World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:9, No:11, 2015

A Hybrid System of Hidden Markov Models and Recurrent Neural Networks for Learning Deterministic Finite State Automata

Pavan K. Rallabandi, Kailash C. Patidar

Abstract—In this paper, we present an optimization technique or a learning algorithm using the hybrid architecture by combining the most popular sequence recognition models such as Recurrent Neural Networks (RNNs) and Hidden Markov models (HMMs). In order to improve the sequence/pattern recognition/classification performance by applying a hybrid/neural symbolic approach, a gradient descent learning algorithm is developed using the Real Time Recurrent Learning of Recurrent Neural Network for processing the knowledge represented in trained Hidden Markov Models. The developed hybrid algorithm is implemented on automata theory as a sample test beds and the performance of the designed algorithm is demonstrated and evaluated on learning the deterministic finite state automata.

Keywords—Hybrid systems, Hidden Markov Models, Recurrent neural networks, Deterministic finite state automata.

I. Introduction

SEQUENCE recognition is a major step in many applications ranging from speech recognition, signature verification, time series modelling and prediction to bioinformatics. Hidden Markov models (HMMs) are one of the most popular techniques for sequence modelling and classification because they are easy to train. However, HMMs generally do not perform satisfactorily on difficult recognition problems. Recurrent neural networks (RNNs) are alternative methods for modelling sequences and they have excellent generalization performance, but training RNNs can be very difficult.

Previously, it has been shown by several researchers [14], [19], [25] that recurrent neural networks are excellent tools for precessing existing domain theories with hidden states through learning and that learned knowledge can be extracted in symbolic form. Recently, there has been a lot of interest in combining symbolic and neural learning. There are different ways in which neural and symbolic learning can be combined to solve a given learning task. It is, however, difficult to interpret the knowledge stored in neural networks.

A. Hybrid Systems: Recurrent Neural Networks Based on Hidden Markov Models

Recurrent neural networks (RNNs) 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 [23]. They have the ability to provide good generalization performance on unseen data but are difficult to train. Hidden Markov models (HMMs), on the other hand, have also been applied to solve difficult real world problems involving time-varying patterns. For instance, they have been very popular in areas of speech recognition [11]. Training HMMs is easy, i.e., they learn faster when compared to recurrent neural network, but their generalization performance may not perform satisfactorily when compared to the performance of recurrent neural networks.

The structural similarities between HMMs and RNNs are the basis for mapping HMMs into RNNs. The recurrence equation in the recurrent neural network resembles the equation in the forward algorithm in the HMMs. 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. We call the new hybrid architecture as hybrid HMM-RNN in further discussions.

B. Significance of HYBRID HMM-RNN

The structural similarities of hidden Markov models and recurrent neural networks form the basis for combining the two paradigms into a hybrid architecture. Why is it a good idea? Most often, first-order HMMs are used in practice in which successor states 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 [4]. Furthermore, the number of states in the HMMs needs to be fixed beforehand for a particular application. However, the numbers of states for different applications vary. The theory on RNNs and HMMs suggest that the combination of the two paradigms may provide better generalization and training performance. Our proposed architecture of hybrid recurrent neural networks may also have the capability of learning higher order dependencies and one does not need to fix the number of states as in the case of HMMs.

In this paper, we designed a new architecture by combining the structural similarities and the first order equations of Hidden Markov Models and Recurrent Neural Networks and developed a gradient descent algorithm using real time recurrent learning algorithm along with the mathematical proof

Pavan K. Rallabandi is with the Department of Computer Science, University of the Western Cape, Bellville, Private Bag X17, 7535 RSA (e-mail: prallabandi@uwc.ac.za).

Kailash C. Patidar is with the Department of Mathematics and Applied Mathematics, University of the Western Cape, Bellville, Private Bag X17, 7535 RSA (e-mail: kpatidar@uwc.ac.za).

and implemented on automata theory as a sample testbeds and a protein classification problem of bioinformatics application.

The rest of this paper is organised as follows. In section II, we explained the models, methods and frameworks used in the developed model. In Section III, the mathematical proof of the proposed hybrid architecture has been presented. In Section IV, the gradient descent algorithm has been explained using the real time recurrent learning of RNN. In Section V, the derived hybrid HMM-RNN algorithm is implemented to learn a deterministic finite state automata and presented the results. Finally, Section VI concludes the paper with the scope for further research directions.

II. Methods and Models

Neural networks and Hidden Markov models are mostly used in the process of sequence recognition or pattern recognition/classification problems. In this paper, we designed a new architecture by taking the advantages and disadvantages of both the models and used the advanced recurrent neural networks as a base model to develop the knowledge based system. The brief overview of the used methods, models and frameworks are described below in the process of building a hybrid HMM-RNN system.

A. Neural Networks

Artificial neural networks are loosely modelled after biological neural systems. They learn by training from past experience data and make generalization on unseen data.

Neural networks have been applied to many real world problems such as speech recognition [4], bio-conservation [6], gesture recognition [20], medical diagnostics [2]. Neural networks learn by training on past experience using an algorithm which modifies the interconnection weights as directed by a learning objective for a particular application.

In this learning process, the error function must be minimized. This minimization has to be done with respect to the weights and bias. If a network has differential activation functions, then the input variable of the differentiable functions are the activations of the output units. These input variable are the weights and bias. One can evaluate the derivative of the error with respect to weights. These derivatives can then be used to find the weights that minimize the error function, by using the popular gradient descent method. For further details on this, as well as on the gradient descent algorithm, one may refer to [24].

Gradient descent optimization method is one of the most popularly used back-propagation learning algorithm. It has been proved as a very successful method in many applications. However, this method does not converge very fast. Moreover, the convergence to the global minimum is not always guaranteed. Many researchers[5], [10] have attempted for improvements to the standard gradient descent method, such as dynamically modifying learning parameters or adjusting the steepness of the sigmoid function. To this end, gradient methods using second-derivatives (Hessian matrix), such as Newton's method, are found to be very efficient under certain conditions [22].

In contrast to feed-forward networks, recurrent networks are dynamical systems whose output depends on the present state of the units in the network; learning in feedback networks corresponds to function approximation [17]. Examples of recurrent networks are the *Hopfield net* [16] with symmetric fully connected feedback neurons with discrete states and hard limiting activation function, and the *Boltzmann machine* [15] which is a stochastic version of the discrete-time Hopfield net with transfer function f_T.

B. Recurrent Neural Networks

Recurrent networks are computational systems with context-varying responses, i.e., they have the ability to model two types of time-dependent behaviour: The first deals with gradually settling into a solution for a complex set of conflicting constraints such as pattern completion, whereas the second concerns the modelling of pattern sequences. Recurrent networks have been successfully applied to problems ranging from formal grammars, speech and image recognition to time series prediction [8], [12].

Recurrent networks generally use some delayed output to calculate the current activation. Some represent this delayed copy explicitly by means of context units, for example Jordan and Elman networks, whereas others employ an implicit signal delay mechanism [18]. Examples of the latter are Hopfield networks, Boltzmann machines and second-order networks [13].

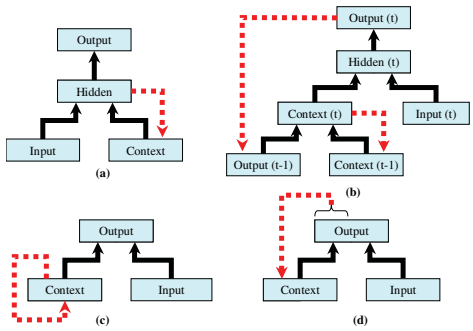

Fig. 1 Architectures of Recurrent Neural Networks

Fig. 1 represents recurrent networks architectures of Elman (a), Jordan (b), Robinson and Fallside (c), and Williams and Zipser (d). Dashed lines indicate feedback connections.

C. Training Methods

1) Backpropagation through Time: Backpropagation is the most widely applied learning algorithm for both feed-forward and recurrent neural networks. Backpropagation employs gradient descent to minimize the squared error between the networks output values and desired values for those outputs. The learning problem faced by backpropagation is to search a large hypothesis space defined by weight values for all the units of the network. Error is propagated from the output layer back to the hidden layers from which the weights are updated.

Backpropagation is used for training feed-forward networks while backpropagation through time (BPTT) is employed for

training recurrent neural networks [27]. The BPTT is the extension of backpropagation algorithm. The general idea behind BPTT is to unfold the recurrent neural network in time so that it becomes a deep multilayer feed-forward network. This can be done by adding a layer for each time step. When unfolded in time, the network has the same behaviour as a recurrent neural network for a finite number of time steps.

2) Real Time Recurrent Learning: Backpropagation-through-time uses the backward propagation of error information to compute the error gradient used in the weight update. An alternative approach for computing the gradient is to propagate the error gradient information forward. Real-time recurrent learning (RTRL) is a real time learning algorithm which updates the weights at the end of each sample string presentation with a gradient descent weight update rule. The algorithm computes the derivatives of states and outputs with respect to all weights as the network processes the sequence during the forward step [29]. There is no unfolding performed or necessary for real time recurrent learning.

In RTRL, the weights can be incremented on-line or at the end of the whole input sequence. Because on-line updating is possible, the RTRL algorithm can deal with input sequences of arbitrary length and does not require memory proportional to the length of input sequence. It allows recurrent networks to learn tasks that require retention of information over time periods having either fixed or indefinite length.

D. Hidden Markov Models

As mentioned in [21], Hidden Markov model (HMM) describes a process, which goes through a finite number of states whilst generating a signal of either discrete or continuous nature. The model probabilistically links the observed signal to the state transitions in the system.

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. These probabilities are used in the algorithms that are used for achieving the desired results. A typical way to combine HMMs and neural networks is to replace the Gaussian density function estimates of emission probabilities by neural networks.

Hidden Markov Models (HMM) are stochastic methods to model temporal and sequence data. They are especially known for their application in temporal pattern recognition such as speech, handwriting, gesture recognition, part-of-speech tagging and bioinformatics(Modelling of Protein domains, probabilistic sequence alignment, DNA binding site modelling and gene finding).

In a regular Markov model, the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a hidden Markov model, the state is not directly visible, but output, dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. Therefore the sequence of tokens generated by an HMM gives some information about the sequence of states.

Notation: Traditionally, HMMs have been defined by the following quintuple where, N is the number of states for the model M is the number of distinct observations symbols per state, i.e. the discrete alphabet size. A is the NxN state transition probability distribution given in the form of a matrix A = aij B is the NxM observation symbol probability distribution given in the form of a matrix B = bj(k) p is the initial state distribution vector p = pi. Note that, if we opt out the structure parameters M and N we have the more often used compact notation: lambda= (A,B,p)

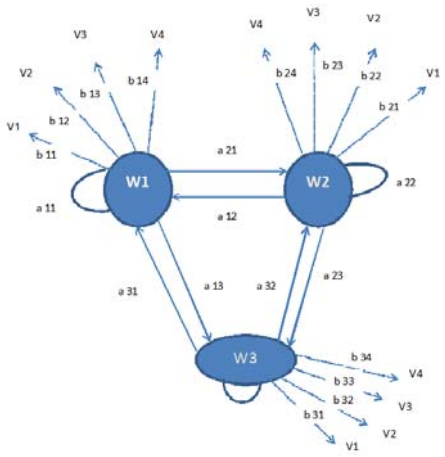

Fig. 2 Hidden Markov Model Architecture

E. Hybrid Systems

Hybrid systems combine strengths of at least two intelligent system paradigms. Examples of hybrid systems include, symbolic connectionist learning, evolutionary neural learning, neural expert systems, neuro-fuzzy systems.

F. Symbolic Connectionist Learning Framework

The general paradigm of symbolic connectionist learning includes the combination of symbolic knowledge in neural networks for better training and generalization performance [1], [9]. In connectionist representation, the approach of using neural networks includes initializing of neural network with small random values and training it using some optimization methods such as gradient descent and genetic algorithms on some known data to perform a certain task. During the entire process, the knowledge remains hidden in the networks adaptable connections, hence the name connectionist representation. The connectionist representation is shown in the Fig.3.

The paradigm in the connectionist representation can be enriched with symbolic knowledge by initializing a network with prior knowledge, i.e., the initial domain theory, prior to training. A translation of information from a symbolic into a connectionist representation is required. This is done by programming subset of weights in the network prior to training instead of choosing small random values. The programmed weights define a starting point in weight space for a search of a solution during training. Examples of this approach include pre-structuring a network with boolean concepts and imposing

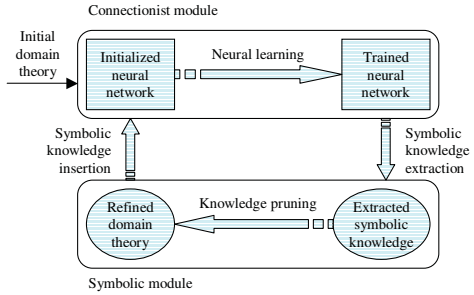

Fig. 3 Knowledge based symbollic connectionist framework

rotation variance in neural networks for image recognition [3], [9], [26].

Fig. 3 represents a new framework for combining symbolic and neural learning. The use of neural networks for knowledge refinement consists of (i) insertion of prior knowledge known as initial domain theory into a neural network, (ii) refinement of knowledge through training a network on examples, and (iii) extraction of learnt knowledge from a trained network in symbolic form, known as refined domain theory.

III. Mathematical Proof of the Designed Architecture

Consider the equation of the forward procedure for the calculation of the probability of the observation O given the model, thus in HMM is given by

$$\alpha_j^t = \left(\left(\sum_i^N \alpha_i^{t-1} a_{ij} \right) \cdot b_j(o^t) \right), \tag{1}$$

where N is the number of hidden states in the HMM, a is the probability of making a transition from state i to j and $b_j(o^t)$ is the Gaussian distribution for the observation at time t. The calculation in (1) is inherently recurrent with resembles to the recursion of recurrent neural networks as shown in (2)

$$x_i^t = f\left(\sum_j^N x_j^{t-1} w_{ji} \right), \tag{2}$$

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

The dynamics of first-order recurrent neural network is given by

$$\mathrm{y}_j(t) = f\left(\sum_{i=1}^N w_{ji} x_i(t) + \sum_{i=1}^M w_{ji} c_i(t) \right). \tag{3}$$

$$\mathrm{c}_i(t+1) = g\left(\sum_j w_{ij} y_j(t) \right), \tag{4}$$

where w_{ji} represent their corresponding weights and $g(\cdot)$ is a sigmoidal discriminant function.

Let us combine (1) with (2) and (4) to form a hybrid architecture. We are replacing the subscript j in $b_j(o^t)$ which denotes the state by time t in hidden Markov models - to incorporate the feature into recurrent neural networks. Hence, the dynamics for the hybrid recurrent neural networks architecture is given by

$$\mathrm{y}_j(t) = \left[f \sum_{i=1}^N w_{ji} x_i(t) + \left[\sum_{i=1}^M w_{ji} c_i(t) \right] b_{t-1}(O) \right], \tag{5}$$

where $\mathrm{b}_{t-1}(O)$ is the Gaussian distribution. Note that the subscript in $\mathrm{b}_{t-1}(O)$, i.e., time t, in (5) is different from the subscript for Gaussian distribution in (1). The dynamics of hidden Markov models and recurrent networks varies in this context; however, we can adjust the parameter for time t as shown in (5) in order to map hidden Markov models into recurrent neural networks. For a single input, the univariate Gaussian distribution is given by

$$\mathrm{b}_t(O) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{1}{2} \frac{(O-\mu)^2}{\sigma^2} \right), \tag{6}$$

where O is the observation at time t, μ is the mean and σ_i^2 is the variance. Similarly, for the discrete inputs, the Gaussian Distribution using histograms is defined as

$$\text{Observation at time } t = \frac{\text{No. of observations occurred at time } t}{\text{Total no. of observations}} \tag{7}$$

Finally, the observation probabilities for discrete case is calculated as frequency of these inputs.

For continuous inputs, we define the Gaussian Mixture Models from [7] which is a parametric probability density function and represented as weighted sum of Gaussian component densities and it is given by:

$$b_j(o_t) = \sum_{m=1}^M C_{jm} \mathcal{N}(o_t; \mu_{jm}, \Sigma_{jm}), \tag{8}$$

where (x, μ, U), denotes a D-dimensional normal density function of mean vector μ and covariance matrix U and M is the number of mixture components for the distribution, and C_{jm}, μ_{jm} and Σ_{jm} are a weight, a L-dimensional mean vector, and a $L\{x\}L$ covariance matrix of mixture component m of state i, respectively.

Mixture weights C_{jm} satisfy the following stochastic constraint $\sum_{m=1}^M C_{jm} = 1$, Hence, $b_i(o)'s$ are normalized as probability density function.

A Gaussian distribution $(o, \mu_{jm}, \Sigma_{jm})$ of each component is defined by

$$\mathrm{b}_t(O) = \frac{1}{2\pi^{d/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(O-\mu)^t \Sigma^{-1} (O-\mu) \right], \tag{9}$$

where O is a d-component column vector, μ is a d-component mean vector, σ is a $d \times d$ covariance matrix, respectively.

We designed the hybrid HMM-RNN architecture based on the structural similarities of Hidden Markov Model and Recurrent Neural Network and presented in Fig. 4 and it shows how the Gaussian distribution for hidden Markov model is mapped into hybrid recurrent neural networks.

Fig. 4 represents the newly developed and designed architecture of a hybrid HMM-RNN. The dashed lined indicates that the architecture can represent more neurons in

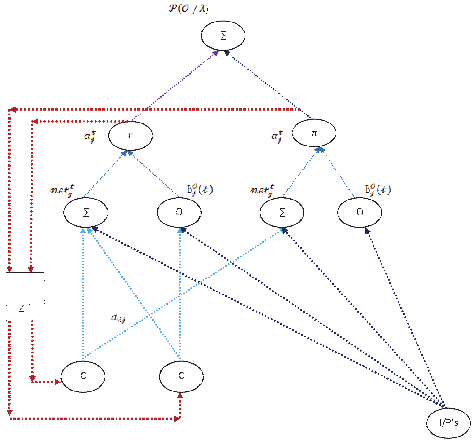

Fig. 4 Hybrid: Hidden Markov Model-Recurrent Neural Network Architecture

hidden and input layer if required. The output of the Gaussian is further multiplied with the output of the neurons in the hidden layer. Note that one Gaussian distribution will be used irrespective of the number of neurons in hidden and input layer. Hence, by combining and representing the forward algorithm of HMM (1) in terms of RNN (2) and taking the derivative with respect to a_{ij}, we get

$$\begin{aligned}\frac{\partial}{\partial a_{ij}}(\alpha_j^t) &= \left[\sum_{i=1}^{N}\frac{\partial}{\partial a_{ij}}\left(\alpha_i^{t-1}a_{ij}\right)b_j(o^t)\right]\\ &= \left[\sum_{i=1}^{N}\frac{\partial}{\partial a_{ij}}\left(\alpha_i^{t-1}a_{ij}\right)\right]b_j(o^t)\\ &= \left[\sum_{i=1}^{N}\frac{\partial}{\partial a_{ij}}\left(\alpha_1^{t-1}a_{1j}+\cdots+\alpha_n^{t-1}a_{nj}\right)\right]b_j(o^t)\end{aligned} \tag{10}$$

Now

$$\frac{\partial}{\partial a_{1j}}(\alpha_1^{t-1}a_{1j}) = \left[\frac{\partial}{\partial a_{1j}}\left(\alpha_1^{t-1}a_{1j}\right)\frac{\partial a_{1j}}{\partial a_{ij}}\right]b_j(o^t),$$

$$\Rightarrow \frac{\partial}{\partial a_{1j}}(\alpha_1^{t-1}a_{1j}) = \begin{cases}\frac{\partial}{\partial a_{1j}}\left(\alpha_1^{t-1}a_{1j}\right)1 & \text{if } i=1\\ 0 & \text{if } i\neq 1,\end{cases}$$

$$\begin{aligned}\Rightarrow \frac{\partial}{\partial a_{1j}}(\alpha_1^{t-1}a_{1j}) &= \frac{\partial}{\partial a_{1j}}(\alpha_1^{t-1}a_{1j})\\ &= \alpha_1^{t-1}\frac{\partial}{\partial a_{1j}}a_{1j}\\ &= \alpha_1^{t-1}.\end{aligned}$$

Similarly,

$$\frac{\partial}{\partial a_{2j}}(\alpha_2^{t-1}a_{2j}) = \alpha_2^{t-1}.$$

Hence

$$\frac{\partial}{\partial a_{ij}}(\alpha_j^t) = \left[\alpha_1^{t-1}+\alpha_2^{t-1}+..+\alpha_N^{t-1}\right]b_j(o^t),$$

$$\begin{aligned}\frac{\partial}{\partial a_{ij}}(\alpha_j^t) &= \left[\sum_{i=1}^{N}\frac{\partial}{\partial a_{ij}}\left(\alpha_i^{t-1}a_{ij}\right)\right]b_j(o^t)\\ &= \left[\sum_{i=1}^{N}\alpha_i^{t-1}\right]b_j(o^t).\end{aligned} \tag{11}$$

In the case of training strings of certain lengths representing finite automaton, a univariate Gaussian for one dimensional input will be used as shown in (6). For real world applications where multiple dimensions are involved, multivariate Gaussian function would be used instead as shown in (8).

IV. Real Time Recurrent Learning for the Hybrid HMM-RNN System

We presented a gradient descent learning algorithm which is based on the Williams and Zipser's real time recurrent learning (RTRL) algorithm [28] of RNN:

In deriving a gradient-based update rule for recurrent networks, we make network connectivity highly unconstrained. We simply suppose that we have a set of input units, $I = x_k(t), 0 < k < m$, and a set of other units, $U = y_k(t),\ 0 < k < n$, which can be hidden or output units. To index an arbitrary unit in the network we can use

$$z_k(t) = \begin{cases}x_k(t)) & \text{if } k \in I,\\ y_k(t)) & \text{if } k \in U.\end{cases} \tag{12}$$

Let W be the weight matrix with n rows and $n+m$ columns, where w_{ij} is the weight to unit i (which is in U) from unit j (which is in I or U). Units compute their activations in the now familiar way, by first computing the weighted sum of their inputs:

$$\text{net}_k(t) = \sum_{I\in U\sqcup I} w_{kI}z_I(t), \tag{13}$$

$$\text{net}_k(t) = \sum_{I\in U\sqcup I} w_{kI}z_I(t)\ b_t(O). \tag{14}$$

Here $b_t(o)$ got two cases:

Case - 1:

$$\text{b}_t(O) = \frac{1}{\sqrt{2\pi\sigma}}\exp\left[-\frac{1}{2}\frac{(O-\mu)^2}{\sigma^2}\right]. \tag{15}$$

Case - 2:

$$\text{b}_t(O) = \frac{1}{2\pi^{d/2}|\Sigma|^{1/2}}\exp\left[-\frac{1}{2}(O-\mu)^t\Sigma^{-1}(O-\mu)\right], \tag{16}$$

where the only new element in the formula is the introduction of the temporal index t. Units then computes some non-linear function of their net input

$$\text{y}_k(t+1) = f_k\left[net_k(t)\right] \tag{17}$$

Usually, both hidden and output units will have non-linear activation functions. Note that external input at time t does not influence the output of any unit until time t+1. The network is thus a discrete dynamical system.

Let $T(t)$ be the set of indices k in U for which there exists a target value $d_k(t)$ at time t. We are forced to use the notation d_k

instead of t here, as t now refers to time. Let the error at the output units be

$$e_k(t) = \begin{cases} d_k(t) - y_k(t) & \text{if } k \in T(t), \\ 0 & \text{otherwise.} \end{cases}$$

and define our error function for a single time step as

$$E(\tau) = -\frac{1}{2} \sum_{k \in U} [e_k(\tau)]^2 .$$

The error function we wish to minimize is the sum of this error over all past steps of the network

$$E_{total}(t_o, t_1) = \sum_{\tau=t_0+1}^{t1} E(\tau).$$

Now, because the total error is the sum of all previous errors and the error at this time step, so also, the gradient of the total error is the sum of the gradient for this time step and the gradient for previous steps

$$\nabla_w E_{total}(t_o, t+1) = \nabla_w E_{total}(t_o, t) + \nabla_w E(t+1).$$

As time series is presented to the network, we can accumulate the values of the gradient, or equivalently, of the weight changes. We thus keep track of the value

$$\triangle\, w_{ij}(t) = -\mu \frac{\partial E(t)}{\partial w_{ij}}.$$

After the network has been presented with the whole series, we alter each weight w_{ij} by

$$\sum_{t=t_0+1}^{t1} \triangle\, w_{ij}(t). \tag{18}$$

We therefore need an algorithm that computes

$$-\frac{\partial E(t)}{\partial w_{ij}} = -\sum_{k \in U} \frac{\partial E(t)}{\partial y_k(t)} \frac{\partial y_k(t)}{\partial w_{ij}} = \sum_{k \in U} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}},$$

at each time step t. Since we know $e_k(t)$ at all times (the difference between our targets and outputs), we only need to find a way to compute the second factor $\frac{\partial y_k(t)}{\partial w_{ij}}$.

Derivation of $\frac{\partial y_k(t)}{\partial w_{ij}}$:

From (17) and (18) we get

$$\frac{\partial y_k(t+1)}{\partial w_{ij}} = f_k\left[net_k(t)\right] \left[\sum_{I \in U \sqcup I} w_{kI} \frac{\partial y_I(t)}{\partial w_{ij}} + \delta_{ik} z_j(t) \right], \tag{19}$$

where δ_{ik} is the Kronecker delta

$$\delta_{ik} = \begin{cases} 1 & \text{if } i = k, \\ 0 & \text{otherwise.} \end{cases}$$

Because input signals do not depend on the weights in the network,

$$\frac{\partial y_I(t)}{\partial w_{ij}} = 0 \; for \; i \in I,$$

Equation (19) becomes

$$\frac{\partial y_k(t+1)}{\partial w_{ij}} = \left[f_k(net_k(t)) \left[\sum_{I \in U \sqcup I} w_{kI} \frac{\partial y_I(t)}{\partial w_{ij}} \right] b_t(O) + \delta_{ik} z_j(t) \right]. \tag{20}$$

This is a recursive equation. Because we have assumed that our starting state $(t = 0)$ is independent of the weights, then we have

$$\frac{\partial y_k(t0)}{\partial w_{ij}} = 0.$$

These equations hold for all. We, therefore, need to define the values

$$p^k_{ij}(t) = \frac{\partial y_k(t)}{\partial w_{ij}},$$

for every time step t and all appropriate i, j and k. We start with the initial condition

$$p^k_{ij}(t) = 0, \tag{21}$$

and compute at each time step along by substituting (11) from Section III, we get

$$p^k_{ij}(t+1) = \left[f_k(net_k(t)) \left[\sum_{I \in U} w_{kI} p^I_{ij}(t) \right] b_t(O) + \delta_{ik} z_j(t) \right]. \tag{22}$$

The algorithm then consists of computing, at each time step t, the quantities $p_{ijk}(t)$ using (21) and (22) and then using the differences between targets and actual outputs to compute weight changes

$$\triangle\, w_{ij}(t) = \mu \sum_{kinU} e_k(t) p^k_{ij}(t), \tag{23}$$

and the overall correction to be applied to w_{ij} is given by

$$\triangle\, w_{ij}(t) = \sum_{t=t_0+1}^{t1} \triangle\, w_{ij}(t). \tag{24}$$

V. Learning Deterministic Finite State Automata Using Hybrid HMM-RNN Algorithm

In the current study, we investigate to show how finite automaton can be used to train recurrent neural networks making them suitable for modeling dynamical systems. We will train first-order recurrent neural networks on deterministic finite-state automata to show their knowledge acquisition. Finally, we will show how our designed new hybrid recurrent neural networks architecture based on hidden Markov models can train and represent finite automaton making them suitable for modelling dynamical systems.

A. Recurrent Neural Networks as Models of Computation

Recurrent neural networks are systems that model dynamical processes. Formal languages such as finite state automata have characteristics of dynamical systems. Using finite state automata we can show that recurrent neural networks can learn and represent dynamical systems. Recurrent neural networks can be trained with strings whose labels are assigned by deterministic finite state automata.

We generate the training data set by presenting string length of 1 to 10 to corresponding finite automaton which labels the output with each corresponding string. Similarly, we generate a testing data set for string lengths from 1 to length 15. A working set contains patterns of increasing order of lengths. The networks trains on each working set of a number of training epochs until the network converges. The training is terminated when the network performs satisfactorily on the entire training set or iterates through all working sets of the training set.

Fig. 5 represents a ten state deterministic finite sate automata which is used for training the hybrid recurrent network architecture interpreted by hidden Markov models.

An example of 10 State deterministic finite state automata is presented in Fig. 5. Double circles show accepting states. Rejecting states are shown by single circles while state 1 is the automaton start state. The training and testing set is obtained upon presentation of strings to this automaton which gives an output, i.e., a rejecting or accepting state depending on the state where the last sequence of the string was presented. For example, the output of a string of length 10, i.e., aaaabaaaba in alphabet {0,1} is state 7. It is an accepting state, therefore the output is 1.

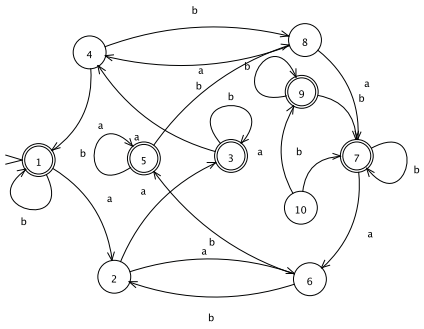

Fig. 5 Ten state deterministic finite state automata

B. Deterministic Finite State Automata Training

To show the effectiveness and efficiency of the proposed algorithm of the hybrid architecture, the following tests were performed.

We generate the training data set by presenting string length of 1 to 10 to a deterministic finite automaton. The deterministic finite automaton labels the output on each string it takes as input depending on the state where the final symbol of the string was presented. Similarly, we generate a testing data set for string lengths from 1 to length 15. We obtain a training data set with 2048 string samples and a testing data set of 65535 strings.

We used the first order recurrent neural networks architecture with the following topology: 2 neurons in the input layer which represents the string input and 1 neuron in the output layer representing the string output. We used a learning rate value of 0.3, momentum rate value of 0.9, sigmoid sensitivity rate of 11.5 and ran experiment with 5, 10 and 15 neurons in the hidden layer along with the hidden Markov model Parameters (No. of observations = 10, No. of observation symbols = 2, No. of States = 3).

While deriving the equations for Hybrid HMM-RNN architecture, we finally ended up with the initial values, weights and observation probabilities of the network. We calculated the training and generalization performances of the data ie., it has correctly classified/learned all the strings in the training and testing sets. The network is presented with data in the testing set and the performance of the networks is determined upon its generalization on the test data. The tables show the results obtained for single order RNN and Hybrid HMM-RNN.

Here, we carried out two experiments

- Case 1: Simple DFA learning using RNN, and
- Case 2: Hybrid HMM-RNN architecture implementation of DFA.

In the first case, the training and generalization performances were quite good and achieved an average of 85 percent performance using simple RNNs, i.e., It has been shown that it correctly classifies all the strings in the training set which means the total number of accepted and rejected strings in the training and the testing sets and in the second case i.e., using the Hybrid HMM-RNN architecture, initially we got zero performance with -1 to 1 and -3 to 3 weight ranges and when we increase the number of hidden units from 5, 10, 15 and weight ranges between -5 to 5, -7 to 7 we got an average of 83 percent on the performances.

C. Learning Deterministic Finite State Automata

Tables I and II show the training and generalization performances of single order RNN and hybrid HMM-RNN architecture with different number of neurons in the input and hidden layers, weight ranges and number of training cycles for learning deterministic finite state automaton.

Finally, in order to reveal that the hybrid recurrent neural network has good recognition/classification and generalization performance to solve the first order and hybrid HMM-RNN problems, the results

TABLE I
SINGLE ORDER RECURRENT NEURAL NETWORK

Input Neurons	Hidden Neurons	Training Cycles	Training MSE	Testing MSE	Generalization MSE
3	5	50	0.194652	92.9169	87.9641
3	5	101	0.177034	78.9566	70.6649
3	5	500	0.150888	85.9326	82.0357
5	10	1000	0.0822267	83.0292	81.4205
5	10	1500	0.0886557	72.4865	71.4137
5	10	2000	0.0854211	60.1392	50.3495
10	15	2500	0.0787231	79.2631	74.1746
10	15	5000	0.184652	68.5339	64.9201

TABLE II
LEARNING DETERMINISTIC FINITE STATE AUTOMATION USING HYBRID HMM-RNN

Hidden Neurons	Weight Range	Training Cycles	Training Performance	Generalization Performance
1	-1 to +1	100	0%	0%
3	-3 to +3	100	0%	0%
5	-5 to +5	53	84.5%	79.3%
10	-7 to +7	31	82.2%	81.3%
15	-15 to +15	9	0%	0%

show that recurrent neural networks can learn deterministic finite state automata by means of gradient descent learning. They show good training and generalization performance compared to other models and it is also seen that the number of training epochs in the last cycle vary for different neural networks topologies depending on the number of neurons in the hidden layer.

VI. CONCLUSION

We have seen how the strengths of intelligent system paradigms can be combined into hybrid systems. We have discussed the combination of neural networks with symbolic knowledge in symbolic connectionist learning framework. We have shown and discussed in detail our proposed hybrid HMM-RNN system and discussed the possible training methods of hybrid recurrent neural networks. Finally, we demonstrated the performance of the developed hybrid Hidden Markov Model-Recurrent Neural Network system by learning a deterministic finite state automata.

The next paper follows on protein classification using the hybrid HMM-RNN architecture and for further future research, the hybrid algorithm can be derived using the Long Short Term Memory networks or Decoupled Extended Kalman Filters to minimize the gradient error.

ACKNOWLEDGMENT

The authors would like to thank South African National Research Foundation for supporting the research.

REFERENCES

[1] Y.S. Abu-Mostafa, Learning from hints in neural networks, *Journal of Complexity* (1990) 6:192.
[2] B. Pandey and R.B. Mishra, Knowledge and intelligent computing system in medicine, *Computers in Biology and Medicine*, 39(3):215-230, 2009.
[3] E. Barnard and D. Casasent, Invariance and neural networks, *IEEE Transactions on Neural Networks*, 2 :498-508, 1991.
[4] Y. Bengio, Neural Networks for Speech and Sequence Recognition, *International Thompson Computer Press*, 1996.
[5] C.M. Bishop, Neural Networks for Pattern Recognition, *Oxford University Press*, 1995.

[6] K. David, D. Haussler, G. Martin Reese and H. Frank Eeckman, A generalized hidden Markov model for the recognition of human genes in DNA, *Procedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, AAAI Press, 1996.
[7] Richard O. Duda, Peter E. Hart, and David G. Stork. 2000. Pattern Classification (2nd Edition). Wiley-Interscience.
[8] J.L. Elman and D. Zipser, Learning the hidden structure of speech, *Journal of the Acoustical Society of America*, 83:1615-1626, 1988.
[9] L. Fu, Mapping rule-based systems into neural architecture, *Knowledge Based Systems*, 3(1):48-56, 1990.
[10] Y. Fukuoka and H. Matsuki, A Modified Back-propagation Method to Avoid Local Minima, *Neural Networks*, 11:1059-1072, 1998.
[11] M.J.F. Gales, Maximum likelihood linear transformations for Hidden Markov Model-based speech recognition, *Computer Speech Language*, 12:75-98, 1998.
[12] A.S. Weigend and N.A. Gershenfeld, The Future of Time Series, Learning and Understanding, *Addison-Wesley*, Reading, MA, 1-17, 1993.
[13] C.L. Giles, D. Chen, C.B. Miller, H.H. Chen, G.Z. Sun and Y.C. Lee, Second-order recurrent neural networks for grammatical inference, *1991 IEEE INNS International Joint Conference on Neural Networks*, Piscataway, NJ, IEEE Press. Reprinted in *Artificial Neural Networks*, eds. E. Sanchez-Sinencia, C. Lau, IEEE Press, 273-281, 1992.
[14] Y. Hayashi and A. Imura, Fuzzy neural expert systems with automated extraction of fuzzy if-then rules from a trained neural network, *Procedings of First IEEE Conference on Fuzzy Systems* (1990) 489-494.
[15] J. Hertz, A. Krogh and R.G. Palmer, *Introduction to the Theory of Neural Computation*. Addison-Wesley Publishing Company, 1991.
[16] J.J. Hopfield, Neural networks and physical systems with emergent collective computational facilities, *Proceedings of the National Academy of Sciences of the USA*, 79:2554–2558, 1982.
[17] K. Hornik, M. Stinchcombe and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks*, 2:359-366, 1989.
[18] J.F. Kolen and S.C. Kremer, (ed.), *A Field Guide to Dynamical Recurrent Networks*, IEEE Press, Piscataway, NJ, 2001.
[19] S. Lawrence, C. L. Giles and A.C. Tsoi, Symbolic conversion, grammatical inference and rule extraction of foreign exchange rate prediction, Decision Technologies for Financial Engineering: *Procedings of the Fourth International Conference on Neural Networks in the Capital Markets, World Scientific, Singapore* (1998) 333-345.
[20] K. Marakami and H Taguchi, Gesture recognition using recurrent neural networks, *Procedings of the SIGCHI conference on Human factors in computing systems*, 237-242, 1991.
[21] L. R. Rabiner, A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition , *Proceedings of the IEEE*, 77(2):257-285, 1989.
[22] R.D. Reed and J. Robert Mark, Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks, The MIT Press, 1999.
[23] A.J Robinson, An application of recurrent neural nets to phone probability estimation, *IEEE transactions on Neural Networks*, 5(2):298-305, 1994.
[24] D.E. Rumelhart, J.L. McCLelland and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Foundations, MIT Press, Cambridge, MA, 1, 1986.
[25] G.G. Towell and J.W. Shavlik, The extraction of refined rules from knowledge-based neural networks, *Machine Learning*, 13(1) (1993) 71–101.
[26] G.G. Towell and J.W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1-2):119-165, 1994.
[27] P.J. Werbos, Backpropagation through time; what it does and how to do it, *Proceedings of the IEEE*, 78:1550-1560, 1990.
[28] R.J. Williams and D. Zipser, A learning algorithm for continually running fully recurrent neural networks, *Neural Computation*, 1(2):270-280, 1989.
[29] R.J. Williams and D. Zipser, Gradient-based learning algorithms for recurrent networks and their computational complexity, In Back-propagation: Theory, Architectures and Applications, 433-486. 1995.

Pavan Kumar Rallabandi is a research student at the University of the Western Cape, South Africa

Kailash Patidar is a professor of applied mathematics at the University of the Western Cape, South Africa. His research is focused on numerical methods and scientific computing for the applied problems that arise from interactions between natural and life sciences as well as engineering. His research is a well balanced combination of analytical investigations and numerical experiments using finite difference, finite element and spline approximation methods. His current research involves the development of reliable numerical methods for mathematical models arising in population biology as well as those for option pricing problems in computational finance.