World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:3, 2008

Evolutionary Training of Hybrid Systems of Recurrent Neural Networks and Hidden Markov Models

Rohitash Chandra, and Christian W. Omlin

***Abstract*—**We present a hybrid architecture of recurrent neural networks (RNNs) inspired by hidden Markov models (HMMs). We train the hybrid architecture using genetic algorithms to learn and represent dynamical systems. We train the hybrid architecture on a set of deterministic finite-state automata strings and observe the generalization performance of the hybrid architecture when presented with a new set of strings which were not present in the training data set. In this way, we show that the hybrid system of HMM and RNN can learn and represent deterministic finite-state automata. We ran experiments with different sets of population sizes in the genetic algorithm; we also ran experiments to find out which weight initializations were best for training the hybrid architecture. The results show that the hybrid architecture of recurrent neural networks inspired by hidden Markov models can train and represent dynamical systems. The best training and generalization performance is achieved when the hybrid architecture is initialized with random real weight values of range -15 to 15.

***Keywords*—**Deterministic finite-state automata, genetic algorithm, hidden Markov models, hybrid systems and recurrent neural networks.

I. Introduction

RECURRENT neural networks have been an important focus of research as they can be applied to difficult problems involving time-varying patterns. Their applications range from speech recognition and financial prediction to gesture recognition [1]-[3]. They have the ability to provide good generalization performance on unseen data but are difficult to train. Hidden Markov models, on the other hand, have also been applied to solve difficult real world problems [4],[5]; for decades, they have been very popular in areas of speech recognition [6]. Training hidden Markov models is easy but their generalization performance may not perform satisfactorily when compared to the performance of recurrent neural networks.

The structural similarities between hidden Markov models and recurrent neural networks is the basis for mapping HMMs into RNNs. The recurrence equation in the recurrent neural network resembles the equation in the *forward* algorithm in the hidden Markov models [7]. The combination of the two paradigms into a hybrid system may provide better generalization and training performance which would be a useful contribution to the field of machine learning and pattern recognition ; in this paper, however we are only going to show that the hybrid system of RNN/HMM can learn and represent deterministic finite-state automata. We will show that the hybrid system may obtain better generalization or training performance in future research studies.

Evolutionary training methods like genetic algorithms have been popular for training neural networks other than gradient decent learning [8]. It has been observed that genetic algorithms overcome the problem of local minima whereas in gradient descent search for the optimal solution, it may be difficult to drive the network out of the local minima which in turn prove costly in terms of training time. Evolutionary neural learning has been successfully applied to many real world problems (e.g. [9]). In the neural network training process, genetic algorithms are used to optimize the weights which represent the knowledge learnt in the training process. Usually, the crossover and the mutation operators are altered in genetic algorithms to represent real numbered weight values of the network. In this paper, we are going to show how genetic algorithms can be applied to training hybrid systems of hidden Markov models and recurrent neural networks.

Recurrent neural networks are dynamical systems and it has shown been that they can represent deterministic finite-state automata in their internal weight representations [10]. In this paper we are also going to show that deterministic finite-state automata can be also be trained and represented by hybrid system of RNN / HMM.

II. Definitions and Methods

A. Recurrent Neural Networks

Recurrent neural networks contain feedback connections. They are composed of an input layer, a context layer which provides state information, a hidden layer and an output layer.

Manuscript received August 31st 2006. This work was supported in part while the first author was a student at the University of the South Pacific. The authors thank the Faculty of Science and Technology of the University of the South Pacific for funding the research for this paper. The first author also thanks the University of Fiji for showing their interest in the research work.

R. Chandra is currently a postgraduate research student working in the field of machine learning and pattern recognition at the School of Science and Technology, The University of Fiji, Lautoka, Fiji Islands (phone: 00679-9307666; e-mail: rohitashc@unifiji.ac.fj).

C. W. Omlin is a professor in computing science at the School of Computing, Information and Mathematical Science, University of the South Pacific, Suva, Fiji Islands (e-mail: omlin_c@usp.ac.fj).

Each layer contains one or more processing units called neurons which propagate information from one layer to the next by computing a non-linear function of their weighted sum of inputs. Recurrent neural networks maintain information about their past states for the computation of future states and outputs. They are nonlinear dynamical systems and it has been previously shown that RNN's can represent DFA states [10]. Popular architectures of recurrent neural networks include first-order recurrent networks [11], second-order recurrent networks [12], NARX networks [13] and LSTM recurrent networks [14]. A detailed study about the vast variety of recurrent neural networks is beyond the scope of this paper. We will map hidden Markov models into first – order recurrent neural networks and show that the hybrid architecture can learn deterministic finite-state automata. Fig. 1 is a diagram for first order recurrent neural networks showing the recurrence from the hidden to the context layer.

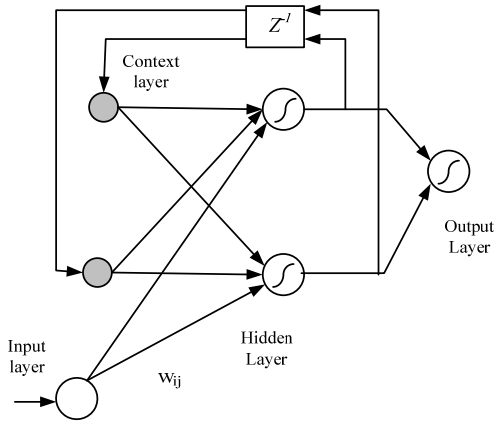

Fig. 1 Recurrent neural network architecture

B. Hidden Markov Models

In a first order Markov model, the state at time *t+1* depends only on state at time *t*, regardless of the states in the previous times [15]. Fig. 2 shows an example of a Markov model containing three states in a stochastic automaton. Π_i is the probability that the system will start in state S_i and a_{ij} is the probability that the system will move from state S_i to state S_j.

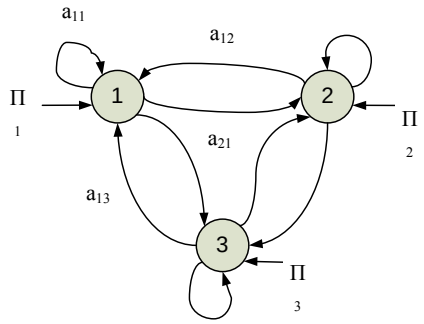

Fig. 2 A Markov model

A hidden Markov model (HMM) describes a process which goes through a finite number of non-observable states whilst generating a signal of either discrete or continuous in nature. The model probabilistically links the observed signal to the state transitions in the system. The theory provides a means by which:

- the probability $P(O|\lambda)$ can be calculated for a HMM with parameter set λ, generating a particular observation sequence O, through what is called the *Forward* algorithm.
- the most likely state sequence the system went through in generating the observed signal through the *Viterbi* algorithm.
- a set of re-estimation for iteratively updating the HMM parameters given an observation sequence as training data. These formulas strive to maximize the probability of the sequence being generated by the model. The algorithm is known as the *Baum-Welch* of *Forward- backward* procedure.

The term "hidden" hints at the process' state transition sequence which is hidden from the observer. The process reveals itself to the observer only through the generated observable signal. A HMM is parameterized through a matrix of transition probabilities between states and output probability distributions for observed signal frames given the internal process state. The probabilities are used in the mentioned algorithm for achieving the desired results.

C. Finite-State Automata as Test Beds for Training

A finite-state automaton is a device that can be in one of a finite number of *states*. In certain conditions, it can switch to another state; this is called a *transition*. When the automaton starts processing input, it can be in one of its initial states. There is also another important subset of states of the automaton: the final states. If the automaton is in a final state after processing an input sequence, it is said to *accept* its input. Finite-state automata are used as test beds for training recurrent neural networks. Presumably, strings used for training do not need to undergo any feature extraction. They are used to show that recurrent neural networks can represent dynamical systems.

D. Deterministic Finite-State Automata

A language is a set of strings over a finite alphabet $\sum = \{\sigma_1, \sigma_2, ..., \sigma_{|\Sigma|}\}$. The length of a string ω will be denoted $|\omega|$. A deterministic finite-state automata (DFA) is defined as a 5-tuple $M = (Q, \sum, \delta, q_1, F)$, where Q is a finite number of states, $\sum$ is the input alphabet, δ is the next state function $\delta : Q \times \sum \rightarrow Q$ which defines which state q' = $\delta(q,\sigma)$ is reached by an automaton after reading symbol σ when in state q, $q_1 \in Q$ is the initial state of the automaton (before reading any string) and $F \subseteq Q$ is the set of accepting states of

the automaton. The language *L(M)* accepted by the automaton contains all the strings that bring the automaton to an accepting state. The languages accepted by DFAs are called regular languages. Fig. 3 shows the DFA which will be used for training the hybrid system of RNN and HMM. State 1 is the automaton's start state; accepting states are drawn with double circles.

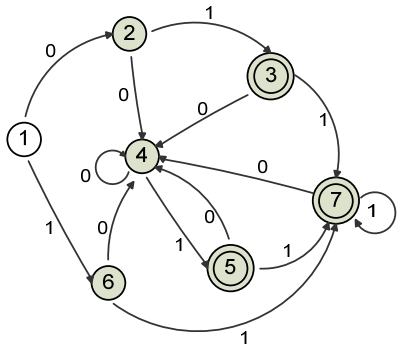

Fig. 3 Deterministic finite-state automata

E. Evolutionary Training of Recurrent Neural Networks

Recurrent neural networks have been trained by evolutionary computation methods such as genetic algorithms which optimize the weights in the network architecture for a particular problem. Compared to gradient descent learning, genetic algorithms can help the network to escape from the local minima. Genetic algorithms rely on the reproduction heuristic of *crossover* operator which forms offspring's by recombining representational components of two members of the population without regard to their content [16]. This approach of creating population members assumes that components of all parent representations may be freely exchanged without altering the search process. Usually, the crossover and mutation operators are altered in genetic algorithm for training neural networks; this is done to represent the real number values of the weights in the network.

F. Mapping Hidden Markov Models into Recurrent Neural Networks

As stated earlier, the structural similarities of hidden Markov models and recurrent neural networks form the basis for combining the two paradigms in a hybrid architecture. Why is it a good idea? Most often, first order HMMs are deployed in practice which means that state transition probabilities are dependent only on the previous state. This assumption is unrealistic for many real world applications of HMMs. It has been shown that RNNs can learn higher-order dependencies from training data [17]. Furthermore, the number of states in the HMM needs to be fixed beforehand for a particular application. In the past, artificial neural networks have been pruned or extended during training to achieve higher discriminative and training performance [18]. The theory on RNNs and HMMs suggest that the combination of the two paradigms may provide better generalization and training performance. The hybrid system may also have the capability of learning higher order dependencies. In the hybrid system, there may not be any requirements of fixing the number of states for HMM prior to training in a real world application. Next we will study the structural similarities of the two paradigms and design a hybrid architecture.

Consider the equation of the forward algorithm for the calculation of $P(O|\lambda)$ in "equation (1)".

$$\alpha_j^t = \left(\sum_i^N \alpha_i^{t-1} a_{ij} \right) b_j\left(O^t\right) \quad 1 \le j \le N \tag{1}$$

where N is the number of hidden states in the HMM, a_{ij} is the probability of making a transition from state i to j and $b_j\left(O^t\right)$ is the probability of generating symbol O^t when in state i. The calculation in "equation (2)" is inherently recurrent and bares resemblance to the recursion of RNN shown in Fig. 1.

$$x_j^t = f\left(\sum_i^N x_i^{t-1} w_{ij} \right) \quad 1 \le i \le N \tag{2}$$

where f() is a non-linearity as sigmoid, N the number of hidden neurons and w_{ij} the weights connecting the neurons with each other and with the input nodes.

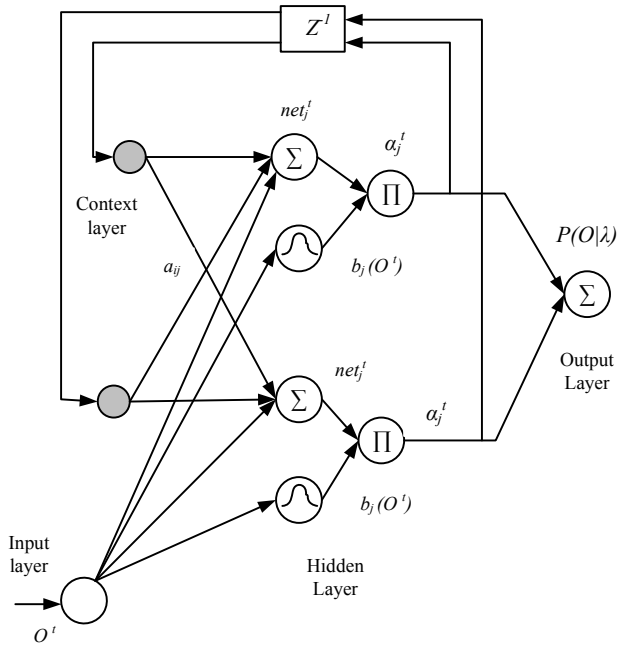

Fig. 4 The hybrid recurrent neural network architecture inspired by hidden Markov models

Fig. 4 shows how HMM is mapped into the RNN by tying the output of the HMM, $P(O|\lambda)$ together by means of a trainable weight leading to an output layer.

The output of the Gaussian function solely depends on the two input parameters which are the mean and the variance. These are parameters that observe the sequence of the input data in the hybrid architecture which may be DFA strings or data from any real-world time series. These parameters will also be represented in the chromosomes together with the weights and biases and will be trained by genetic algorithm. We used a univariate Gaussian for one dimensional input of DFA training strings. For real world applications where multiple dimensions are involved, multivariate Gaussian function would be used instead. The univariate Gaussian function used in this hybrid architecture is given by "equation (3)".

$$b_i\left(O^t\right) = \frac{1}{\sqrt{2\pi}\,\sigma_i} \exp\left[-\frac{1}{2}\,\frac{\left(O^t - \mu_i\right)^2}{\sigma_i^{\,2}}\right] \quad (3)$$

where $b_i\left(O^t\right)$ is the Gaussian function, O^t is the observation at time t, μ is the mean and σ^2_i is the variance. The observation sequences for training a DFA are the DFA strings which are the inputs to the hybrid architecture.

G. Evolutionary Training of the Hybrid System

In the hybrid system of hidden Markov models and recurrent neural networks, the neurons in the hidden layer compute the weighted sum of their inputs only without further computing the nonlinear sigmoidal function of the output. The outputs of the corresponding neurons in the hidden layer are further multiplied with the output of the corresponding Gaussian function. The product of the neuron and the output of the Gaussian function are then propagated from the hidden layer to the output layer as shown in Fig. 4.

We altered the crossover and mutation operators in genetic algorithms so that the genes could represent real numbered weight values in the hybrid architecture. Prior to the training process, a population size is defined and then the algorithm randomly chooses two parent chromosomes; they are combined into a child chromosome using the crossover operator. The child chromosome is further mutated according to a mutation probability. The mutation operator adds a small real random number to a random gene in the child chromosome. The child chromosome then becomes part of the new generation. A chromosome represents the weights, biases, mean and variance as parameters of the hybrid system of HMM and RNN. The fitness function computes the reciprocal of the squared error for each chromosome; thus, genetic algorithm is used for reducing the squared error. The chromosome with the least squared error from the hybrid architecture then slowly begins to affect the entire population until a solution is reached. Evolutionary computation such as genetic algorithm thus finds the best chromosomes representing the weights, biases and other parameters of the hybrid system.

H. Experimentation

We generated a training set of strings from the DFA shown in Fig. 2 consisting of all strings of lengths 1-10 and a testing set of 1-15. We ran experiments of the evolutionary processes of training genetic algorithms with population sizes of 60, 80 and 100 chromosomes. We used the crossover operator probability as 0.7 and mutation probability as 0.01. We used different weight initialization ranges to observe which weight initialization ranges are best for training the hybrid architecture. We used a training bound of 50 generations; if the hybrid system could be trained within 50 generations, we stopped. We ran two major experiments as follows:

Experiment 1: We trained all the parameters of the hybrid architecture, i.e. the weights connecting input to hidden layer, weights connecting hidden to output layer and weights connecting the context to hidden layer. We also trained the bias weights and the mean and the variance as parameters of the univariate Gaussian function along the hidden layer. The network topology used for this experiment was as follows: we used one neuron in the input layer for string input, 5 neurons in the context and hidden layer and one output neuron in the output layer.

Experiment 2: We trained only the weights connecting the context layer to the hidden layer and used a constant value of 1 for the weights that connected the input layer to the hidden layer and those weights that connected the hidden to the output layer. We did not train the latter weights to investigate if deterministic finite-state automata can be represented by the context and hidden weight layers alone. We trained the mean and variance as parameters of the univariate Gaussian function in the hidden layer. The network topology used for this experiment is as follows: we used one neuron in the input layer for string input, 10 neurons in the context and hidden layer and one output neuron in the output layer. The results of these experiments are shown in the following section.

III. Results and Discussion

The results for both experiments show that evolutionary training for hybrid system of RNN and HMM can be difficult if the weights, biases, and parameters of the Gaussian are initialized with random real number values from -1 to 1. Both experiments reveal that deterministic finite-state can be learned and be represented by the hybrid system of HMM and RNN. Table I shows the results for *experiment 1*, where all parameters in the hybrid system are trained.

TABLE I
RESULTS FOR EXPERIMENT 1

Population size	Weight range	Training performance	Generalization performance	training time
60	-1 to 1	0.05%	0.05%	max
80	-1 to 1	0.05%	0.05%	max
100	-1 to 1	0.05%	0.05%	max
60	-3 to 3	100%	100%	3
80	-3 to 3	100%	100%	2
100	-3 to 3	100%	100%	2
60	-15to 15	100%	100%	2
80	-15to 15	100%	100%	2
100	-15to 15	100%	100%	2
60	-30 to 30	100%	100%	2
80	-30 to 30	100%	100%	2
100	-30 to 30	100%	100%	2

The training and generalization performance show the percentage of strings correctly classified by the hybrid system of HMM and RNN. The training time is given by 'generations'. The maximum training time used was 50 generations which is denoted by the word 'max' in the table.

It can be seen from Table I that deterministic finite-state automata can be trained and represented by the hybrid architecture; the results show 100% training and generalization performance. There is no significant difference for different population sizes. The best weight value initializations for faster training performance are real weight values within a range of -15 to 15. Other weight initializations also show satisfactorily performance except for small random values for weight initializations.

TABLE II
RESULTS FOR EXPERIMENT 2

Population size	Weight range	Training performance	Generalization performance	training time
60	-1 to 1	0.05%	0.05%	max
80	-1 to 1	0.05%	0.05%	max
100	-1 to 1	0.05%	0.05%	max
60	-3 to 3	100%	100%	2
80	-3 to 3	100%	100%	5
100	-3 to 3	100%	100%	2
60	-15to 15	100%	100%	2
80	-15to 15	100%	100%	2
100	-15to 15	100%	100%	2
60	-30 to 30	100%	100%	3
80	-30 to 30	100%	100%	2
100	-30 to 30	100%	100%	8

The training and generalization performance show the percentage of strings correctly classified by the hybrid system of HMM and RNN. The training time is given by 'generations'. The maximum training time used was 50 generations which is denoted by the word 'max' in the table.

Table II shows the results for *experiment 2* which reveal that deterministic finite-state can be represented by only training the context weights and Gaussian parameters of the hybrid architecture. The experiment shows a 100% training and generalization performance. The system does not perform satisfactorily when initialized with small random weights.

IV. CONCLUSION

We have successfully mapped hidden Markov models into recurrent neural networks. The structural similarities between hidden Markov models and recurrent neural networks have been the basis for the successful mapping in the hybrid architecture. We have used genetic algorithms to train the hybrid system of hidden Markov models and recurrent neural networks. We altered the crossover and mutation operators in genetic algorithms to represent real numbered weight values of the hybrid architecture. We used genetic algorithms to train the parameters in the hybrid architecture which were the weights, biases, the mean and the variance of the univariate Gaussian function. We used deterministic finite state automata for training to show that the hybrid architecture can represent dynamical systems. We ran two major experiments which show that deterministic finite automata can be trained and represented by hybrid systems of HMM and RNN. Both experiments had a difficulty in training when initialized with small random weight values. It is shown that the hybrid architecture can train and represent dynamical systems with one dimensional input. In the future, we will investigate if successful training can be done for real world application problems where the data sets are multi-dimensional.

REFERENCES

[1] A.J Robinson, An application of recurrent nets to phone probability estimation, *IEEE transactions on Neural Networks,* vol.5, no.2 , 1994, pp. 298-305.

[2] C.L. Giles, S. Lawrence and A.C. Tsoi, Rule inference for financial prediction using recurrent neural networks, *Proc. of the IEEE/IAFE Computational Intelligence for Financial Engineering*, New York City, USA, 1997, pp. 253-259

[3] K. Marakami and H Taguchi, Gesture recognition using recurrent neural networks*, Proc. of the SIGCHI conference on Human factors in computing systems: Reaching through technology,* Louisiana, USA, 1991, pp. 237-242.

[4] T. Kobayashi, S. Haruyama, Partly-Hidden Markov Model and its Application to Gesture Recognition*, Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing* , vol. 4, 1997, pp.3081.

[5] P.A Stoll, J. Ohya, Applications of HMM modeling to recognizing human gestures in image sequences for a man-machine interface, *Proc. of the 4th IEEE International Workshop on Robot and Human Communication*, Tokyo, 1995, pp. 129-134.

[6] M. J. F. Gales, Maximum likelihood linear transformations for HMM-based speech recognition, *Computer Speech and Language*, vol. 12, 1998, pp. 75-98.

[7] T. Wessels, C.W. Omlin, Refining Hidden Markov Models with Recurrent Neural Networks, *Proc. of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, vol. 2, 2000, pp. 2271.

[8] Kim Wing C. Ku, Man Wai Mak, and Wan Chi Siu. Adding learning to cellular genetic algorithms for training recurrent neural networks. *IEEE Transactions on Neural Networks*, vol. 10, no.2, 1999, pp. 239-252.

[9] Abbass Hussein, An evolutionary artificial neural network approach for breast cancer diagnosis. *Artificial Intelligence in Medicine*, vol. 25, no. 3, 2002, pp.265-281.

[10] Lee Giles, C.W Omlin and K. Thornber, Equivalence in Knowledge Representation: Automata, Recurrent Neural Networks, and dynamical Systems, *Proc. of the IEEE*, vol. 87, no. 9, 1999, pp.1623-1640

[11] P. Manolios and R. Fanelli, First order recurrent neural networks and deterministic finite state automata. *Neural Computation*, vol. 6, no. 6, 1994, pp.1154-1172.

[12] R. L. Watrous and G. M. Kuhn, Induction of finite-state languages using second-order recurrent networks, *Proc. of Advances in Neural Information Systems*, California, USA, 1992, pp. 309-316.
[13] T. Lin, B.G. Horne, P. Tino, & C.L. Giles, Learning long-term dependencies in NARX recurrent neural networks. *IEEE Transactions on Neural Networks,* vol. 7, no. 6, 1996, pp. 1329-1338.
[14] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Computation,* vol. 9, no. 8, 1997, pp. 1735-1780.
[15] E. Alpaydin, *Introduction to Machine Learning*, The MIT Press, London, 2004, pp. 306-311.
[16] P. J. Angeline, G. M. Sauders, and J. B. Pollack, An evolutionary algorithm that constructs recurrent neural networks, *IEEE Trans. Neural Networks*, vol. 5, 1994, pp. 54-65.
[17] Y. Bengio, *Neural Networks for Speech and Sequence Recognition*. London UK, International Thompson Computer Press, 1996.
[18] Y. LeCun, J. Denker and S. Solla, Optimal Brain Damage, *Advances in Neural Information Processing Systems 2*, Morgan Kaufman Publishers, San Mateo, CA, 1990.