ISSN:2231-0711
www.ijcset.net

Dynamic Learning Machine Using Unrestricted Grammar

Dr. P. Dinadayalan[*1] Dr. Gnanambigai Dinadayalan[*2]

[*] *Department of Computer Science*
K.M.Centre for P.G. Studies, Puducherry
pdinadayalan@hotmail.com

[*] *Department of Computer Science*
Indira Gandhi College of Arts and Science, Puducherry
dgnanambigai@hotmail.com

***Abstract*— This paper shows a Dynamic Network Learning for performing unrestricted grammar trained Recurrent Neural Network and proves the relationship between Recurrent Neural Network and Turing machine. Dynamic Network Learning employs bi-directional neural network which has feedback path from their outputs to the inputs, the response of such network is dynamic. Turing machine is a finite-state machine associated with an external storage. It prevents indefinite lengthy training sessions. The Dynamic Network Learning architecture is a Recurrent Neural Network and its working principle is related to Turing machine structure. This work exhibits how Turing machine recognizes recursive language and it is also elucidated that Dynamic Network Learning is a stable network.**

***Keywords*— Finite State Machine, Formal languages, Perceptron, Recurrent Neural Network**

I. INTRODUCTION

A Neural Network [1][2][6][9][11] is a mathematical model or computational model that tries to simulate the structure and functional aspects of biological neural networks. The basic building block of a brain and the neural network is the neuron. An artificial neuron consists of many inputs and one output. It has an interconnected group of artificial neurons and processes information using a connectionist approach to computation. The neural network demonstrates the behaviour of the feedforword network and feedback network[1][3][6][7]. In feedforward networks, neurons are organized into different layers that have unidirectional connections between them. Feedforward networks are static in the sense that the outputs depend only on the present input. Feedback neural networks have feedback path from their outputs to the inputs, the response of such network is dynamic or recursive. There are many different types of neural networks, each of which has different strengths particular to their applications.

Finite State Technology describes Finite State Machines and their languages[4][5][6][10]. Each language has its own grammar; based on these grammars the Finite-State Machines behave. Finite-State Machines have four divisions namely Finite State Automaton, Push-down Automaton, Linear Bounded Automaton and Turing Machines. Finite State Automaton accepts regular grammar or type3 grammar. Push-down Automaton generates Context free grammar or type2 grammar. Linear bounded Automaton produces Context sensitive grammar or type1 grammar. Turing Machine generates unrestricted grammar or type0 grammar. Every regular grammar is Context free and every Context -free grammar is Context sensitive and every Context sensitive is unrestricted grammar. Neural network architecture is trained to perform the same computation from a set of examples to recognize a language from a set of examples. Viewing finite state machine as language generators or language acceptors allows to view examples as input strings (of symbols from an appropriate alphabet), belonging or not to the language, to the computation, that is to be learned by the neural network. Section 2 briefly analyses the background work. Section 3 proposes Dynamic Network Learning illustration. Section 4 demonstrates Dynamic Network Learning as Recurrent Neural Network with illustration. Section 5 summarizes the paper.

II. BACKGROUND WORK

The conventional Neural Network Structure [1][4][6][8][13] used for Formal Language Recognizers is Feedforward Recurrent Neural Network. It uses two types of Neural Finite State Machines such as Dynamical Language Recognizer using Regular Grammars and Dynamical Language Recognizer using Context-Free Grammars. Dynamical Language Recognizer using Regular Grammars are less powerful than other Dynamical Language Recognizers. The grammars used in Regular Languages are left-linear and right-linear which takes more number of rules (productions) to generate the language [4][6][13][15]. It solves the problem using regular expression. Some examples of regular expressions are a^*, b^* ,a^*b^* and $(a+b)^*$ where * (Kleene star or looping) denotes indefinite loop symbol.

The conventional Dynamical Language Recognizer using Context-Free Grammar [6][7][13] is an improved model over Dynamical Language Recognizer using Regular Grammar. It

generates context-free language. Every regular language is a context free language. It generates the problems like a^nb^n and wcw^R and solves some definite formal language problems. It does not solve non-context-free languages like $a^nb^na^n$, $a^nb^nc^n$ and $a^mb^nc^{m+n}$. It does not solve halting problem and cannot process recursive languages. Thus the structure of Dynamical Language Recognizer using Regular Grammar and Dynamical Language Recognizer using Context-Free Grammar are mostly unstable. Most of the conventional Dynamical Language Recognizer [1][4][6][7][8][13][15] are used as Feedforward Recurrent Neural Network like Perceptron and Backpropagation Neural Networks. The Perceptron Neural Network is Feedforward single layer network which is not effective for producing formal languages. As the training process in the Feedforward single layer network is not effective, it produces irrelevant and inconsistent results. Backpropagation network or multilayer network has n number of layers. It has more number of hidden layers. The training process in the hidden layers using conventional Dynamical Language Recognizers takes long training session and leads to wrong direction producing inconsistent and improper results. The reasons are that these models cannot achieve stability, computational network interactions, incapable of network learning and insufficient memory.

III. DYNAMIC NETWORK LEARNING

This paper illustrates a novel Dynamic Network Learning which alleviates all the problems in the traditional methods [1][4][6][7][8][13][15]. It is more suitable for Recurrent Neural Network using the power of Turing machine. Turing machine is used for solving all halting problems and leads to proper termination. If the machine is properly terminated, then the network training achieves stability. The Turing machine architecture is applied in Recurrent Neural Network to achieve parallelism. As the structure of Recurrent Neural Network and Turing machine are parallel in nature, the equivalence of Recurrent Neural Network and Turing machine is easily achieved. As the Turing machine generates unrestricted grammar, the grammar formation is very simple and derivation steps of grammar are simpler than other traditional model. The concepts of the conventional research works reveal that a step-by-step advancement is observed in every research work from the previous one. Thus the need of this work is to design a Recurrent Neural Network, which has greater degree of flexibility in terms of learning speed, parallelism, generalization and stability.

A. Dynamic Network Learning as Turing Machine

This section deals with Dynamic Network Learning to generate recursive language to attain stability. The Dynamic Network Learning structure is a Recurrent Neural Network and its working design is based on Turing machine structure. The principle for designing Dynamic Network Learning is derived from Recurrent Neural Network which functions according to the transitions of Turing machine. The Turing machine consists of three independent tapes with three separate heads. The states and inputs of three-tapes are the same as in a standard Turing machine. The Turing machine reads the tape simultaneously but has only one state. It is a finite-state machine associated with an external storage or memory medium. An input tape is divided into cells, one next to the other. Each cell contains a symbol from some finite alphabet. The alphabet contains a special blank symbol and one or more other symbols. The tape is assumed to be arbitrarily extendable to the left and to the right, i.e., the Turing machine is always supplied with as much tape as it needs for its computation. Cells that have not been written to before are assumed to be filled with the blank symbol. In some models the tape has a left end marked with a special symbol; the tape extends or is indefinitely extensible to the right.

A transition is determined by the state and the symbols are scanned by each of tape heads. A transition in three-tape machine may change the state, with a symbol on each of the tape, and independently reposition each of the tape heads. The repositioning consists of moving the tape head one cell to the left or one cell to the right or leaving it at its current position. A model of computation consisting of a set of states, a start state, an input alphabet, and a transition function that maps input symbols and current states to a next state. Computation begins in the start state with an input string. It changes to new states depending on the transition function. A read/write that can read and write symbols on the tape and move the tape left, right one (and only one) cell at a time and stationary (no move). A finite control stores the state of the Turing Machine. The number of different states is always finite and there is one special start state. In one move the Turing machine, depending upon the symbol scanned by the tape and the state of the finite control: either erase or write a symbol, and then move the head ('L' for one step left *or* 'R' for one step right or 'S' for stationary), and then assume the same or a new state as prescribed

A three-tpe Turing machine M is constructed to accept the recursive language L. The derivations of the unrestricted grammar G are simulated on a three tape Turing machine. $Tape_1$ holds the input pattern and $tape_2$ a representation of rule of G. A rule $u \rightarrow v$ is represented on $tape_2$ by the string u#v. Rules are separated by two consecutive #'s. The derivations of G are simulated on tape3. When a rule $u \rightarrow v$ is applied to the string xuy on $tape_3$, the string xvy is written on the tape following xuy#. The symbol # is used to separate the derived strings. The $tape_1$ holds the target pattern. The $tape_2$ contains the productions of the grammar. The $tape_3$ has the derivation of the grammar where the production is replaced by another production. If $tape_1$ and $tape_3$ match, the Dynamic Network Learning accepts and halts. Sometimes the traditional dynamic networks are unstable [1][4][6][7][8][13][15]. That is, the training sessions are indefinite lengthy sessions and training will not complete. Therefore, the network will not halt. But Dynamic Network

Learning produces the constant result and the Turing machine can properly terminate. Here it compares length of the target pattern ($tape_1$) and length of the actual output ($tape_3$). If the length of $tape_3$ string is less than the length of $tape_1$ string, then the training process continues until the $tape_1$ and $tape_3$ matches. If the length of $tape_3$ string is greater than the length of $tape_1$ string, then the training process stops and the Turing machine halts in a rejecting state. Hence the given input is not present in the language. Here when an input string (pattern) is given, the Dynamic Network Learning accepts it if the string belong to the language or rejects it if the string does not belong to the language. Thus the Dynamic Network Learning generates the recursive language

B. Experimental Results

The languages such as $\{a^nb^nc^n\}$ can be generated using unrestricted grammar G. The unrestricted grammar G is implemented in the Turing machine to produce the language L. As Turing machine generates recursive language, the language L generated by Turing machine is also a recursive language. It is hereby proved that Dynamic Network Learning produces recursive language which helps to achieve stability. This is explained with an illustration given below.

An unrestricted grammar G = (V, T, P, S) generates the language $L = \{a^nb^nc^n \mid n \geq 1\}$, where V = {S,A,C} is a set of non-terminals, T={a,b,c} is the set of terminals, P is the set of productions and S is the start symbol.

The production P is described as follows.

P: $S \rightarrow aAbc \mid abc$

$A \rightarrow aAbc \mid abC$

$Cb \rightarrow bc$

$Cc \rightarrow cc$

A sample for the word pattern *w* = 'aaabbbccc' is taken to generate the language $L = \{a^nb^nc^n \mid n \geq 1\}$ using the unrestricted grammar G and the derivation is given below.

S => a<u>A</u>bc (since S → aAbc)

=> aaAbC<u>bc</u> (since A → aAbC)

=> aaab<u>Cb</u>Cbc (since A → abC)

=> aaabbC<u>Cb</u>c (since Cb → Cb)

=> aaabb<u>Cb</u>Cc (since Cb → Cb)

=> aaabbbC<u>Cc</u> (since Cc→ cc)

=> aaabbb<u>Cc</u>c (since Cc → cc)

=> aaabbbccc (since Cc → cc)

TABLE I

LANGUAGE $L=\{a^nb^nc^n\}$ IS IMPLEMENTED DYNAMIC NETWORK LEARNING WITH CORRECT PATTERN

Transition of Turing Machine	Desired output ($tape_1$)	actual output ($tape_3$)	Rule extracted from $tape_2$	A rule (u →v) u is replaced by v Length(u)≤ length(v)	
				Length(u)	length(v)
q_1	Aaabbbccc	<u>S</u>	---	0	1
q_2	aaabbbccc	=>a<u>A</u>bc	S→aAbc	1	4
q_3	aaabbbccc	=>aa<u>A</u>bCbc	A→aAbC	1	4
q_4	aaabbbccc	=>aaab<u>Cb</u>Cbc	A→abC	1	3
q_4	aaabbbccc	=>aaabbC<u>Cb</u>c	Cb→bC	2	2
q_4	aaabbbccc	=>aaabb<u>Cb</u>Cc	Cb→bC	2	2
q_4	aaabbbccc	=>aaabbbC<u>Cc</u>	Cb→bC	2	2
q_5	aaabbbccc	=>aaabbbC<u>Cc</u>	Cc→cc	2	2
q_5	aaabbbccc	=>aaabbb<u>Cc</u>c	Cc→cc	2	2
q_5	aaabbbccc	=>aaabbbccc	Cc→cc	2	2

A three tape Turing machine is taken to implement the language $L=\{a^nb^nc^n \mid n \geq 1\}$ in Dynamic Network Learning. The input word *w* = 'aaabbbccc' is stored in $tape_1$. The productions of grammar G are stored in $tape_2$. The derivation process which carried out in $tape_3$ is described in the table.1. The table.1 consists of state transition of Turing machine, desired output ($tape_1$), actual output ($tape_3$), rules extracted from $tape_2$, length (u) and length (v). The input pattern *w* = 'aaabbbccc' is taken from the language L to process it. A rule u→v is taken from $tape_2$. Every derivation is calculated and tabulated. The length (u) and length (v) should satisfy the production rule so that Dynamic Network Learning is refrained from entering into indefinite loop or indefinite cycle.

For every transition, the length of desired output and length of actual output are compared from the table 1. If the length of actual output is less than the length of desired output, the processing of Dynamic Network Learning continues. If the length of actual output is equal to the length of desired output and all its symbols contain terminals, then Dynamic Network Learning accepts the given input pattern and halts. If the length of actual output is less than the length of desired output and all its symbols contain terminals, then the given input pattern is wrong. Therefore the Dynamic Network Learning rejects the input pattern and halts. This is shown in the table 2. If the length of actual output and the length desired output than Dynamic Network Learning rejects the input pattern and halts. Thus Dynamic Network Learning is abstained from entering into indefinite loop. From the above discussion it is concluded that Dynamic Network Learning accepts recursive language and Dynamic Network Learning achieve stability.

TABLE II
LANGUAGE L={$a^n b^n c^n$} IS IMPLEMENTED DYNAMIC NETWORK LEARNING WITH INCORRECT PATTERN

Transition	desired output ($tape_1$)	actual output ($tape_3$)	Rule extracted from $tape_2$	A rule (u →v) u is replaced by v length(u)≤ length(v)	
				length(u)	length(v)
q_1	aabbbcc	=>S	---	0	1
q_2	aabbbcc	=>aAbc	S→aAbc	1	4
q_3	aabbbcc	=>aabCbc	A→abC	1	3
q_4	aabbbcc	=>aabbCc	Cb→bC	2	2
q_5	aabbbcc	=>aabbcc	Cc→cc	2	2

IV. DYNAMIC NETWORK LEARNING AS RECURRENT NEURAL NETWORK

In this section it is proved that the structure of the Dynamic Network Learning is designed based on Recurrent Neural Network. The Dynamic Network Learning is possible to keep recurrent network in a stable position through recursive language and thus it is proved that Dynamic Network Learning is a recursive language Recognizer. The details of the recurrent network from the transitions of the Turing machine are discussed below. The proof given in this section illustrates that Dynamic Network Learning is a language Recognizer. Recursive languages are generated by Turing machine. Recursive language is to be implemented in recurrent network (dynamical network). That is, recurrent network generates recursive languages. The advantage of generating recursive language is to achieve stability in recurrent network. The Dynamic Network Learning is used to train the given input pattern and produce a desired output. In the Dynamic Network Learning if the input pattern given is correct or even if the input is partially incomplete or partially incorrect, the Dynamic Network Learning trains and produces the correct desired output. Then the desired output is compared with the target output.

Dynamic Network Learning consists of three layers: $layer_1$ serves no computational function which simply distributes the outputs back to the inputs, each $layer_2$ neuron computes the weighted sum of its inputs and $layer_3$ is an output layer which produces the actual output. If the actual output and target output match, then the Dynamic Network Learning recognizes the input pattern and halt. Otherwise, training process continues until the network produces a constant value. The internal representation of Dynamic Network Learning is implemented by Turing machine. Each and every iterations in the training process is called a state. The Turing machine recognizes a language (the set of string accepted by the Recognizer) by being presented with an input string. The given string is either accepted or rejected as part of the language, depending on the resulting point.

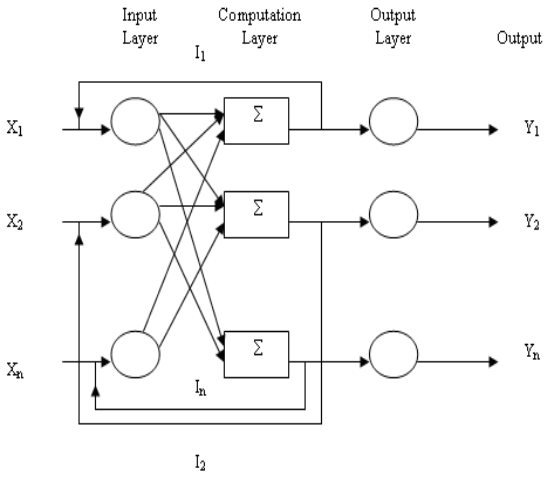

Fig. 1 Architecture of Dynamic Network Learning using Recurrent Neural Network

The Dynamic Network Learning has three primary layers: input layer, computation layer and output layer. The neurons in input layer serve only as fan-out points and no computation. The computation layer is fully connected with the output layer and does the supervised learning. The Dynamic Network Learning is trained in two successive steps. The first step is performed between the input layer and the computation layer. The second step is performed between the computation layer and output layer. An input vector is applied, the input layer inputs are established, and the Dynamic Network Learning outputs are calculated as in normal operation. Each weight is adjusted only if it connects to an input layer. The Dynamic Network Learning training is supervised. Supervised training requires input pattern with target pattern representing desired output. When an input pattern is applied, the output of the network is calculated and compared to the corresponding target pattern, and weights are changed according to an algorithm that tends to minimize the error. Computation layer produces the desired output which is the input of the next iteration.

The start symbol S in the grammar of recursive language is given as the input of recurrent network. i.e., S is assigned to X where X is the input of recurrent network. All the productions of the grammar are considered as the weight matrix W. The output of the recurrent network is recirculated to the input of it till the target is achieved. To begin the training, input X is given as the input of recurrent network. Weight W is applied to X to get NET value. The NET value is called the actual output. The actual output is compared with the target output. When the target is matched with the actual output the recurrent network recognize the input, generates recursive language and halts. When the target does not match with the actual output, the output is recirculated as input to continue the training process. In the training process the length of the actual output and the length of the target are calculated in order to avoid indefinite training process. When

the length of the actual output is longer than the target, the training process is stopped to reject the given input and halt the machine. Otherwise, it leads to instability. Therefore, the Dynamic Network Learning will either accept or reject the given input but it will not go to indefinite state. Thus the Dynamic Network Learning generates recursive language as it produces recursive language recurrent network is a Turing machine.

Dynamic Network Learning has a closed loop in the network topology. In Dynamic Network Learning, every neuron receives inputs from every other neuron in the network. It is defined as one in which either the network's computation unit activations or output values are fed back into the network as inputs. In this network, inputs are received from an external source, passed to a computation layer, and then on to the output layer. The signal from the output layer is passed to an external source, as well as back to a same layer which then acts as an input layer (along with the actual input layer) to the computation on the next pass. As the output of the network is used along with a new input to compute the output of the network, the response of the network is dynamic. That is, the network's response can be stable (successive iterations produce smaller and smaller output changes until the outputs become constant). The structure of the Dynamic Network Learning is dynamic and stable. The Dynamic Network Learning properly ends with the input pattern.

A. Experimental Results

In section 4, it has been proved that Turing machine is used in Recurrent Neural Network to generate recursive language through which is achieved. The example discussed in section 3.2 is taken here for illustration. The language such L = $\{a^nb^nc^n\}$ can be generated using unrestricted grammar G. The unrestricted grammar G is implemented in the Turing machine to produce the language L. The Turing machine consists of (Q, ∑, Γ, δ, q_0, B, F), where Q= $\{q_0,q_1,q_2,q_3\}$ is a set of stats, ∑ = {a,b,c} is the set of alphabets, Γ = {a, b, c, S, A, C} is the set of input tape symbols, δ is the transition function which maps from Q to Γ, q_0 is the initial state, B = { Δ } is the blank symbol and F = $\{q_3\}$ is the final state. The transition function δ is described as follows.

$\delta (q_0,S,R) = (q_1, aAbc)$

$\delta (q_0,S,R) = (q_0, abc)$

$\delta (q_1,A,R) = (q_1, aAbC)$

$\delta (q_1,A,R) = (q_2, abC)$

$\delta (q_2,Cb,L) = (q_2, bC)$

$\delta (q_3,Cc,L) = (q_3, cc)$

The equivalence of state transition function δ of Turing machine with the training sets for Recurrent Neural Network. The equivalence of state transition of Turing machine with training set of Recurrent Neural Network is demonstrated in table 3. The table 3 consists of two divisions. The first division shows the transition functions of Turing machine and the division deals with training set of Recurrent Neural Network. The transition function of Turing machine contains present state with input, tape move direction and next state with output.

The Recurrent Neural Network training set consists present input pattern X_i, actual output O_i and adjustment weight W_i. Every iterations of Recurrent Neural Network has its equivalent state transition of Turing machine. The training set contains the input patternX = { S }, actual output O_i = $\{a^nb^nc^n\}$ and the weight vector W_i = { S → aAbc ; S → abc ; A → aAbc ; A → abC; Cb → bc ; Cc → cc}. For example in table 3, iteration 3 shows Turing machine transition function which is denoted by $\delta(q_1,A,R) = (q_2,abC)$ and its corresponding Recurrent Neural Network training set which are X = { aaAbCbc }, O = { aaabCbCbc } and W = { abC }. An input pattern is applied either from input vector X_i or from a output vector O_i. Each of these is replaced by a weight vector W_i and actual output is produced. The state of the network is based on the state of the Turing machine. In the table 3, the language L=$\{a^nb^nc^n\}$ is applied by the transitions of the Turing machine to implement Recurrent Neural Network. Therefore Recurrent Neural Network not only generates recursive language but also achieves stability. Thus Dynamic Network Learning is trained to behave like Turing machine to generate recursive language to reach stable form.

TABLE III
EQUIVALENCE OF TURING MACHINE AND RECURRENT NEURAL NETWORK

Iteration	TM transitions function	Recurrent Neural Network		
		Present input pattern X_i	Actual output O_i	Adjustment weight vector W_i
1	$\delta(q_0,S,R) = (q_1,aAbc)$	S	aAbc	aAbc
2	$\delta(q_1,A,R) = (q_1,aAbC)$	aAbc	aaAbCbc	aAbC
3	$\delta(q_1,A,R) = (q_2,abC)$	aaAbCbc	aaabCbCbc	abC
4	$\delta(q_3,Cb,L) = (q_2,bC)$	aaabCbCbc	aaabbCCbc	bC
5	$\delta(q_3,Cb,L) = (q_2,bC)$	aaabbCCbc	aaabbCbCc	bC
6	$\delta(q_3,Cb,L) = (q_2,bC)$	aaabbCbCc	aaabbbCCc	bC
7	$\delta(q_3,Cc,L) = (q_3,cc)$	aaabbbCCc	aaabbbCcc	cc
8	$\delta(q_3,Cc,L) = (q_3,cc)$	aaabbbCcc	aaabbbccc	cc

V. CONCLUSION

Dynamic Network Learning overcomes the drawbacks faced by the traditional Dynamical language Recognizers. The Dynamic Network Learning is a Dynamical Neural Network and its internal structure is designed as Turing machine. Dynamic Network Learning generates recursive languages. The internal representation of Dynamic Network Learning is implemented by Turing machine. The Turing machine recognizes a language by being presented with an input string. The input pattern is either accepted or rejected as part of the language, depending on the resulting point. The advantage of generating recursive language is to achieve stability in dynamic network. Dynamic Network Learning produces the constant result and the Turing machine can properly terminate. Dynamic Network Learning produces recursive language, which is a suitable language for solving decidable problems. It is dynamic in nature and the architecture is Recurrent Neural Network. Hence the Dynamic Network Learning is a language Recognizer.

REFERENCES

[1] Aussem, A., Murtagh, F., and Sarazin, M. (1995). Dynamical recurrent neural networks towards environmental time series prediction. International Journal of Neural Systems, 6:145 - 170.

[2] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2):157-166.

[3] Bradley, M. J. and Mars, P. (1995). Application of recurrent neural networks to communication channel equalization. In IEEE International Conference on Acoustics, Speech and Signal Processing, volume 5, pages 3399-3402.

[4] Carrasco, R. C., Forcada, M. L., Valdes-Munoz, M..A., and Neco, R. P. (2000). Stable encoding of finite-state machines in discrete-time recurrent neural nets with sigmoid units. Neural Computation, 12(9):2129 - 2174.

[5] Cechin, A.L, Regina. D, Simon. (2003). State automata extraction from recurrent neural nets using k-means and fuzzy clustering. Chilean Computer Science Society, SCCC 2003. Proceeding. 23rd International conference of the Volume, Issue, 6-7, Pages: 73-98.

[6] Dinadayalan.P, Gnanambigai Dinadayalan, R. Vasantha Kumari, (2010), "Neuro Language Generator", International Journal on Computer Science and Engineering (IJCSE), ISSN: 1453 - 1461, Vol. 02, No. 04, 2010, 0975-3397, August 2010..

[7] Elman, J.L., (1995). Language as a dynamical system in (eds) Port, R.F. & van Gelder, T. Mind as Motion: Explorations in the Dynamics of Cognition, pp 195-225, Cambridge MA: MIT Press.

[8] Forcada, M. L. and Carrasco, R. C. (2001). Simple stable encodings of finite state machines in dynamic recurrent networks, pages 103-127. IEEE Press.

[9] Gnanambigai Dinadayalan, P. Dinadayalan, K. Balamurugan, (2011), "Hybrid Network Learning", International Journal of Computer Applications, ISSN: 0975 – 8887, 21(10):30-34.

[10] Gomez.J (2004). An Incremental Learning Algorithm for Deterministic Finite Automata Using Evolutionary Algorithms, Proc. Genetic and Evolutionary Computation.

[11] M. Hasenjager and H. Ritter (2002). Active Learning in neural networks. Physica-Verlag Studies in Fuzziness and Soft Computing Series, New learning paradigms in soft computing, Ed. L. C. Jain and J. Kacprzyk, 137-169.

[12] Shinichi Kikuchi, Masakazu Nakanishi (2003). Recurrent neural network with short-term memory and fast structural learning method. Wiley Periodicals, Inc. Syst. Comp Jpn, 36(6): 69-79.

[13] Simon M. Lucas, T. Jeff Reynolds (2005). Learning Deterministic Finite Automata with a Smart State Labeling Evolutionary Algorithm. IEEE transactions on Pattern analysis and machine intelligence, pp.1063-1074.

[14] Tabor, W., (2001). Sentence Processing and Linguistic Structure in Kolen, J.F. & Kremer, S.C. (eds), A Field Guide to Dynamical Recurrent Networks, pp 291-309, New York: IEEE Press.

[15] Wiles, J., Blair, A.D. & Boden, M., (2001). Representation Beyond Finite States: Alternatives to Pushdown Automata in Kolen, J.F. & Kremer, S.C. (eds) A Field Guide to Dynamical Recurrent Networks, pp 129-142, New York: IEEE Press.