: AbdelbarA@brandonu.ca

Khalid M. Salama
School of Computing
University of Kent
Canterbury, United Kingdom
Email: kms39@kent.ac.uk

***Abstract*—High order neural networks (HONN's) are neural networks that employ neurons which combine their inputs non-linearly. HONEST (High Order Network with Exponential SynapTic links) is a HONN that employs product units, and inter-neuronal connections with associated adaptable exponential weights. Previous work has found that HONEST benefits from the inclusion in the network's error function of a regularization term that penalizes high-magnitude exponents. In the present work, we use $ACO_{\mathbb{R}}$, a recent Ant Colony Optimization (ACO) algorithm, to optimize the parameters of HONEST's exponent regularization process, using a collection of UCI datasets. We then evaluate HONEST with the evolved parameters on a second non-overlapping collection of UCI datasets against Support Vector Machines (SVM). We find HONEST's test set predictive accuracy to be competitive with SVM, with no statistically significant difference between the two.**

I. Overview

High order neural networks (HONN's) are neural networks that employ neurons which combine their inputs non-linearly [1]–[3]. HONN's have been the subject of considerable research [1]–[5], and have generally been found to have superior generalization, at the expense of being more computationally intensive and sometimes harder to train. HONEST (High Order Network with Exponential SynapTic links) [6]–[14] is a HONN that employs product units, and inter-neuronal connections with associated adaptable exponential weights. An HONEST network has an input layer, a single hidden layer, and an output layer. The output layer has standard weighted-sum neurons, while the hidden layer neurons raise their inputs to an exponent and combine them multiplicatively. In addition, the HONEST network uses identity activation functions for all of its neurons. This design makes the HONEST network both powerful and easy to interpret.

Previous work [10] has found that HONEST benefits from the inclusion in the network's error function of a regularization term that penalizes high-magnitude exponents—which are often an indication of over-fitting the training set. The regularization term was based on an inverted generalized-bell function, with parameter values that were determined in an *ad hoc* manner. Experimental results indicated that the regularization term resulted in a lower frequency of high-magnitude exponents and a corresponding improvement in generalization.

On the other hand, the downside of the regularization process is that it introduces three new parameters to the HONEST netwok: two parameters control the shape of the generalized-bell function, and a third parameter contols the relative emphasis of the regularization term. This is an example of a general problem in computational intelligence (CI): the ever-increasing number of adjustable external parameters in CI models. To alleviate this problem in the present case, we would like to use a systematic method to find a "good" set of parameter values for HONEST's three regularization parameters that can be used as default parameter values that are not specifically optimized for a specific dataset. It is of course true that, like most CI models, better performance can likely be obtained if HONEST's parameters are tuned for a specific dataset or group of datasets. However, tuning HONEST, or any other learning model, for a specific dataset is time-consuming and may not be practical in many situations.

Therefore, our objective in the present study is to identify, using $ACO_{\mathbb{R}}$, a recent Ant Colony Optimization (ACO) algorithm, a "good" set of parameter values that are not specifically tuned to a particular dataset. We use a collection of 5 UCI datasets in the optimization process; the output of the optimization process is an optimized setting of these three parameters. We then fix the value of these three parameters to the optimized setting, and evaluate the network on a second collection of 20 UCI datasets, with no overlap between the two collections. Since the network is evaluated on a different group of datasets than that with which the parameters were optimized, the performance of network on the evaluation datasets can be considered indicative of the generality of the parameter settings. Of course, if the parameters were tuned specifically for each of the 20 evaluation datasets, it is likely that performance can be improved (as with almost any other CI learning model), but the purpose of the present study is to identify a "good" set of default parameter settings that can be used in any future work with the HONSET

network. We compare the performance of HONEST on the 20 evaluation datasets to Support Vector Machines (SVM), widely-considered to be a state-of-the-art supervised learning model.

We begin in Section II with a brief review of the HONEST network and its regularization process. Section III presents in greater detail our proposed approach in using $ACO_{\mathbb{R}}$ to evolve HONEST's regularization parameters. Our experimental methodology and results are presented in Sections IV and V, respectively, and final comments are offered in Section VI.

II. The HONEST Neural Network.

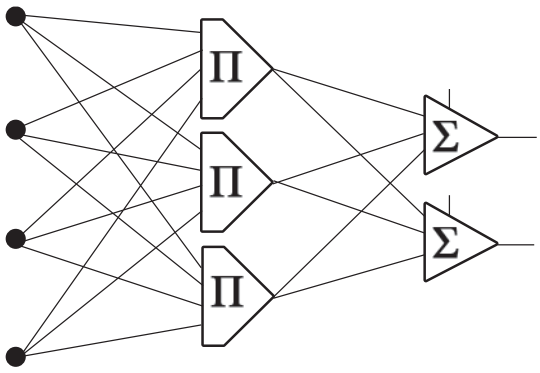

Fig. 1. Architecture of the HONEST network.

HONEST (High Order Network with Exponential Synap-Tic links) [6]–[14] is a high order neural network that uses neurons with adaptable exponents. It can be considered a generalization of the sigma-pi model [1], [15], [16], while sharing some similarities with the ExpoNet [17] and GMDH [18], [19] network models. An HONEST network, as illustrated in Fig. 1, is a three-layer feedforward network made up of two different neuron types. Neurons in the hidden layer are all of high-order, whereas output layer neurons are simple linear units with an identity activation function.

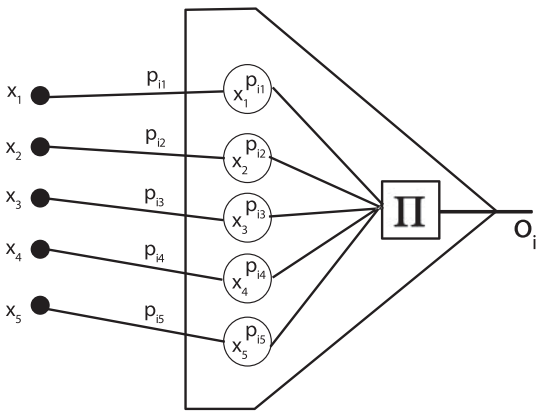

Fig. 2. Hidden Layer Neuron in the HONEST Architecture.

The functionality of a hidden layer neuron i, as illustrated in Fig. 2, is described by

$$h_i = \prod_j x_j^{p_{ij}} \tag{1}$$

where p_{ij} is an exponential power associated with the synapse connecting unit j to a hidden neuron i.

The connections from the input layer to the hidden layer do not have an associated multiplicative weight; instead, they have an associated adaptable exponential power. The output of a hidden layer neuron is the product of its inputs after each one is raised to its associated power, as indicated in Eq. (1).

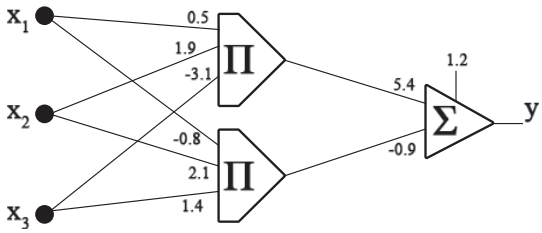

Fig. 3. Example of an HONEST network.

The output layer neurons are all simple linear units. The functionality of an output layer neuron i is described by

$$y_i = \sum_j w_{ij} h_j + \theta_i \tag{2}$$

Each of the outputs, y_i, of an HONEST network can be expressed in terms of the network inputs by an expression of the form

$$y_i = \sum_h w_{ih} \prod_j x_j^{p_{hj}} + \theta_i \tag{3}$$

For example, the equation

$$y = 3.2x_1^{2.5} + x_2^{-1.3} - 0.5x_1^2 x_2^{-3} + 2x_1^4 - 4.3 \tag{4}$$

is represented by the network shown in Fig. 3. The form of Eq. (3) is similar to the form of a polynomial with the number of hidden neurons corresponding to the number of terms in the polynomial. The difference is that, in a polynomial, the exponents are restricted to being non-negative integers. Therefore, the form of Eq. (3) can be seen as a generalization of the form of a polynomial. This means that an HONEST network can always be prescribed to represent any given polynomial mapping.

After a conventional MLP network is trained, interpreting the weights of the network in a meaningful way is very difficult. For a three-layer network, each network output is expressed as a sigmoidal function of a linear sum of sigmoidal functions of linear sums of the network inputs. For non-trivial network sizes, it is practically impossible to reach an intuitive understanding of how the networks inputs are mapped to its outputs.

The structure of an HONEST network is more transparent and easier to interpret; each of the network outputs can be expressed in terms of the network inputs by a polynomial-like

equation. It is thus easier to understand how the network's inputs come to be mapped to the networks outputs. For example, using the form of Eq. (3), it is easier to determine if one particular input is more significant than the others. It may also be easier to detect undesirable situations where the network has over-learned or memorized the training set. Furthermore, it is possible to use external expert knowledge of the problem domain to examine the validity of the network solution. Alternatively, examining the HONEST network solution may serve to enrich ones understanding of the problem domain.

Tsai [12]–[14] has considered variations of HONEST in which sigmoidal activation functions are used, and in which there potentially exist multiple layers of multiplicative units. The drawback of this approach is that it causes HONEST to lose its transparency and interpretability.

It is often desirable for an HONEST network to avoid exponents of very high-magnitude. Such exponents reduce the transparency and interpretability of the network, and make the mapping learned by the network more jittery and more prone to over-fitting. In addition, if a hidden layer neuron has only small-magnitude incoming exponents, then a domain expert may decide to neglect the term in Eq. (3) corresponding to that neuron, thus resulting in a simpler model. Alternatively, exponent regularization could be combined with hidden layer neuron pruning during the training process.

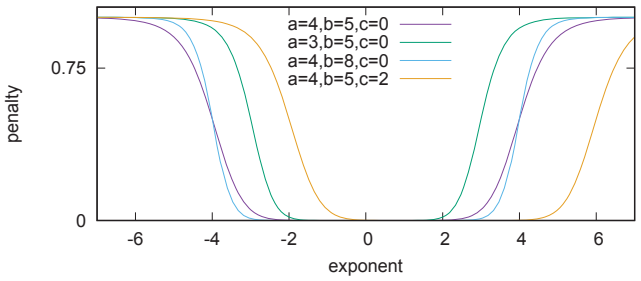

Fig. 4. Illustration of the generalized inverted-bell function, ψ, with different parameter settings.

Thus, it would be beneficial to apply a regularization function that only affects the exponents. In previous work, we used a network error function E consisting of two components:

$$E = E_{MSE} + \alpha E_{reg} \tag{5}$$

where E_{MSE} is the conventional mean-squared-error function, normalized relative to the number of patterns and the number of output neurons:

$$E_{MSE} = \frac{1}{2mP} \sum_{i=1}^{m} (t_i - y_i)^2 \tag{6}$$

where y_i denotes output neuron i, t_i denotes the target output for output neuron i, m denotes the number of output neurons, and P denotes the number of patterns. The regularization component E_{reg} consists of

$$E_{reg} = \frac{1}{nHP} \sum_{j=1}^{n} \sum_{h=1}^{H} \psi(p_{hj}) \tag{7}$$

where n denotes the number of input units, H denotes the number of neurons in the hidden layer, and ψ denotes the generalized inverted-bell function (illustrated in Fig. 4):

$$\psi(p) = \frac{1}{1 + \left(\frac{p-c}{a}\right)^{-2b}} \tag{8}$$

where a, b, and c are parameters that control the shape of the inverted bell function. The a and b parameters control the width of the curve and its slope, and c controls where the curve is centered. For our purposes, we would like the generalized inverted-bell function to be symmetric around zero; therefore, the c parameter is fixed at 0, reducing the ψ function to

$$\psi(p) = \frac{1}{1 + \left(\frac{a^2}{p^2}\right)^{b}} \tag{9}$$

In previous work, we set $a = 4$ and $b = 5$, based on the observation that this setting had the following desirable properties:

- when $p \approx \pm 2$, then $\psi(p) \approx 0$;
- when $p \approx \pm 4$, then $\psi(p) \approx 0.5$;
- when $p \approx \pm 5$, then $\psi(p) \approx 0.9$;

These properties mean that the penalty ψ increases from 0 to 0.5 as the exponent progresses from ± 2 to ± 4, and then continues to increase to 0.9 as the exponent nears ± 5. This means that exponents will be increasingly penalized as they move outside the range $[-2, 2]$. Fig. 4 illustrates the generalized inverted-bell function with these parameter settings ($a = 4$, $b = 5$).

The α coefficient in Eq. (5) is an external parameter that controls the relative emphasis of the regularization term. Because E_{MSE} and E_{reg} are both normalized, it is possible to set α in a manner that is problem-independent. In previous work, α was set to 0.05.

III. Approach

HONEST's regularization function has three adjustable parameters: a, b, and α. Our objective in this paper is to use $ACO_{\mathbb{R}}$, a recent Ant Colony Optimization (ACO) algorithm, to optimize these three parameters. Ant Colony Optimization (ACO) [20] is a general-purpose, biologically-motivated, population-based meta-heuristic, and is based on a number of primitive processing elements, each operating in parallel with little centralized control. The processing elements in ACO are called *ants*, and the collection of processing elements are called a *colony*. In an ACO algorithm, there is usually a central data structure, analogous to pheromone information in biological ant systems, that represents the time-evolving collective knowledge of the group. In each iteration, each ant typically generates a candidate solution, making use of the

central pheromone data structure in some way in its solution construction. After all ants have generated their solutions, a subset of those solutions is then used to update the central data structure in some way. ACO has been applied to a wide variety of domains, and has been applied to supervised learning using a variety of learning models including classification rules [21]–[26], decision trees [27], [28], and various types of Bayesian network classifiers [29], [30].

$\text{ACO}_{\mathbb{R}}$ is a fairly-recent ACO algorithm for continuous optimization [31], [32], and has been applied to the training of neural networks [11], [33], [34], optimizing feedforward neural network topology [35], and optimizing the external parameters of support vector machines [36]. Variations of $\text{ACO}_{\mathbb{R}}$ include a variation called incremental $\text{ACO}_{\mathbb{R}}$ in which the size of the population archive grows incrementally over time [37], and a variation in which the search width ξ gradually decays over time [38].

The central data structure, analogous to pheromone information in natural ants, that is maintained by $\text{ACO}_{\mathbb{R}}$ is an archive A of R previously-generated candidate solutions. Each element s_k in the archive, for $k = 1, 2, \ldots, R$, is a 3-dimensional real-valued vector that represents a configuration of the three parameters (a, b, α). For example, $s_{k,2}$ represents the value of the HONEST regularization parameter b in the k-th solution in the archive. Similarly, $s_{k,1}$ and $s_{k,3}$ represent the value of the parameters a and α, respectively, in the k-th solution in the archive.

$\text{ACO}_{\mathbb{R}}$ is a problem-independent meta-heuristic; to apply it to a specific problem, a problem-dependent quality evaluation function Q must be specified. The quality evaluation function takes as input a candidate solution vector, and returns a non-negative real value that represents the problem-dependent "goodness" of the candidate solution. Our quality evaluation function is described towards the end of this section.

The archive A is always maintained in a state of being sorted by solution quality, so that $Q(s_1) \geq Q(s_2) \geq \ldots \geq Q(s_R)$. Each solution s_k in the archive has an associated weight ω_k that is related to $Q(s_k)$, so that $\omega_1 \geq \omega_2 \geq \ldots \geq \omega_R$.

The $\text{ACO}_{\mathbb{R}}$ algorithm consists of repeated iterations where each iteration consists of two phases: solution construction and pheromone update. In the solution construction phase, each ant probabilistically constructs a solution based on the solution archive A (representing pheromone information). The solution archive A is initialized with R randomly-generated solutions, where the size R is a user-supplied parameter of the $\text{ACO}_{\mathbb{R}}$ algorithm. In our case, the elements of the archive is randomly initialized using a Gaussian distribution centered around the parameter values $(4, 5, 0.05)$ used in previous work.

In the pheromone update phase, the m constructed solutions (where m is the number of ants) are added to A, resulting in the size of A temporarily being $R + m$. The archive A is then sorted by solution quality, and the m worst solutions are discarded, so that the size of A returns to being R.

The heart of the algorithm is the solution construction phase. In this phase, each ant i generates a candidate solution s_i, where s_i is a 3-dimensional vector, and $s_{i,j}$ represents an assignment to the j-th parameter. In constructing its solution s_i, ant i is influenced by one of the R solutions in the archive A. The ant first probabilistically selects one of the R solutions in the archive according to:

$$\Pr(\text{select } s_k) = \frac{\omega_k}{\sum_{r=1}^{R} \omega_r} \tag{10}$$

Thus, the probability of selecting the k-th solution is proportional to its weight ω_k. Recall that the archive A is sorted by quality, so that solution s_k has rank k. The weights ω_k that are used in Eq. (10) are constructed in each iteration as:

$$\omega_k = g(k; 1, qR) \tag{11}$$

where g is the Gaussian function:

$$g(y; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} \tag{12}$$

Thus, Eq. (11) assigns the weight ω_k to be the value of the Gaussian function with argument k, mean 1.0, and standard deviation (qR). The value of q is a user-supplied parameter of the algorithm, where smaller values of q cause the better ranked solutions to have higher weights ω (and thus makes the algorithm more exploitative), while larger values of q result in a more uniform distribution.

Let s_k be the solution of A that is selected by ant i according to Eq. (10) in a given iteration. Ant i then generates each solution element $s_{i,j}$ by sampling the Gaussian probability density function (PDF):

$$s_{i,j} \sim N(s_{k,j}, \sigma_{k,j}) \tag{13}$$

where $N(\mu, \sigma)$ represents the Gaussian PDF with mean μ and standard deviation σ.

In Eq. (13), $s_{k,j}$ represents the value that the solution s_k assigns to variable V_j, and the standard deviation $\sigma_{k,j}$ is computed according to:

$$\sigma_{k,j} = \xi \sum_{r=1}^{R} \frac{| s_{k,j} - s_{r,j} |}{R - 1} \tag{14}$$

where ξ is a user-supplied parameter of the algorithm. The effect of Eq. (14) is that the average distance from s_k to other solutions in the archive, for the j-th dimension, is computed, and is then multiplied by ξ. The parameter ξ plays a role in $\text{ACO}_{\mathbb{R}}$ similar to that of evaporation rate in other ACO algorithms. The higher the value of ξ, the less the extent to which the search is biased towards the area of the search space around the solutions stored in the archive, and the slower the algorithm will converge. Once each ant constructs its solution, the archive A is updated as described above.

If the top solution in the archive does not improve for I' iterations, then the algorithm is said to be *stagnated*. In such a case, the archvie is re-initialized with random solutions. The process terminates if the top solution has not improved for I_{conv} iterations, where I' and $I_{conv}$are user-supplied parameters with $I' < I_{conv}$. In all, besides I' and I_{conv}, the algorithm has four user-supplied parameters m, R, q, and ξ. The parameter m determines the number of ants; the

parameter R determines the number of solutions stored in the archive A; the parameter q controls the extent to which the top solutions in the archive will dominate solution construction (Eq. 11); and the parameter ξ influences the degree of diversity in solution construction (Eq. 14).

Our quality evaluation function uses a set of training set/test set pairs $\mathcal{T}$. In other words, $\mathcal{T}$ consists of a set of pairs $(\mathcal{D}_{train}, \mathcal{D}_{test})$ where each $\mathcal{D}_{train}$ is to be used as a training set and $\mathcal{D}_{test}$ is to be used as a test set. To compute the quality of a candidate solution $s = (a, b, \alpha)$, we apply an HONEST network, using the regularization parameter settings specified by the vector s, to the dataset pairs of $\mathcal{T}$. For each dataset pair $(\mathcal{D}_{train}, \mathcal{D}_{test}) \in \mathcal{T}$, we start with an HONEST network with randomly-initialized exponents and weights; the exponents are randomly initialized, with a uniform distribution, in the range $[0.9, 1.1]$, and all other weights are initialized, with a uniform distribution, in the range $[-0.25, 0.25]$. The randomly-initialized HONEST network is then trained using the Resilient Propagation (R-Prop) algorithm [39] for 50 epochs using the training set $\mathcal{D}_{train}$. The performance of the network is then evaluated as the accuracy rate on the test set $\mathcal{D}_{test}$. The accuracy rate is computed as the ratio of the correctly classified patterns of $\mathcal{D}_{test}$ to the total number of patterns of $\mathcal{D}_{test}$. The quality measure Q is computed as the average of the accuracy rate on the datasets in $\mathcal{T}$.

R-Prop [39] is a classic neural network training algorithm, that has been previously applied to HONEST [9], [10]. Unlike the well-known Back-Propagation algorithm [40], R-Prop uses only the sign of the gradient, not the magnitude. Each weight w_i has its own adaptable step size s_i. Note that we use the generic term weight to refer to all adaptable parameters in HONEST, including exponents, multiplicative weights, as well as biases.

At iteration t of R-Prop,

$$\Delta w_i = \begin{cases} -s_i(t) & \text{if } \frac{\partial E}{\partial w_i}(t) > 0 \\ +s_i(t) & \text{if } \frac{\partial E}{\partial w_i}(t) < 0 \\ 0 & \text{if } \frac{\partial E}{\partial w_i}(t) = 0 \end{cases} \tag{15}$$

In other words, if $\frac{\partial E}{\partial w_i}(t)$ is positive, then this means that a decrease in the value of $w_i(t)$ will result in a decrease in the error E; thus, $w_i(t)$ is decreased by $s_i(t)$. On the other hand, if $\frac{\partial E}{\partial w_i}(t)$ is negative, then this means that an increase in $w_i(t)$ will result in a decrease in E; therefore, $w_i(t)$ is increased by $s_i(t)$.

The step size s_i is itself adapted every epoch, as follows:

$$s_i(t) = \begin{cases} \gamma^+ \cdot s_i(t-1) & \text{if } \frac{\partial E}{\partial w_i}(t) \cdot \frac{\partial E}{\partial w_i}(t-1) = 1 \\ \gamma^- \cdot s_i(t-1) & \text{if } \frac{\partial E}{\partial w_i}(t) \cdot \frac{\partial E}{\partial w_i}(t-1) = -1 \\ s_i(t-1) & \text{if } \frac{\partial E}{\partial w_i}(t) \cdot \frac{\partial E}{\partial w_i}(t-1) = 0 \end{cases} \tag{16}$$

where $0 < \gamma^- < 1 < \gamma^+$. All the step sizes s_i are initialized to Δ_0, where Δ_0 is a third parameter of the R-Prop algorithm (along with γ^- and γ^+).

In R-Prop, whenever the partial derivative changes sign (i.e. whenever $\frac{\partial E}{\partial w_i}(t)$ and $\frac{\partial E}{\partial w_i}(t-1)$ have different signs), indicating that a minimum was missed, the previous weight change is reversed and a flag is set to prevent the step size from being updated in the next iteration.

IV. Experimental Methodology

A. Experiment A

In Experiment A, the $\text{ACO}_{\mathbb{R}}$ algorithm is used to optimize the three HONEST regularization parameters a, b, and α, as described in Section III. The objective of this experiment is to determine an optimized setting of those three parameters.

The quality evaluation function Q uses a suite of five datasets obtained from the popular University of California Irvine (UCI) machine learning repository. Table I shows important characteristics of the datasets used in Experiments A and B of this paper

In both Experiments A and B, each dataset is divided into ten mutually exclusive partitions (folds), with approximately the same number of instances and roughly the same class distribution in each fold. In Experiment A, one fold is randomly selected to be the test set, and the other nine folds are merged and used as the training set.

In both Experiments A and B, each dataset went through the following preprocessing steps before being presented to the HONEST network. Each continuous (numeric) attribute was scaled to the range $[0.1, 0.9]$. Each categorical attribute with c category labels was transformed to c network inputs, one input for each category label. For each pattern, the input corresponding to the category label for each categorical attribute was set to 0.9, and all the others were set to 0.1. Any missing value for a continuous attribute was set to the mean value for that attribute. Any missing value for a categorical attribute was set to the mode (the most popular category) for that attribute. If the number of classes is m, then the network will have m output neurons, one corresponding to each class. For each pattern, the target value for the output neuron corresponding to the correct class was set to 0.9, and for all other output neurons was set to 0.1.

In Experiment A, the number of HONEST's hidden neurons was set to $(n + m)/2$, where n and m represent the number of input and output neurons, respectively.

Table II shows our $\text{ACO}_{\mathbb{R}}$ parameters used in Experiment A, which follow [32]. Table III shows our R-Prop parameter settings used in both Experiment A and Experiment B. These parameter values represent typical settings for R-Prop [41]. The number of R-Prop epochs in Experiment A was 50.

B. Experiment B

Experiment B uses the evolved HONEST regularization parameter settings determined in Experiment A, and compares HONEST to Support Vector Machines (SVM) using 20 datasets (that do not include the 5 datasets used in Experiment A).

In Experiment B, each dataset is partitioned into 10 mutually exclusive folds, as described in Sect. IV.A. Each method, HONEST and SVM, is applied to the dataset 10 times, where each time a different fold takes its turn as the test set and

TABLE I
CHARACTERISTICS OF THE DATASETS USED IN THE EXPERIMENTS.

	Dataset	Instances	Classes	Features		
				Total	Numeric	Categorical
Experiment A	breast-tissue	106	6	9	9	0
	ecoli	336	8	7	7	0
	hay	132	3	4	0	4
	liver-disorders	345	2	6	6	0
	pima	768	2	8	8	0
Experiment B	automobile	205	7	25	15	10
	balance	625	3	4	0	4
	breast-l	283	2	9	0	9
	breast-p	198	2	32	32	0
	breast-w	569	2	30	30	0
	chess	3,196	2	36	0	36
	credit-a	690	2	14	6	8
	credit-g	1,000	2	20	7	13
	horse	366	2	22	7	15
	ionosphere	351	2	34	34	0
	monks	556	2	6	0	6
	parkinsons	195	2	22	22	0
	pop	90	3	8	0	8
	s-heart	270	2	13	6	7
	thyroid	215	3	5	5	0
	transfusion	722	2	4	4	0
	ttt	958	2	9	0	9
	voting	425	2	16	0	16
	wine	178	3	13	13	0
	zoo	101	7	16	0	16

TABLE II
PARAMETER SETTINGS OF THE $\text{ACO}_{\mathbb{R}}$ ALGORITHM.

Parameter	Description	Setting
m	number of ants	5
R	size of archive	90
q	influence of top solutions	0.05
ξ	width of search	0.68
I'	# of iterations after which restart is triggered if the top solution has not improved	650
I_{conv}	# of iterations after which algorithm terminates if the top solution has not improved	5000

TABLE III
PARAMETER SETTINGS OF THE R-PROP ALGORITHM.

Parameter	Description	Setting
γ^+	acceleration multiplier	1.2
γ^-	deceleration multiplier	0.5
Δ_0	initial step size	0.07

the other nine folds as the training set. Because HONEST is stochastic, the entire process is repeated for HONEST 10 times—using a different random seed each time. The accuracy rate on each of the 10 test set folds is recorded, and the average test set performance, aggregated over all ten folds, is reported as being representative of the performance of each method. In the case of SVM, this is the average test set accuracy over 10 runs (1 run for each fold); for HONEST, this is the average over 100 runs (10 repetitions for each fold).

In Experiment B, HONEST is trained by R-Prop for 100 epochs (unlike Experiment A where only 50 epochs were used). The remaining R-Prop parameter settings were the same as in Experiment A and are shown in Table III. In Experiment B, the number of HONEST's hidden neurons was set to $(n + m)$, where n is the number of input units, and m is the number of output neurons. We used Weka's [42] SMO implementation of SVM, and used Weka's default parameters for SVM.

V. EXPERIMENTAL RESULTS

A. Experiment A

TABLE IV
EXPERIMENT A: EVOLVED HONEST REGULARIZATION PARAMETERS.

Parameter	Description	Setting
a	inverted-bell half-width	3.78
b	inverted-bell slope	2.31
α	weight of regularization term	0.0635

Figs. 5 and 6 show the evolution of the HONEST regularization parameters (a, b, α) over the course of execution of $\text{ACO}_{\mathbb{R}}$. In Fig. 5, the x-axis represents the $\text{ACO}_{\mathbb{R}}$ iteration number, and the y-axis represents the value of the a and b parameters. In Fig. 6, the x-axis similarly represents the $\text{ACO}_{\mathbb{R}}$ iteration number, and the y-axis the value of α. The final evolved values of the regularization parameters are reported in Table IV. Fig. 7 contrasts the ψ function with the evolved parameter settings to the ψ function with the initial parameter settings. Comparing the two parameter configurations, we note the following:

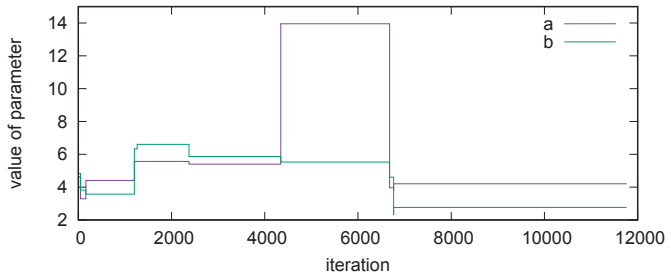

Fig. 5. Evolution of the a and b regularization parameters. The x-axis represents the $\text{ACO}_{\mathbb{R}}$ iteration number, and the y-axis represents the corresponding parameter value at that iteration.

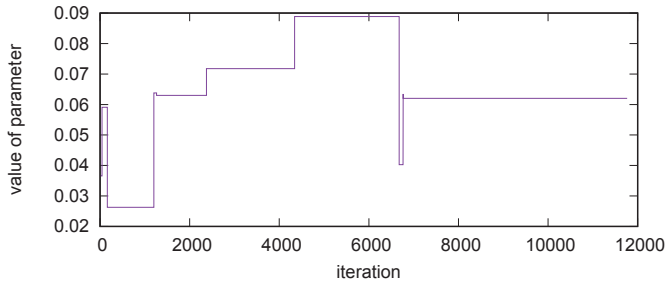

Fig. 6. Evolution of the α parameter. The x-axis represents the $\text{ACO}_{\mathbb{R}}$ iteration number, and the y-axis represents the corresponding value of α at that iteration.

- The initial ψ function is flat and near-zero as its input exponent goes from zero to just past ± 2, whereas the evolved ψ function starts rising much earlier.
- The "inverted legs" of the evolved ψ are noticeably wider than the initial ψ. The initial ψ is nearly 0.9 when its input is nearly ± 5, and is near-one when its input is nearly ± 7. In contrast, the evolved ψ is nearly 0.9 when its input is nearly ± 7, and when its input is ± 9, its output is still slightly less than 1.

B. Experiment B

Table V shows the results of applying HONEST (with the evolved regularization parameters) and SVM to the 20 datasets identified in Table I. Each entry in this table represents the average test set accuracy, aggregated over the 10 cross-validation folds and the 10 repetitions (in the case of HONEST). The best performance for each dataset is indicated in boldface. The results indicate that HONEST has better test set accuracy in 9 datasets, and SVM in 11 datasets.

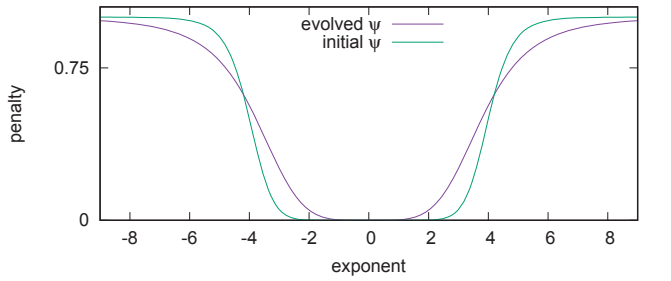

Fig. 7. Comparison of the generalized inverted bell function, ψ, with the final evolved parameter settings and with the initial parameter settings.

TABLE V
EXPERIMENT B: AVERAGE TEST SET PREDICTIVE ACCURACY RESULTS FOR SVM AND FOR HONEST WITH THE EVOLVED REGULARIZATION PARAMETERS.

Dataset	SVM	HONEST
automobile	**68.74**	62.84
balance	90.83	**91.00**
breast-l	71.68	**72.03**
breast-p	**76.29**	70.43
breast-w	**97.90**	82.44
chess	95.72	**96.54**
credit-a	84.93	**85.71**
credit-g	73.90	**75.38**
horse	**81.51**	81.02
ionosphere	88.50	**90.94**
monks	63.64	**64.60**
parkinsons	**87.16**	80.93
pop	**72.50**	65.25
s-heart	**84.45**	82.15
thyroid	**88.88**	84.74
transfusion	71.75	**72.26**
ttt	**98.42**	95.40
voting	92.97	**94.80**
wine	**98.82**	94.06
zoo	**98.75**	98.50
#wins	**11**	9

TABLE VI
EXPERIMENT B: RESULTS OF APPLYING A (TWO-TAILED) WILCOXON SIGNED-RANKS TEST COMPARING SVM TO HONEST WITH THE EVOLVED REGULARIZATION PARAMETERS.

Comparison	N	W	z	p	sig.?
HONEST vs. SVM	20	61	-1.64	0.101	no

Table VI reports the results of applying a Wilcoxon signed-ranks test to the results of Table V. The results indicate that no statistically significant difference is detected at the 0.05 conventional threshold.

VI. CONCLUSIONS & FUTURE WORK

In this paper, we have used $\text{ACO}_{\mathbb{R}}$, an ant colony algorithm for continuous optimization, to optimize the parameters of the HONEST network's regularization process. We identified a set of regularization parameter settings (reported in Table IV) that we propose should be used as default parameters in any future work with HONEST.

Comparing HONEST with the evolved parameter settings to Support Vector Machines (SVM), a machine learning technique that is widely acknowledged to have state-of-the-art performance, we find that HONEST's test set predictive accuracy is competitive with SVM, with no statistically significant difference detected between the two.

In future work, we would like to explore mechanisms for pruning hidden neurons dynamically while the network is

being trained. In such a case, we would add an additional regularization component that favors small weights (between hidden and output neurons). Hidden neurons whose incoming exponents and outgoing weights are small would then be considered candidates for exploratory pruning.

ACKNOWLEDGMENT

Partial support of a grant from the Brandon University Research Council is gratefully acknowledged.

REFERENCES

[1] D. E. Rumelhart, G. E. Hinton, and J. L. McClelland, "A general framework for parallel distributed processing," *Parallel Distributed Processing*, vol. 1, no. 2, 1986.

[2] M. Fallahnezhad, M. H. Moradi, and S. Zaferanlouei, "A hybrid higher order neural classifier for handling classification problems," *Expert Systems with Applications*, vol. 38, no. 1, pp. 386–393, 2011.

[3] M. Zhang, *Applied Artificial Higher Order Neural Networks for Control and Recognition*. IGI Global Press, 2016.

[4] C. L. Giles and T. Maxwell, "Learning, invariance, and generalization in high-order neural networks," *Applied Optics*, vol. 26, no. 23, pp. 4972–4978, 1987.

[5] A. Martínez-Estudillo, F. Martínez-Estudillo, C. Hevás-Martínez, and N. García-Pedrajas, "Evolutionary product-unit based neural networks for regression," *Neural Networks*, vol. 19, pp. 477–486, 2006.

[6] A. Abdelbar, "Achieving superior generalisation with a high order neural network," *Neural Computing & Applications*, vol. 7, no. 2, pp. 141–146, 1998.

[7] A. M. Abdelbar, S. Attia, and G. A. Tagliarini, "A hybridization of Bayesian and neural learning," *Neurocomputing*, vol. 48, no. 1, pp. 443–453, 2002.

[8] A. Abdelbar and G. Tagliarini, "HONEST: A new high order feed-forward neural network," *IEEE International Conference on Neural Networks*, vol. 2, pp. 1257–1262, 1996.

[9] I. El-Nabarawy, A. Abdelbar, and D. Wunsch, "Levenberg-Marquardt and conjugate gradient methods applied to a high order neural network," in *Proceedings IEEE International Joint Conference on Neural Networks*, 2013, pp. 2162–2132.

[10] I. El-Nabarawy and A. Abdelbar, "Advanced learning methods and exponent regularization applied to a high order neural network," *Neural Computing & Applications*, vol. 25, no. 3, pp. 897–910, 2015.

[11] A. M. Abdelbar, I. El-Nabarawy, D. C. Wunsch, and K. M. Salama, "Ant colony optimization applied to the training of a high order neural networks with adaptable exponential weights," in *Applied Artificial Higher Order Neural Networks for Control and Recognition*, M. Zhang, Ed. IGI Global Press, 2016, pp. 362–374.

[12] H.-C. Tsai, "Hybrid high order neural networks," *Applied Soft Computing*, vol. 9, no. 3, pp. 874–881, 2009.

[13] ——, "Predicting strengths of concrete-type specimens using hybrid multilayer perceptrons with center-unified particle swarm optimization," *Expert Systems with Applications*, vol. 37, no. 2, pp. 1104–1112, 2010.

[14] H.-C. Tsai, Y.-W. Wu, Y.-Y. Tyan, and Y.-H. Lin, "Programming squat wall strengths and tuning associated codes with pruned modular neural network," *Neural Computing & Applications*, vol. 23, no. 3, pp. 741–749, 2013.

[15] R. Durbin and D. E. Rumelhart, "Product units: A computationally powerful and biologically plausible extension to backpropagation networks," *Neural Computation*, vol. 1, no. 1, pp. 133–142, 1989.

[16] Y. Shin and J. Ghosh, "The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation," *Proceedings of the International Joint Conference on Neural Networks, IJCNN-91*, vol. 1, pp. 13–18, 1991.

[17] S. Narayan, "ExpoNet: A generalization of the multi-layer perceptron model," *Proceedings of the World Congress on Neural Networks*, vol. 3, pp. 494–497, 1993.

[18] A. Ivakhnenko, "Polynomial theory of complex systems," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 1, pp. 364–378, 1971.

[19] V. Puig, M. Witczak, F. Nejjari, J. Quevedo, and J. Korbicz, "A GMDH neural network-based approach to passive robust fault detection using a constraint satisfaction backward test," *Engineering Applications of Artificial Intelligence*, vol. 20, no. 7, pp. 886–897, 2007.

[20] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA, USA: MIT Press, 2004.

[21] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas, "Data mining with an ant colony optimization algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 321–332, 2002.

[22] D. Martens, M. De Backer, R. Haesen, J. Vanthienen, M. Snoeck, and B. Baesens, "Classification with ant colony optimization," *IEEE Transactions on Evolutionary Computation*, vol. 11, pp. 651–665, 2007.

[23] K. Salama, A. Abdelbar, and A. Freitas, "Multiple pheromone types and other extensions to the Ant-Miner classification rule discovery algorithm," *Swarm Intelligence*, vol. 5, no. 3-4, pp. 149–182, 2011.

[24] K. Salama, A. Abdelbar, F. Otero, and A. Freitas, "Utilizing multiple pheromones in an ant-based algorithm for continuous-attribute classification rule discovery." *Applied Soft Computing*, vol. 13, no. 1, pp. 667–675, 2013.

[25] F. Otero, A. Freitas, and C. Johnson, "A new sequential covering strategy for inducing classification rules with ant colony algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 17, pp. 64–76, 2013.

[26] F. Otero and A. Freitas, "Improving the interpretability of classification rules discovered by an ant colony algorithm," in *Genetic and Evolutionary Computation Conference (GECCO-2013)*, 2013, pp. 73–80.

[27] U. Boryczka and J. Kozak, "An adaptive discretization in the ACDT algorithm for continuous attributes," in *3rd International Conference on Computational Collective Intelligence: Technologies and Applications (ICCCI'11)*. Berlin, Heidelberg: Springer, 2011, pp. 475–484.

[28] F. Otero, A. Freitas, and C. Johnson, "Inducing decision trees with an ant colony optimization algorithm," *Applied Soft Computing*, vol. 12, no. 11, pp. 3615–3626, 2012.

[29] K. Salama and A. Freitas, "Learning Bayesian network classifiers using ant colony optimization," *Swarm Intelligence*, vol. 7, no. 2-3, pp. 229–254, 2013.

[30] ——, "Ant colony algorithms for constructing Bayesian multi-net classifiers," *Intelligent Data Analysis*, vol. 19, no. 2, pp. 233–257, 2015.

[31] K. Socha and C. Blum, "An ant colony optimization algorithm for continuous optimization: Application to feed-forward neural network training," *Neural Computing & Applications*, vol. 16, pp. 235–247, 2007.

[32] T. Liao, K. Socha, M. Montes de Oca, T. Stützle, and M. Dorigo, "Ant colony optimization for mixed-variable optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 503–518, 2014.

[33] K. Socha and C. Blum, "Training feed-forward neural networks with ant colony optimization: An application to pattern classification," in *5th International Conference on Hybrid Intelligent Systems (HIS '05)*, 2005, pp. 233–238.

[34] A. M. Abdelbar and K. M. Salama, "A gradient-guided ACO algorithm for neural network learning," in *Proceedings IEEE Swarm Intelligence Symposium (SIS-15)*, 2015, pp. 1133–1140.

[35] K. M. Salama and A. M. Abdelbar, "Learning neural network structures using ant colony optimization," *Swarm Intelligence*, vol. 9, no. 4, pp. 229–265, 2015.

[36] H. B. Alwan and K. R. Ku-Mahamud, "Optimizing support vector machine parameters using continuous ant colony optimization," in *Proceedings International Conference on Computing and Convergence Technology (ICCCT-12)*, 2012, pp. 164–169.

[37] T. Liao, M. Montes de Oca, D. Aydin, T. Stützle, and M. Dorigo, "An incremental ant colony algorithm with local search for continuous optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-11)*, 2011, pp. 125–132.

[38] A. M. Abdelbar and K. M. Salama, "An extension of the $ACO_{\mathbb{R}}$ algorithm with time-decaying search width, with application to neural network training," in *Proceedings IEEE International Joint Conference on Neural Networks (IJCNN-16)*, 2016.

[39] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," *IEEE International Conference on Neural Networks*, pp. 586–591, 1993.

[40] P. J. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. New York, NY, USA: Wiley, 1994.

[41] R. Reed and R. Marks, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. Cambridge, MA, USA: MIT Press, 1999.

[42] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann, 2010.