Representing Uncertainty with a New Type of Stochastic Neural Networks

Jumpol Polvichai[1] **and Surapont Toomnark**[2]
Department of Computer Engineering King Mongkut's University of Technology Thonburi
126 Tungkru, Bangkok, 10140 Thailand, Tel: 662-470-9261, Fax: 662-872-5050
E-Mail: [1]jumpol@cpe.kmutt.ac.th, [2]surapontt@cpe.kmutt.ac.th

Abstract: Many interesting complex systems are stochastic. In order to model such complex systems, much ongoing research is looking at how to precisely model uncertainty in performance. In this paper, we proposed a novel type of stochastic neural network (SNN), in which dynamic features are added to the input layer allowing any non-deterministic system to be modeled. The SNNs capture randomness from the additional input nodes fed with internal random signals. These random signals, combined with weights between the additional nodes and the hidden nodes, allow stochastic output even though the network is deterministic. To validate this approach, a preliminary experiment was performed. To show the SNN's basic ability to represent uncertainty, a SNN model is trained to represent a model of beta distribution. Experiments verify the basic feasibility of the approach.

1. Introduction

To accurately characterize a complex system requires a model of the system as well as a model of the uncertainty impacting the system. That inherent uncertainty leads to uncertainty in the performance of the system. The importance of modeling uncertainty in system performance has been recognized in many fields [8][6][7]. Notice that for many of these systems, the uncertainty can not be captured as a normal distribution and will often not have been characterized at all. While a range of techniques exist to describe a system and its performance[1], techniques for capturing the uncertainty in that performance are not as readily available. This paper presents a novel approach to capturing both the performance of a system and the uncertainty related to that performance in a concise, easy to use form.

Previous approaches have shown significant potential for representing uncertainty as a special form of stochastic neural networks (SNNs)[9]. However, the implementation is very complicated and becomes more significantly more difficult when the number of input parameters grow. Alternatively, neural networks with stochastic resonance were introduced by using a time periodic signal as a stochastic element[6]. Although such stochastic neural networks are trained with an extended back propagation method, their performance is limited to very simple tasks. Recently, stochastic models of neural networks were used to representing complicated gene regulatory networks by using Poisson random signals, see [8]. However, such SNNs only capture Poisson and Normal uncertainty distributions. Thus, although promising neural networks have no effective way of handling arbitrary uncertainty distributions.

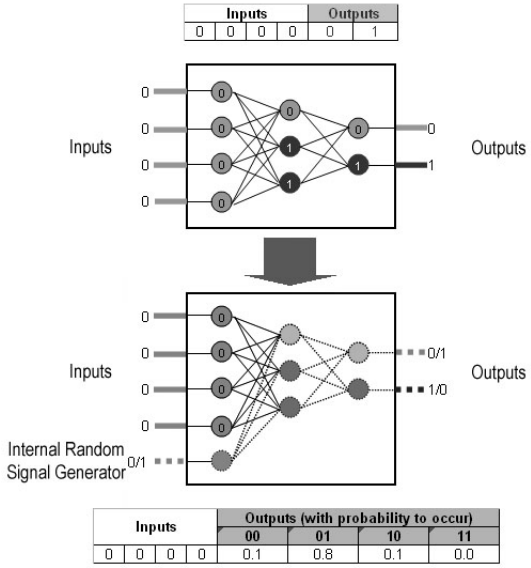

Figure 1. A traditional neural network can be transformed to a SNN by adding extra input nodes and feeding them with internal random signals to the input layer. These random signals are stochastically changed every time the network is executed. Output distributions can be generated by executing the network a number of times.

2. Algorithm and Approach

In this section, the concept of SNNs is described in detail. In the first step, data is collected from target system. Then, collected data is preprocessed so that it is in the right form for use in the learning process. The process of learning takes place, until the termination criteria is reached. Finally, the SNNs are used for representing the target system. Details are described below.

2.1 Stochastic Neural Network Models

With inspiration from the dynamic rearrangement [3], a concept called Stochastic Neural Networks (SNNs) is presented here. This concept allows output nodes to act stochastically even while input is held constant and internal nodes act deterministically. In general, an ANN is an interconnected, layer by layer, chain of simple processing nodes, where each node receives a number of inputs and sends an output to other nodes. Each node is deterministic in that its output is based entirely on its input values. To make output nodes act effectively stochastic, an internal stochastic component is included in the network, which is responsible for producing random signals. These internal random signals are simply treated as

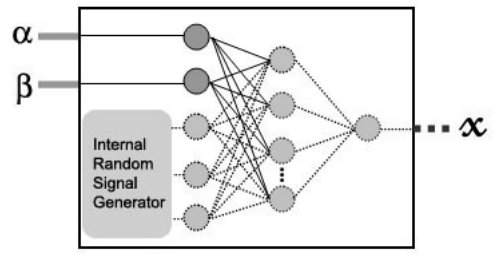

Figure 2. The network architecture for the experiment consists of three layers. The input layer has α, β and internal random signals. The output layer has only a variable x. The hidden layer usually has two layers of nodes.

additional input signals of the ANN, i.e., they are connected to every internal node by adjustable weights. Changing weights result in changing the behaviors of the stochastic networks. Thus, it makes possible to manipulate the neural networks stochastic behavior by adjusting the weights inside the network. If a target system has a high variation in outputs even for the same input configuration, in non-deterministic cases, the SNN adapts their weights to match these variances by being stochastic. If the target system is deterministic, which means the outputs are static, the SNN adapts the weights to ignore stochastic components.

As shown in Figure 1, a typical ANN network can be transformed into a SNN by adding extra input nodes to the input layer and feeding them with internal random signals. These random signals are stochastically changed every time the network is executed. Executing the network for a number of times, a distribution of possible outputs can be generated. The random signals that we normally use are random numbers internally generated from a uniform distribution between 0 and 1. All nodes in the network use sigmoid units. In addition, a three-layer feed-forward network is typically preferred since it is capable of representing any arbitrary function [7].

2.2 Data Collecting and Preprocessing

The initial step in modeling a very complex system is the collection of a large volume of data. This data either already exists or needs to be gathered and is stored in various forms. Data preprocessing techniques are then utilized to convert the data into a format which SNN can accept.

The process of creating and preprocessing training sets for SNNs is atypical. Because, in the learning process part, distributions of target outputs and actual outputs are required in fitness function. We need a set of distributions of target outputs for using in the evaluation process. Subsequently, each set training data is made up of two subsets: a set of every input value and a set of every output distribution. A simple way for creating output distributions may be executing as follow. First, each output is partitioned its all possible range into a number of disjoint subset or slots. However, the partitioning process can be done by any means; however it has to be the same process when partitioning the output of SNNs. Then, by collecting output data with the same input values for a number of times, the distribution of the outputs is computed by counting how many times the output values fall into each slot. The same procedure is applied when obtaining the distributions of outputs for the SNNs during learning process.

2.3 Learning Process

A genetic algorithm is a search technique loosely based on the mechanism of natural selection and genetics. Given an environment and a goal formulated as a fitness function, an initial population is generated at random and a set of genetic operators defined. New generations of individuals are generated using three common genetic operators: reproduction, crossover, and mutation. This process repeats until either a sufficiently fit individual is found or time expired. The solution of the problem is found in the final population.

For this work, each generation of the population contained 2,000 individuals. The chromosomes of each individual defined the weights of the SNN connection. All weights were randomly generated at the start. After evaluation of the training data set, the 500 best performers were kept and used to produce the 1500 new individuals, via genetic operations. Genetic algorithms were chosen for training because the relationship between input variables was not only non-linear, but also stochastic, which is problematic for back propagation methods due to the large number of local minima.

To evaluate each individual in every generation, every SNN is required to execute with the same input values (excluding all stochastic signals) a number of times, so that distributions of actual outputs can be measured. These actual output distributions are compared in the evaluation process against the target output distributions from training sets via the goodness-of-fit test or Pearson Chi-square (χ^2) test [4] :

$$\chi^2 = \sum_{i=1}^{n} \left(\frac{(f_s - f_s^*)^2}{f_s^*} \right), n = n_a + n_t, \tag{1}$$

where n denotes the number of all slots, n_a denotes the number of slots in actual distribution, n_t denotes number of slots in target distribution, f_s denotes observed frequency in a particular slot s, and f_s^* denotes predicted frequency in a particular slot s. The predicted frequency in a particular slot s can be calculated from:

$$f_s^* = \frac{t_o \cdot t_s}{T}, \tag{2}$$

where T denotes the total number of observations, t_o denotes the total number of observations in actual or target distributions, and t_s denotes the total number of observations in the same slots combined. From [4], we can get an index of the strength for the relation between these two distributions, by using the formula:

$$V = \sqrt{\frac{\chi^2}{N(k-1)}},$$

where N is the total frequency and $k = 2$ in this case. Then, this index is used to estimate the percentage of error between

actual and target distributions. In addition, we need to add an admissible constraint, which is an absolute difference between means of actual and target distributions, to the fitness function for directing the learning process to convert. Thus, the fitness function of individual i in population is:

$$Fitness_i = \frac{\sum_{d \in D} \sum_{p \in P} \left(\chi^2_{d,p} + |\mu_a^{d,p} - \mu_t^{d,p}| \right)}{|D| \cdot |P|}, \quad (3)$$

where D is the set of training data ($d \in D$), P is the set of output nodes ($p \in P$), $\chi^2_{d,p}$ is the χ^2 of the p^{th} output of the data entry d, $\mu_a^{d,p}$ is a mean of actual distribution of the p^{th} output of the data entry d, $\mu_t^{d,p}$ is a mean of target distribution of the p^{th} output of the data entry d, $|D|$ is the size of training data and $|P|$ is the number of output nodes.

To compute fitness value for a SNN$_i$ in a particular generation, the process is as follows:

1) for each $data_k$ in training data set,
   a. execute a SNN$_i$ with the same input values from $data_k$ t_a times
   b. set $Score = 0$
   c. for each $output_j$:
      i. compute distributions of actual $output_j$ and its actual mean ($\mu_a^{k,j}$)
      ii. use target $output_j$ distribution from $data_k$ to compute the target mean ($\mu_t^{k,j}$)
      iii. compute predicted frequency f_s^* in every slot s (Equation 2)
      iv. compute $\chi^2_{k,j}$ (Equation 1)
      v. compute $Score = Score + \chi^2_{k,j} + |\mu_a^{k,j} - \mu_t^{k,j}|$
   d. $Score = Score/|P|$
2) $Fitness_i = Score/|D|$

Then, $Fitness_i$ is assigned to the i^{th} SNN in the current population. These values are used in a part of ranking process for generating the new population of SNNs in the next generation.

3. Modeling a Beta Distribution

A beta distribution is a versatile distribution that has been commonly used for modeling data with uncertainty in many applications [2]. The probability density function of the beta distribution can be calculated with the equation :[5]

$$f(x; \alpha, \beta) = \frac{1}{Beta(\alpha, \beta)} x^{\alpha-1}(1-x)^{\beta-1}, 0 < x < 1, \quad (4)$$

where $\alpha > 0$, $\beta > 0$ and $Beta(\alpha, \beta)$ denote the beta function defined by

$$Beta(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1} dt.$$

The key feature of a beta distribution is that different density function shapes can be obtained by changing α and β. For instance, uniform distribution can be obtained when $\alpha = 1$ and $\beta = 1$ and when $\alpha = 1/2$ and $\beta = 1/2$, a U-shape distribution called arc-sine distribution is produced [2]. When $\alpha = \beta > 1$, the distribution generates a symmetric normal distribution. When $\alpha \neq \beta$, $\alpha > 1$ and $\beta > 1$, an asymmetric distribution is generated.

3.1 Experiments

To model the beta distribution, we used multilayer feed-forward neural networks as shown in Figure 2. The network topology consists of five nodes in the input layer (two input nodes representing α and β parameters, and three stochastic signal nodes), sixteen nodes in the first hidden layer, eight nodes in the second hidden layer, and one node in the output layer. The ranges of α and β are limited to $0 < \alpha \leq 10$ and $0 < \beta \leq 10$. All stochastic signals are uniformly distributed between 0.0 and 1.0. All nodes are sigmoid units.

To create training sets, data is collected by using Equation 4. The data collection process is described as follow.

1) First, to get a target distribution of x, we divide variable x into 11 slots, where $x = 0.0, x = 0.1, x = 0.2, \cdots, x = 1.0$.

2) Values of α and β are randomly generated.

3) Using Equation 4, the values of α and β are used with values of x in all 11 slots to get its target distribution.

4) If not done, go to step 2.

In this experiment, the training set consists of 20 data records, which are randomly generated. The training set will be changed to a new set in every generation. The learning process terminates when the best individual achieves less than 10 % error.

After running about 2000 generations (approximately 20 hours, using PC-Intel Celeron 1.4GHz, 480MB), the results are shown in Figure 3. The target distributions are shown in solid black bars and actual distributions are shown in white bars. These plots are generated from one SNN by changing values of α and β. Each plot is a result of executing the network 110 times and counting frequencies of 11 slots of the output node x. It is clear SNN can handle symmetric normal distributions and asymmetric distributions well (when $\alpha > 1$ and $\beta > 1$). Uniform distributions (when $\alpha = \beta = 1$) and U-shape distributions are more difficult. It was found that the learning process adapted quickly to match the symmetric normal distributions and asymmetric distributions, but took much more time to adapt to the more difficult ones.

An second experiment was run to determine the optimal number of additional random inputs that are needed to solve the problem. Without any additional random input, the SNNs learned only to represent the means of the target distributions. Having only one additional random input present the SNN took a very long time to reach the error threshold whereas having two or five present the threshold could be met in about 2000 generations (approximately 20 hours) on average. When we compare the time taken in hours to reach a 25 percent error threshold, the results shows that the optimal time was achieved with three additional random inputs. Having too few inputs makes it difficult to convert into complex input-output relationships while having too many means more time

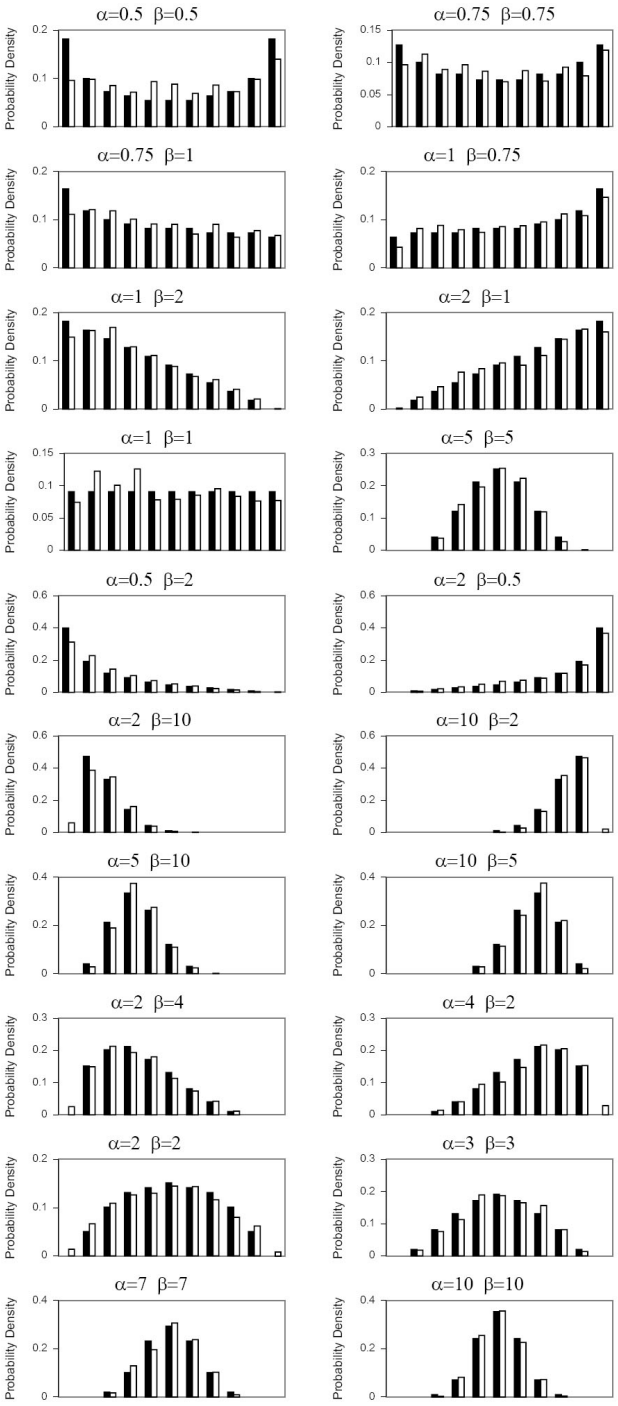

Figure 3. With approximately 10 percent of error, these is a set of 20 plots with different values of α and β produced by one of the best generated SNNs comparing with target distributions. The target and actual distributions are shown in solid black bars and white bars respectively.

is required to adjust more weights. This number must be determined on a per problem basis.

An interesting feature of SNNs is that they can modify the density of possible outputs and still get the same distributions. When a trained SNN was modified its output discretization by doubling its possible output ranges, the results still get the same shape of distribution. The size of distribution is smaller since there are more possible outcomes.

4. Conclusions and Future Work

In this paper, we have presented a preliminary approach to represent uncertainties. We demonstrate the capability of SNNs to represent the model of beta distribution, indicating the potential to represent uncertainty. The techniques presented capture model uncertainty concisely and are simpler and more effective than previous approaches.

One future question is how to obtain greater precision in presenting the complex relationships within such large dynamic models. Another issue is how well results or even modeling approaches may generalize between domains.

References

[1] B. Ayyub and G. Klir. *Uncertainty Modeling And Analysis in Engineering And the Sciences*. Chapman & Hall-CRC, 1st edition, 2006.

[2] F. Cribari-Neto and K. L. P. Vasconcellos. Nearly unbiased maximum likelihood estimation for the beta distribution. *Journal of Statistical Computation and Simulation*, 72(2):107–118, 2002.

[3] P. Eggenberger, A. Ishiguro, S. Tokura, T. Kondo, and Y. Uchikawa. Toward seamless transfer from simulated to real worlds: A dynamically-rearranging neural network approach. In *Proceeding of 1999 the Eighth European Workshop in Learning Robot* , pages 44–60, 1999.

[4] F. N. Kerlinger and H. B. Lee. *Foundations of Behavioral Research*. Harcourt College Publishers, 4th edition, 2000.

[5] S. Nadarajah and A. K. Gupta. Characterizations of the beta distribution. *Communications in Statistics - Theory and Methods*, 33(12):2941 – 2957, 2004.

[6] T. Nobori and N. Matsui. Stochastic resonance neural network and its performance. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks* , volume 2, pages 13–17, July 2000.

[7] L. I. Perlovsky. *Neural Networks and Intellect: Using Model-Based Concepts*. Oxford University Press, 2001.

[8] T. Tian and K. Burrage. Stochastic neural network models for gene regulatory networks. In *The 2003 Congress on Evolutionary Computation - CEC '03*, volume 1, pages 162–169, December 2003.

[9] C. Turchetti, M. Conti, P. Crippa, and S. Orcioni. On the approximation of stochastic processes by approximate identity neural networks. *IEEE Transactions on Neural Networks*, 9(6):1069–1085, 1998.