Proceedings of the 28th IEEE
EMBS Annual International Conference
New York City, USA, Aug 30-Sept 3, 2006

Inferring Network Interactions Using Recurrent Neural Networks and Swarm Intelligence

Habtom W. Ressom, Yuji Zhang, Jianhua Xuan, Yue Wang, and Robert Clarke

Abstract— **We present a novel algorithm combining artificial neural networks and swarm intelligence (SI) methods to infer network interactions. The algorithm uses ant colony optimization (ACO) to identify the optimal architecture of a recurrent neural network (RNN), while the weights of the RNN are optimized using particle swarm optimization (PSO). Our goal is to construct an RNN that mimics the true structure of an unknown network and the time-series data that the network generated. We applied the proposed hybrid SI-RNN algorithm to infer a simulated genetic network. The results indicate that the algorithm has a promising potential to infer complex interactions such as gene regulatory networks from time-series gene expression data.**

I. INTRODUCTION

Recent advances in bio-technologies for high throughput data acquisition have generated challenges in the discovery of network interactions in biological systems. The challenges provide computer scientists, statisticians, and engineers with opportunities to expand their knowledge of intelligent methods to provide models for better understanding of biological systems. The field of system modeling plays a significant role in the discovery of network interactions. Several system modeling approaches have been proposed to reverse-engineer network interactions including a variety of continuous or discrete, static or dynamic, quantitative or qualitative methods [1-6].

The use of computational intelligence methods for system modeling has gained particular interest, because it requires little a priori knowledge about the underlying system and the model can be derived from data. Artificial neural networks (ANNs) have been one of the approaches for nonlinear and dynamic system modeling problems. In [7], ANNs are developed for solving the inverse metabolic problem. A neural model is used to simulate the dynamics of the lambda phage regulatory system [8]. Genetic algorithms (GAs) have also been applied to decipher genetic networks from gene expression data [9-11]. Shin and Iba [11] developed an inference algorithm based on GAs for the optimization of the influence matrix of gene regulatory network. In [13], GAs and ANNs are combined to determine gene interactions in temporal gene expression data.

In this paper, we propose to apply a hybrid of ANNs and swarm intelligence (SI) methods [12] to infer network interactions from time-series data. The architecture and the synaptic weights of a recurrent neural network (RNN) are optimized using ant colony optimization (ACO) and particle swarm optimization (PSO) methods, respectively. Unlike previous computational methods, which targeted at one-step-ahead prediction of time-series data [13], our method enables a multi-step-ahead prediction. This is achieved through our RNN, which is self-evolutionary. The RNN starts with a given initial condition, evolves, and eventually reaches final states. The proposed hybrid SI-RNN algorithm selects the architecture of the RNN and weights not only to mimic the response of the unknown network at each time point but also to identify the structure of the network that generated the time-series data. This is a challenging task given that there may be many possible structures with responses that closely match the generated data. The algorithm evaluates various structures through the cross-validation method to avoid the selection of a wrong structure and to make sure that the correct structure is identified despite the presence of noise and complexity of the unknown network. We successfully applied the algorithm to infer simulated network interactions.

II. INFERRING NETWORK INTERACTION USING SI-RNN

In building RNNs for inferring networks, the identification of the correct structure and determination of the free parameters (weights and biases) to mimic the real data is a challenging task given the limited available quantity of data. For example, in inferring a gene regulatory network from microarray data, the number of time points is considerably low compared to the number of genes involved. Considering the complexity of the biological system, it is difficult to adequately describe the pathways involving a large number of genes with few time points. In this paper, we apply ACO and PSO methods to select the architecture of an RNN and to update its free parameters, respectively.

A. Recurrent Neural Network

Fig. 1(a) shows an RNN, where the output of each neuron is fed back to its input after a unit delay and is connected to other neurons. It can be used as a model of gene regulatory network, where every gene in the network is considered as a

H.W. Ressom and Y. Zhang are with the Department of Biostatistics, Bioinformatics, and Biomathematics, Lombardi Comprehensive Cancer Center, Georgetown University, 3970 Reservoir Rd NW, Washington, DC 20057 USA.

J. Xuan is with the Department of Electrical Engineering and Computer Science, The Catholic University of America, 620 Michigan Ave, NW, Washington, DC 20064 USA.

Y. Wang is with the Department of Electrical and Computer Engineering, Advanced Research Institute, Virginia Polytechnic Institute and State University, 4300 Wilson Blvd, Arlington, VA 22203 USA.

R. Clarke is with the Departments of Oncology and Physiology & Biophysics, Lombardi Comprehensive Cancer Center, Georgetown University, 3970 Reservoir Rd NW, Washington, DC 20057 USA.

neuron; RNN model considers not only the interactions between genes but also gene self-regulation.

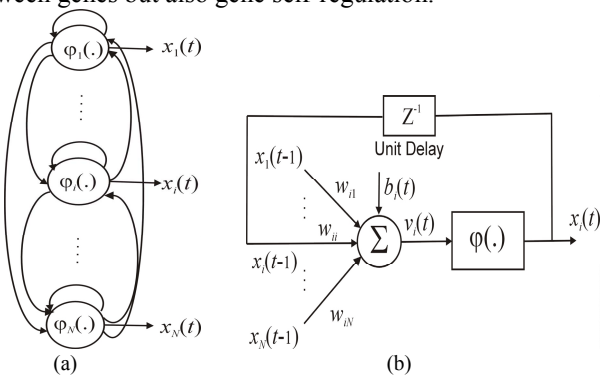


Fig. 1. (a) Architecture of a fully connected RNN; (b) Details of a single recurrent neuron.

Fig. 1(b) illustrates the details of the *i*th self-feedback neuron (e.g. *i*th gene in the GRN), where v_i, known as the induced local field (activation level), is the sum of the weighted inputs (the regulation of other genes) to the neuron (*i*th gene); and φ(.) represents an activation function (integrated regulation of the whole RNN on *i*th gene), which transforms the activation level of a neuron into an output signal (regulation result). The induced local field and the output of the neuron, respectively, are given by:

$$v_i(t) = \sum_{j=1}^{N} w_{ij} x_j(t-1) + b_i \tag{1}$$

$$x_i(t) = \varphi(v_i(t)) \tag{2}$$

where the synaptic weights w_{i1}, w_{i2},…, w_{iN} define the strength of connection between the *i*th neuron (e.g. *i*th gene) and its inputs (e.g. expression level of genes). Such synaptic weights exist between all pairs of neurons in the network. b_i denotes the bias for the *i*th neuron. We denote $\vec{s}$ as a structure vector that describes the architecture of the network, and $\vec{w}$ as a weight vector that consists of all the synaptic weights and biases in the network. $\vec{s}$ and $\vec{w}$ are adapted during learning to yield the desired network outputs. The activation function φ introduces nonlinearity to the model. When information about the complexity of the underlying system is available, a suitable activation function can be chosen (e.g. linear, logistic, sigmoid, threshold, hyperbolic tangent sigmoid or Gaussian function.) If no prior information is available, our algorithm uses the hyperbolic tangent sigmoid function.

As a cost function, we use the sum-squared error between the expected output and the network output across time (from the initial time point t_0 to the final time point t_f) and across neurons in the network. The cost function can be written as:

$$E(\vec{w}) = \sum_{t=t_0}^{t_f} \sum_{i=1}^{N} \left[x_i(t) - \hat{x}_i(t) \right]^2 \tag{3}$$

where $x_i(t)$ and $\hat{x}_i(t)$ are the true and predicted values (expression levels) for the *i*th neuron (gene) at time *t*. The goal is to determine the structure vector $\vec{s}$ and weight vector $\vec{w}$ that minimize this cost function. We propose ACO and PSO to optimize $\vec{s}$ and $\vec{w}$, respectively.

B. Ant Colony Optimization

Ant colony optimization studies artificial systems that take inspiration from the behavior of real ant colonies. The basic idea of ACO is that a large number of simple artificial agents are able to build good solutions to solve hard combinatorial optimization problems via low-level based communications.

We propose to use ACO to optimize the structure vector $\vec{s}$. Each possible network structure $\vec{s}$ is defined by a combination of *n* features $\vec{s} = [s_1\ s_2 \cdots s_n]$, where s_j is an *n*-bit binary string that indicates which neurons are controlled by neuron *j*. Each s_j is selected from 2^n candidate features. For each neuron *j*, we define the function in Eq. (4) to determine the probability of selecting a feature *i* among the 2^n candidate features:

$$P_i^j(k) = \frac{\tau_i^j(k)}{\sum_{i=1}^{n} \tau_i^j(k)} \qquad j = 1,..,n \quad i = 1,...,2^n \tag{4}$$

where $\tau_i^j(k)$ is the amount of pheromone trail for the *i*th feature at iteration *k*. At *k*=0, $\tau_i^j(k)$ is set to a constant for all features, allowing each feature to have equal probability of being selected. Thus, in the first iteration, each ant chooses randomly *n* features that make up a structure ($\vec{s}$, a trail). Let $\vec{s}$ be an ant consisting of *n* features $\vec{s} = [s_1\ s_2 \cdots s_n]$. Depending on the performance of $\vec{s}$, the amount of pheromone trail of all features in $\vec{s}$ will be updated. The performance function here is evaluated on the basis of mimicking the response of the system under study. To estimate the performance of $\vec{s}$, we construct an RNN that has the structure defined in $\vec{s}$. Then, we optimize the weights of the RNN using PSO. The response of the resulting RNN will be compared with the measured/observed response of the system under study. The amount of pheromone trail for each element in $\vec{s}$ is updated in proportion to the performance of the structure. Assuming, the *i*th feature for the *j*th neuron was in $\vec{s}$, the corresponding amount of pheromone trail will be updated as follows:

$$\tau_i^j(k+1) = \rho . \tau_i^j(k) + \Delta\tau_i(k) \tag{5}$$

where ρ is a constant between 0 and 1, representing the evaporation of pheromone trails. $\Delta\tau_i(k)$ is an amount proportional to the performance by $\vec{s}$. $\Delta\tau_i(k)$ is set to zero, if $s_i \notin \vec{s}$. This update is made for all *N* ants ($\vec{s}_1, \ldots, \vec{s}_N$). Note that at *k*=0, $\Delta\tau_i(k)$ is set zero for all features. The updating rule allows trails that yield good performance to have their amount of pheromone trail increased, while others will evaporate. As the algorithm progresses, features with large amounts of pheromone trails influence the probability function to lead the ants towards them.

C. Particle Swarm Optimization

In the PSO algorithm, each particle is represented as a vector $\vec{w}_i$ and instantaneous trajectory vector $\Delta\vec{w}_i(k)$, describing its direction of motion in the search space at iteration *k*. The index *i* refers to the *i*th particle. The core of the PSO algorithm is the position update rule (6) which governs the movement of each of the *n* particles through the search space.

$$\vec{w}_i(k+1) = \vec{w}_i(k) + \Delta\vec{w}_i(k+1)$$

$$\Delta\vec{w}_i(k+1) = \chi(\Delta\vec{w}_i(k) + \Phi_1(\vec{w}_{i,best}(k) - \vec{w}_i(k)) + \Phi_2(\vec{w}_{G,best}(k) - \vec{w}_i(k)))$$

where

$$\Phi_1 = c_1\begin{bmatrix} r_{1,1} & 0 & 0 & 0 \\ 0 & r_{1,2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & r_{1,D} \end{bmatrix} \text{ and } \Phi_2 = c_2\begin{bmatrix} r_{2,1} & 0 & 0 & 0 \\ 0 & r_{2,2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & r_{2,D} \end{bmatrix} \quad (6)$$

At any instant, each particle is aware of its individual best position, $\vec{w}_{i,best}(k)$, as well as the best position of the entire swarm, $\vec{w}_{G,best}(k)$. The parameters c_1 and c_2 are constants that weight particle movement in the direction of the individual best positions and global best positions, respectively; and $r_{1,j}$ and $r_{2,j}$, $j = 1,2,\ldots D$ are random scalars distributed uniformly between 0 and 1, providing the main stochastic component of the PSO algorithm.

The constriction factor, χ, may also help to ensure convergence of the PSO algorithm, and is set according to the weights c_1 and c_2 as in (7).

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, \quad \varphi = c_1 + c_2, \quad \varphi > 4 \quad (7)$$

The key strength of the PSO algorithm is the interaction among particles. The second term in (6), $\Phi_2(\vec{w}_{G,best}(k) - \vec{w}_i(k))$, is considered to be a "social influence" term. While this term tends to pull the particle towards the globally best solution, the first term, $\Phi_1(\vec{w}_{i,best}(k) - \vec{w}_i(k))$, allows each particle to think for itself. The net combination is an algorithm with excellent trade-off between total swarm convergence, and each particle's capability for global exploration. Moreover, the relative contribution of the two terms is weighted stochastically.

The algorithm consists of repeated application of the velocity and position update rules presented above. Termination can occur by specification of a minimum error criterion, maximum number of iterations, or alternately when the position change of each particle is sufficiently small as to assume that each particle has converged.

Selection of appropriate values for the free parameters of PSO plays an important role in the algorithm's performance. In our study, parameters c_1 and c_2 were arbitrarily selected (c_1 = 2.05, c_2 = 2.05), with constriction factor, χ, determined by (7) and the maximum velocity was set at 2.

D. SI-RNN

In this section, we illustrate how two SI methods, ACO and PSO, work together to optimize both $\vec{s}$ and $\vec{w}$ of an RNN to mimic a network of interactions. Each node in the true network will be represented by a neuron in the RNN. We assume that the number of nodes in the network is known (e.g. the number of genes for which a gene regulatory network is to be modeled is known), but the way the nodes interact is assumed to be unknown.

ACO starts with initial candidate ants that define various structures. For example, let Fig. 2a be the structure for the true interaction that has three nodes and Fig. 2b be one of the randomly selected initial structures by ACO. For each of these two systems, the corresponding structure vector $\vec{s}$ is shown in the figure, where s_j(j = 1, 2, 3) is a three-bit binary string that indicates which neurons (including itself) are controlled by the *j*th neuron. For example, if s_1 = [0 1 1], it implies that the first neuron "controls" all others except itself and s_2 = [0 0 1] implies that the second neuron controls the third neuron only.

PSO searches for the optimal weight vector $\vec{w}$ to minimize the difference between the output of the true network and the RNN using the training data. Only the elements of $\vec{w}$ that correspond to nonzero entries in $\vec{s}$ are updated by PSO. For example, the weight vector $\vec{w}$ in Fig. 2b only contains two variables: 0.25 and 0.4, corresponding to the two nonzero entries in the structure vector $\vec{s}$.

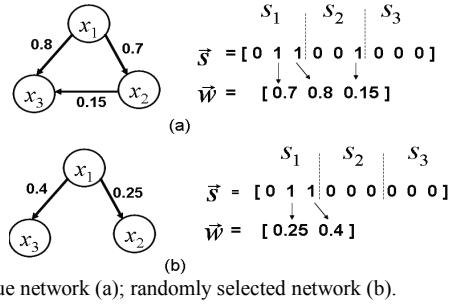


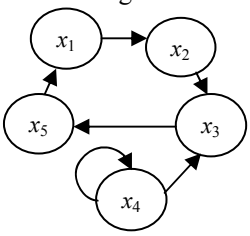
Fig. 2. True network (a); randomly selected network (b).

The optimal weight vectors for all randomly selected structures (ants) will be evaluated with previously unseen validation dataset. The validation dataset may be generated at different initial conditions or by perturbing the parameters of the true network. The performance of each particle in simulating the validation dataset will be returned to ACO to update the trails of the ants in the search space (i.e., update the structure vector $\vec{s}$). The new s_j's in $\vec{s}$ will be used to construct a new candidate structure. This will lead to a structure that is more similar to the global best structure than the previous one. The assumption in this algorithm is that the prediction error of an arbitrary network will be larger than a network that matches the correct structure. Through subsequent iterations, the ACO and PSO search for the optimal structure vector $\vec{s}$ and weight vector $\vec{w}$ to make accurate predictions.

III. Simulated Data and Results

We used the SI-RNN algorithm to identify the network in Fig. 3. Three datasets of 20 time points were generated from the network with different initial conditions: training, validation, and testing datasets. An RNN of five neurons with hyperbolic tangent sigmoid activation function was

considered. PSO used the training dataset to determine the optimal weight vector $\vec{w}$ for each structure vector $\vec{s}$ defined by ACO. The performance of each structure in predicting the outputs of the network in the validation dataset is used by ACO to determine the optimal structure. The algorithm was run 100 times. In each run, Eq. 3 was evaluated 1000 times to identify the structure that leads to the least cost. 54 runs (out of 100) predicted a RNN with identical structure to Fig. 3. Figure 4 shows the outputs of the true network and the predicted RNN for the testing dataset. 16% of the runs also predicted the true network structure, but they had two or three additional connections. The remaining 30% consisted of arbitrary structures.



$$x_1(t) = 0.5x_5(t-1)$$
$$x_2(t) = 0.3x_1(t-1)$$
$$x_3(t) = -0.6x_2(t-1) + x_4(t-1)$$
$$x_4(t) = 0.7(x_4(t-1))^2$$
$$x_5(t) = -0.8\sqrt{|x_3(t-1)|}$$

Fig. 3. A simulated five-node network.

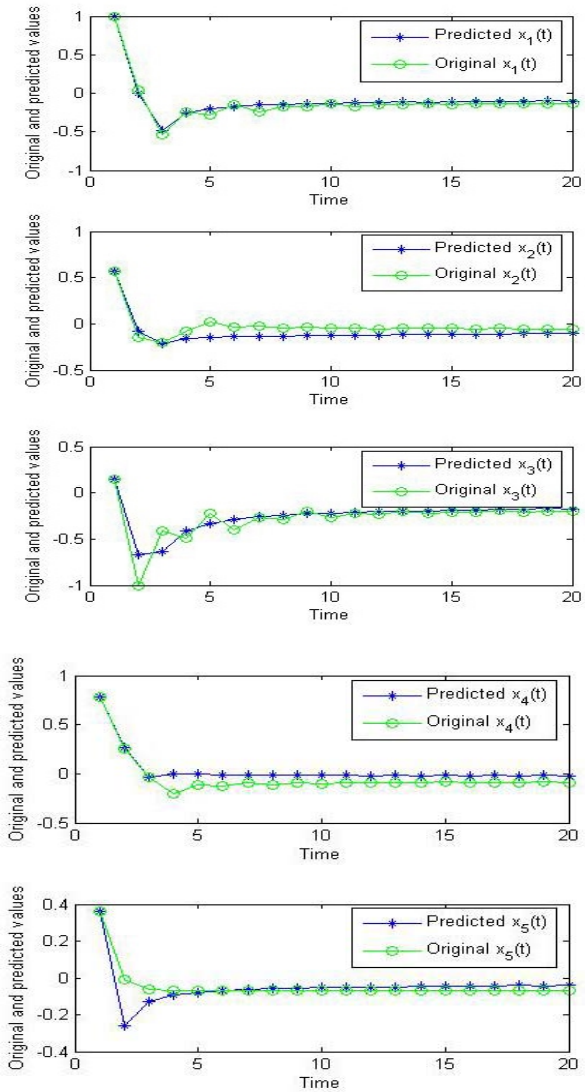


Fig. 4. Original and predicted outputs of the testing set.

IV. Conclusion

In this paper, we explored the combined advantages of the nonlinear and dynamic properties of RNN, and the global search capabilities of swarm intelligence methods to infer network interactions. The algorithm is tested in a dataset generated from a simulated network yielded promising results. Limitations of the proposed algorithm include: (i) while for linear systems the predicted parameters can be compared with the original parameters by using linear activation functions, for nonlinear problems the predicted parameters cannot be directly compared; and (ii) because of the stochastic properties of the algorithms, not all runs identify the correct network structure. Our future work will focus on improving the rate at which the correct structure is identified. Our ultimate goal is to apply the algorithm to infer gene regulatory networks from real time-series gene expression data. Since the real gene expression data contains noise, limited time points and unequal time intervals, we anticipate challenges in applying the proposed method for real data. To partially address these challenges, we plan to incorporate known biological information into the algorithm.

References

[1] I. Shmulevich, E.R. Dougherty, S. Kim, W. Zhang, "Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks," *Bioinformatics*, vol. 18, No. 2, 2002, pp. 261-274.

[2] W.A. Schmitt Jr., R.M. Raab, and G. Stephanopoulos, "Elucidation of gene interaction networks through time-lagged correlation analysis of transcriptional data," *Genome Research*, vol. 14, No. 8, 2004, pp. 1654–1663.

[3] N. Friedman, M. Linial, I. Nachman, D. Pe'er D, "Using Bayesian networks to analyze expression data," *J Comput Biol*., vol. 7, No. 3-4, 2000, pp. 601-620.

[4] P. D'haeseleer, X. Wen, S. Fuhrman, and R. Somogyi, "Linear modeling of mRNA expression levels during CNS development and injury," *Pacific Symposium on Biocomputing*, vol. 4, 1999, pp. 41-52.

[5] T. Chen, H.L. He, and G.M. Church, "Modeling gene expression with differential equations," *Pacific Symposium on Biocomputing*, vol. 4, 1999, pp. 29-40.

[6] S. Liang, S. Fuhrman, and R. Somogy, "REVEAL, a general reverse engineering algorithm for inference of genetic network architectures," *Pacific Symposium on Biocomputing*, vol. 3, 1998, pp. 18-29.

[7] P. Mendes and D. B. Kell, On the analysis of the inverse problem of metabolic pathways using artificial neural networks," *BioSystems*, vol. 38, 1996, pp. 15-28.

[8] J. Vohradsky, "Neural model of the genetic network," *J Biol Chem*.,vol. 276, no. 39, 2001, pp. 36168-36173.

[9] T. Ueda, I. Ono, and M. Okamoto, "Development of system identification technique based on real-coded genetic algorithm," *Genome Informatics*, vol. 13, 2002, pp. 386-387.

[10] S. Kikuchi, D. Tominaga, M Arita, K Takahashi, and M Tomita, "Dynamic modeling of genetic networks using genetic algorithm and S-system," *Bioinformatics*, vol. 19, No. 5, 2003, pp.643-650.

[11] A. Shin, and H. Iba, "Construction of genetic network using evolutionary algorithm and combined fitness function," *Genome Informatics*, vol.14, 2003, pp. 94-103.

[12] A.P. Engelbrecht, *Computational Intelligence: An Introduction*. New York: John Wiley, 2003.

[13] E. Keedwell, and A. Narayanan, "Discovering Gene Regulatory Networks with a Neural-Genetic Hybrid," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 2, No. 3, 2005, pp. 231-243.