# Neuro Language Generator

P. Dinadayalan
Department of Computer Science, K. M. centre for P.G. Studies, Puducherry, India

Gnanambigai Dinadayalan
Department of Computer Science, Indira Gandhi Arts and Science College, Puducherry, India.

R. Vasantha Kumari
Principal, Villanur College for Women, Puducherry, India.

***Abstract*** **- 'Neuro Language Generator using Finite State Machine' is based on neural network and finite state machine. The fundamental properties of neural network along with the power of Turing machine prove how it can be implemented for formal language processing. This paper elaborates the conventional dynamical language generators, limitations of the conventional dynamical language generators and proposes a new architecture for formal language processing. The conventional dynamical language generators used for neural language generators is feedforward RNN. It expresses dynamical language generator using finite automaton and dynamical language generator using pushdown automaton. Conventional dynamical generators tend to have stability problem, incapable of network training and lack of memory. It is proposed that the new method 'Neuro Language Generator using Finite State Machine' solves most of the problems, which the traditional methods fail to do. The approach employs finite state technology for a RNN in the task of learning to achieve stability in network structure. RNN architecture performs the same computation as a Turing machine. The RNN architecture acts as a language generator, which accepts formal language. Neuro Language Generator is a RNN that uses feedback connections. NLG can be used to solve more complicated problems compared to traditional dynamical generator**.

***Keywords*** **- Artificial Neural Network ; Dynamical Language Generators; Finite State Machine; Recurrent Neural Network; Turing Machine**

## I. CONVENTIONAL DYNAMICAL LANGUAGE GENERATORS

As most of the work performed by researchers in connection with neural networks, it is inferred that formal language and computation theory concern the training of RNN to identify neural finite-state automata and regular language and, neural pushdown automata and context-free language. RNN architecture is capable of performing the same computation as a finite-state machine (FSM).

### *A. Neural Acceptors/Generators*

RNN may be trained to accept strings belonging to a language and reject strings not belonging to it, by producing suitable labels after the whole string has been processed [1][5]. In view of the computational equivalence between some RNN architectures and some finite-state machine classes, it is reasonable to expect RNN to learn regular (finite-state) languages. A set of neural acceptors (separately or merged in a single RNN) may be used as a neural classifier.

### *B. Neural Transducers/Translators*

If the output of the RNN is examined not only at the end but also after processing each one of the symbols in the input, then its output may be interpreted as a synchronous, sequential transduction (translation) for the input string [5]. DTRNN may be easily trained to perform synchronous sequential transductions and also some asynchronous transductions.

### *C. Neural Predictors*

RNN may be trained to predict the next symbol of strings in a given language. The trained RNN, after reading string outputs a mixture of the possible successor symbols; in certain conditions , the output of the RNN may be interpreted as the probabilities of each of the possible successors in the language[5]. In this last case, the RNN may be used as a probabilistic generator of strings. When RNN are used for grammatical inference, the following have to be defined: learning set and learning algorithm. The learning set may contain: strings labeled as belonging or not to a language or as belonging to a class in a finite set of classes (recognition/classification task); a draw of unlabeled strings, possibly with repetitions, generated according to a given probability distribution (prediction/generation task); or pairs of strings (translation/transduction task). A learning algorithm (including a suitable error function and a suitable

stopping criterion) and a presentation scheme (the whole learning set may be presented from the beginning or a staged presentation may be devised). Section I elaborates the conventional dynamical language generators. Section II briefly gives the power of Turing machine. Section III analyses the limitations of conventional dynamical language generators. Section IV presents a new dynamical approach 'Neuro Language Generator using Turing machine'. Section V proposes a new architecture 'Neuro Language Generator'. The new architecture consists of training data, network structure, objective function, training algorithm and output of the proposed method. Section VI illustrates the experimental results of Neuro Language Generators.

*D. Dynamical language generators using finite state automata*

Finite automata with RNNs are finite state machines that consist of a finite number of states, and transition between those states based only on the current state and an input symbol [2][7][10][11]. When the input has all been consumed, the machine accepts the string of inputs, based on the final state of the machine. These machines happen to correspond exactly with regular languages, with the set of strings accepted by any finite automaton being a regular language, and with every regular language having a machine that accepts all and only strings from that language. The class of regular languages is equivalent to the class of languages recognized by finite automata. A finite state transducer (FST) with neural network is a finite state automaton, which generates both input and output [11][12][13]. There are two finite state transducers such as neural moore and neural mealy machine. The class of languages generated by finite automata is known as the class of regular languages. The Moore machine with neural networks [6] uses only entry actions, i.e. output depends only on the state. The output from a Moore machine is associated with the state only. The advantage of the Moore model is a simplification of the behaviour. Mealy machine with neural networks is a finite state transducer that generates an output based on its current state and input [1][7][10][11]. This means that the state transition includes both an input and output for each transition. In contrast, the output of a Moore finite state machine depends only on the machine's current state; transitions are not directly dependent upon input.

*E. Dynamical language generators using pushdown automata*

Similar to the neural finite state automata except that it has an available stack which is allowed to grow to arbitrary size. The state transitions additionally specify whether to add a symbol to the stack, or to remove a symbol from the stack [1][2][3]. Pushdown Automata are finite-state automatons with a stack. A stack is a data structure that can contain any number of elements, but for which only the top element may be accessed (hence PDAs have an infinite set of states) but which can be only accessed in a Last-In-First-Out (LIFO) fashion. Stack functions as the required memory. The Finite State Control (FSC) reads inputs, one symbol at a time. Based on the input symbol, current state and the top symbol on the stack, FSC does some state transitions and operations to the stack content. Stack could be kept unchanged, or some thing could be pushed into the stack and could be popped out of the stack. The languages which can be recognized by PDA are precisely the context free languages.

As finite-state automata with neural network correspond to regular languages, the Context-Free Languages (CFL) have corresponding network system called pushdown automata (PDA) [5][12][13]. Regular expressions are generators for regular languages and Finite Automata are generators for them. Similarly for Context-free Languages, Context Free Grammars (CFGs) are neural generators and Pushdown Automata (PDAs) are recognizers. The class of context free languages is the same as the class of languages recognized by machines called pushdown automata. Pushdown automata are equivalent to context-free grammars: for every context-free grammar, there exists a pushdown automaton such that the language generated by the grammar is identical with the language generated by the automaton, and for every pushdown automaton there exists a context-free grammar such that the language generated by the automaton is identical with the language generated by the grammar. A language L is said to be a Context-Free-Language (CFL) if its grammar is Context-Free [5][12][13]. More precisely, it is a language whose words, sentences and phrases are made of symbols and words from a Context-Free-Grammar. Context-free grammars are simple enough to allow the construction of efficient parsing algorithms which for a given string determine whether and how it can be generated from the grammar. Context-free grammars are important because they are powerful enough to describe the syntax of programming languages.

PDAs with neural networks are better than FSAs with neural networks. As with the regular languages, there are many languages which are not context-free. The stack on the PDA, while it provides infinite storage capacity, is still a stack, and so only the last element placed on it can be accessed at any given time. Accessing earlier elements requires removing and thus losing the later elements, since there is no other stack on which to place them. The PDA is also limited in that it must consume the input characters in the order in which they are received, and cannot access them again, except by placing them on the stack.

## II. POWER OF TURING MACHINE

The Turing machine is one of the intriguing intellectual discoveries [4][8][9]. Turing machine is a simple and useful abstract model of computation that is general enough to embody any computer program. It is equivalent to power of the most programming languages. The Turing machine is the most powerful machine with power to emulate any of the other machines. The Turing machine is far more powerful than any DFA, PDA or LBA. It is similar to a DFA or PDA [8][9]. The Turing machine takes a tape with a string of symbols on it as an input, and can respond to a given symbol by changing its internal state, writing a new symbol on the tape, shifting the tape right or left to the next symbol, or halting. It can read or write to the tape. It can move left or right on the tape. It halts as soon as it reaches either the special accept state or the special reject state. Equivalent to the class of unrestricted languages is the class of languages recognized by a Turing machine. Unrestricted grammars are much more powerful than restricted form like the regular and context-free grammars. It has the following abilities.

A Turing machine with a two-way infinite tape [8][9] is identical to the standard model except that the tape extends indefinitely in both directions. Since a two-way tape has no left boundary, the input can be placed anywhere on the tape. All other tape positions are assumed to be blank. The tape head is initially positioned on the blank to the immediate left of the input string.

A multi-tape Turing machine consists of n tapes and n independent tape heads [4][8][9] . The states and inputs of a multi-tape machine are the same as in a standard Turing machine. The Turing machine reads the tape simultaneously but has only one state. A transition is determined by the state and the symbols are scanned by each of the tape heads. A transition in a multi-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 Turing machine with k-dimension is called multidimensional Turing machine [8] [9]. The device has the usual finite control, but the tape consists of a k-dimensional array of cells infinite in all 2k directions, for some fixed k. Depending on the state, symbol scanned, the device changes state, prints a new symbol, and moves its tape head in one of 2k directions, either positively or negatively, along one of the k axes. Initially, the input is along one axis, and the head is at the left end of the input.

A k-head Turing machine has some fixed number, k, of heads [8][9]. The heads are numbered 1 through k, and a move of the Turing machine depends on the state and on the symbol scanned by each head. In one move, the heads may each move independently left, right, or remain stationary.

An off-line Turing machine is a multi-tape Turing machine whose input tape is read only [8][9]. Usually the input is surrounded by end markers, ¢ on the left and $ on the right. The Turing machine is not allowed to move the input tape head off the region between ¢ and $. It should be obvious that the off-line Turing machine is a special case of the multi-tape Turing machine.

A Turing machine halts when it no longer has any available moves. If it halts in a final state, it accepts its input; otherwise, it rejects its input. The TM is started on a tape containing a string w $\epsilon$ $\Sigma^*$ at the beginning of the tape and blanks B after it. A TM accepts a string w when it enters a final state in F; if the string is not accepted, the TM may or may not stop. It may be shown [8] that the class of languages accepted by TM is the same as the class of languages generated by unrestricted grammars.

The Halting Problem is a very strong, provably correct, statement that no one will ever be able to write a computer program or design a Turing machine that can determine if a arbitrary program will halt (stop, exit) for a given input [4][8][9]. Turing machines can solve halting problems and compute results based on inputs. A halting problem is a computational problem where the answer is always yes or no (accept/reject). The halting problems are recursive. A language is recursive if there exists a Turing machine that accepts every string of the language and rejects every string that is not in the language. Unrestricted grammar generates recursive language.

## III. LIMITATIONS OF CONVENTIONAL DYNAMICAL LANGUAGE GENERATORS

From the literature survey, neural finite automaton dynamical language generators (neural finite automata) and pushdown automaton dynamical language generators (neural PDA) [1][5][7][10][11][12][13] are various recurrent networks having their own strengths and limitations. The conventional dynamical language generators [1][5][6][7] [10][11][12][13] used for formal language generations is feedforward neural network. The existing works express neural finite automaton and neural PDA. As it has been examined that recurrent network can be modeled by neural finite automaton, it generates a specific class of the language. Neural finite state automaton generates regular languages. Neural finite automata with outputs are neural Moore machine and mealy machine [1][7]10][11]. These machines have both input pattern and target pattern. In neural Moore machine the target pattern depends on its present state. In neural mealy machine the target pattern depends on its transition. These two machines

also generate regular languages. The limitations of these machines are that machines mostly produce indefinite loops when finite automata are implemented in RNN symbol [1][6][10][11] [12][13]. The neural finite automata leads to indefinite loop and thus stability and generalization cannot be attained in neural finite state automaton. The neural finite automaton can remember only current input pattern. It does not remember previous long sequence of input pattern.

The conventional neural PDA [1][2][3] is an improved model over neural finite state automaton which uses context-free language. It uses a stack, which helps the networks to remember any arbitrary long input sequence and thus generates larger classes of languages than that of neural finite automata. As these networks are developed using stack concepts, they could not solve all the problems of the formal language. The training algorithm depends on stack operation of neural PDA. Neural PDA does not solve non context-free languages. It does not solve halting problem and cannot process recursive languages. Neural PDA tends to have stability problems when presented with the input strings which are longer than those used for training. The major limitations of neural PDA are: it solves limited problems, has the stability problem that does not halt, generate only context free languages, and has less computational capability such as speed, storage, retrieval and comparison [1][2][3]. Feedforward network is used in neural finite automaton and neural PDA [1][5][6][7][10][11][12][13]. The structure of neural finite state automaton and neural PDA are mostly unstable. The feedforward neural network is not effective for producing formal languages. As the training process in the feedforward is not effective, it produces irrelevant and inconsistent results. The training process in the feedforward neural network using finite state automaton and PDA takes long training session and leads to wrong direction producing inconsistent and improper results. The conventional dynamical language generators [1][5][6][7][10][11][12][13] cannot achieve stability, computational network interactions, incapable of network learning and insufficient memory. Moreover feedforward dynamical language generators produce incorrect and inconsistent results. In the training processes in most of the occasions, the feedforward dynamical generator never comes to an end thereby the dynamical generator attains the indefinite loop. The time taken for training process is more and the training process never comes to an end. So, the stability of the traditional approaches is very low. It is very difficult to design dynamical generators using finite automata and PDA recurrent networks.

## IV. NEURO LANGUAGE GENERATOR USING TURING MACHINE

This section presents a new dynamical approach, 'Neuro Language Generator using finite state machine'. On analysis of the related domain such as neural finite automata and neural PDA, their limitations reveal that the researchers [1][5][6][7][10][11][12][13] have given less notice to halting problem. Moreover the previous research contributions fail to achieve stability and generalization. As researchers have not contributed the efforts in this direction there have been many unanswered questions which may lead to construct a new model to alleviate the hurdles faced by the conventional models. Traditionally, finite state technology and neural network have been investigated along with finite automata based RNN and pushdown automata based RNN. Investigating finite automata based RNN and pushdown automata based RNN their limitations are identified and the computational difficulty of this task is clarified by examining specific model for formal language processing [1][5][6][7][10][11][12][13]. The proposed model expands the approach that encompasses RNN by coupling the power of Turing machine with its speed and storage efficiency. Turing machine is dominant than other finite state machines such as finite automata and pushdown automata. Turing machine is used to achieve stability and generalization of the neural network. It is quite natural to consider the possibilities of integrating the two paradigms into a new kind of system where the desired strengths of both systems are utilized and combined appropriately. In fact there has been a great amount of interest and practice in the synergetic combination of finite state technology and neural networks, with an expectation that the capacity of the hybrid system will be greatly enhanced. Treating finite state technology and a neural network as two different computational elements, they can be configured at a system level in a hierarchical manner. The overall task is divided hierarchically into levels, some of which are completed by neural networks and the others by finite state machines. Traditional models are static which never complete the training. But still the network has certain limitations and which are removed by new dynamical network called Neuro Language Generator using Finite State Machine (Turing machine). The proposed Neuro Language Generator using finite state machine is a RNN that can change its behaviour to accept perpetual innovation. This work discusses the use of Neuro Language Generator and Turing machine to generate recursive languages. The internal architecture of Neuro Language Generator is a Turing machine. Turing machine prevents indefinite training. Neuro Language Generator is a RNN that uses feedback connections. NLG can be used to solve more complicated problems compared to traditional dynamical generator.

## V. ARCHITECTURE OF NEURO LANGUAGE GENERATOR

Neuro Language Generator structure is based on RNN with a Turing machine. Neuro language generator is a single layer RNN (dynamical network) which consists of three-

layer of connection weights. Neuro Language Generator possesses as rich repertoire of dynamics which result in their being capable of performing powerful tasks such as formal language generation. The structure of the Neuro Language Generator is dynamic and stable. The Neuro Language Generator properly ends with the input pattern (string). The network of NLG is also called dynamical language generator where there are feedback-connection from a unit back to itself. Neuro language generator consists of a set of highly interconnected entities, called nodes or units. Each accepts a weighted set of inputs and responds with an output. Neuro language generator can be divided into five units, namely training data unit, network unit, objective function unit, training unit and output unit. The architecture of Neuro language generator is shown in figure1.

*A. Training data unit*

Neuro language generator is applicable in virtually every situation in which a relationship between the predictor variables (independents, inputs) and predicted variables (dependents, outputs) exists, even when that relationship is very complex and not easy to articulate in the usual terms of correlations or differences between groups. The input/output training data are fundamental in neural network technology, because they convey the necessary information to discover the optimal operating point.

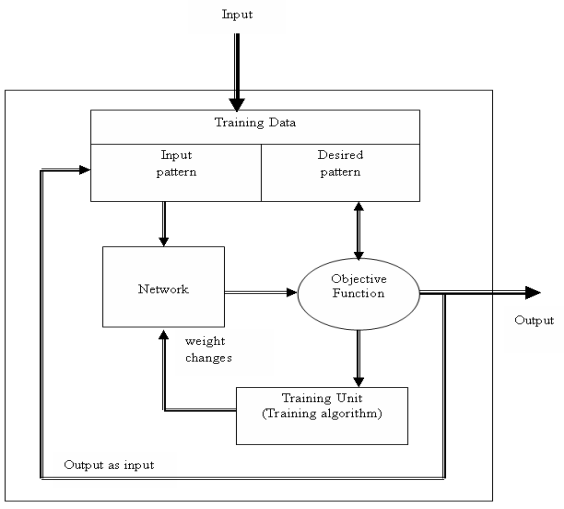


Figure 1. Architecture of Neuro Language Generator

The training data unit consists of input pattern and target pattern. The input and target pattern can be taken from formal language. Initially, the input pattern is the start symbol (initial state of the Turing machine) of the formal grammar. During the training process the current output is changed into input data. The training data unit expands the input pattern to the network unit. The output of the network unit data is compared with the target pattern which is stored in training data unit.The nature of the network processing elements provides the system with lots of flexibility to achieve practically any desired input/output map. Training set can be made directly from formal grammar. Certain grammar of measured values is used as inputs and the value to be predicted is used as required output. An input is presented to the NLG and a corresponding desired or target response set at the output. An error is composed from the difference between the desired response and the system output. This error information is fed back to the system and adjusts the system parameters in a systematic fashion (the learning rule). The process is repeated until the performance is acceptable.

*B. Network unit*

The network structure of neuro language generator is a feedbackward (dynamic). That is, Neuro language generator network has a feedback structure: signals flow from inputs, feed backwards through the same layer, eventually reaching the output units. Such a structure has stable behavior. The important property of NLG is the dynamical properties of the network. If the input pattern is presented, the unit computes its activation just as in a feed forward network. However its net input now contains a term which reflects the state of the network. Activation values of the units undergo a relaxation process such that the neural network will evolve to a stable state in which these activations do not change anymore. The change of the activation values of the output neurons is significant, such that the dynamical behaviour constitutes the output of the neural network.

In Neuro Language Generator, network activations and signals are in a flux of change until they settle down to a steady state. It has feedback paths from their outputs back to their inputs, the response of such networks is dynamic. (ie) After applying a new input, the output is calculated and fed back to modify the input. The output is then recalculated and the process is repeated again and again. In NLG, successive iterations produce smaller and smaller output changes until eventually the outputs become stable. For many traditional networks, the process never ends and such networks are unstable.

Network unit computes the weighted sum of its inputs and produces the actual output. If the actual output and target output match, then the NLG recognizes the input pattern and halt. Otherwise, training process continues until the network produces a constant value. The internal representation of Neuro Language Generator 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 generator) 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.

*C. Objective function unit*

NLG receives a number of inputs (either from original data, or from the output of NLG). Each input comes through a connection that has a strength (or weight); these weights correspond to synaptic efficacy in a biological neuron. Each neuron also has a single threshold value. The weighted sum of the inputs is formed, and the threshold subtracted, to compose the activation of the neuron.

When the NLG is executed, the input variable values are placed in the input units, and then computational and output layer units are progressively executed. Each of them calculates its activation value by taking the weighted sum of the outputs of the units in the computational layer, and subtracting the threshold. The activation value is passed through the objective function to produce the output of the neuron. When the entire network has been executed, the outputs of the output layer act as the output of the entire network.

As shown in the figure 2, the objective function minimizes the error in the training process. That is, the objective function finds the difference between actual output and target output. The objective function of NLG is given by:

$$f_{n+1}(x) \begin{cases} 0, & \text{if } d > 0 \\ f_n(x), & \text{if } d < 0 \\ 1, & \end{cases} \quad \ldots(1)$$

where x is the number of iterations in the training process and x lies between 0 and n, d is the difference between actual output and target output. i.e., d = length(actual output) – length (target output).

If $f_{n+1}(x) = 0$, then the given input pattern is not the correct sentence of the given language and the NLG is in rejecting state. If $f_{n+1}(x) = 1$, then the given input pattern is the correct sentence of the given language and the NLG is in accepting state. Otherwise the training process continues.

*D. Training unit*

The training unit consists of set of weight productions and NLG training algorithm. The neuro language generator gathers formal language patterns, and then invokes training algorithms to automatically learn the structure of the data. Neuro language generator is a recurrent learning to perform a function (an input/output map) from data. Recurrent learning means that the system parameters are changed during recurrent operation, normally called the training phase.

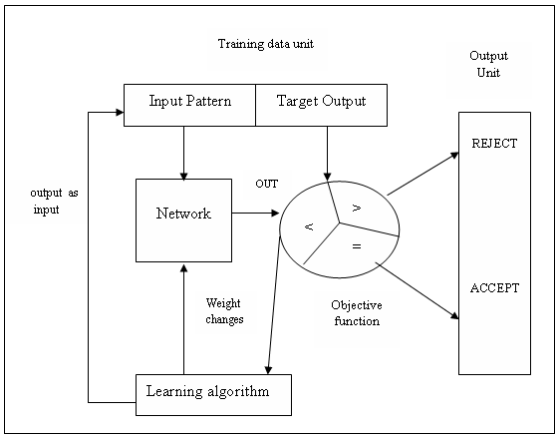


Figure 2. Structure of Neuro Language generator with output

Neuro language generator is built with a systematic step-by-step procedure to optimize a performance criterion or to follow some implicit internal constraint, which is commonly referred to as the learning rule. Supervised training in which the network is trained by providing it with input and matching output patterns. These input-output pairs are provided by training data unit (from external environment), or by the system which contains the network. Supervised training is based on the target value or the desired outputs. During training the network tries to match the outputs with the desired target values. The method of training a neural network is trial and error. The network uses the trial and error method and produces results.

*Training algorithm for Neuro language generator:*

*Step1 : Load the input pattern and target pattern to training data unit.*
*Step 2 : Training data unit expands the input pattern to the network unit.*
*Step 3 : Network unit determines the OUT value, where OUT is the actual output of NLG.*
*Step 4: Calculate the objective function, d = length (OUT) – length (target pattern)*
*Step 5: a) If d > 0, then the given target pattern is not the correct sentence of the given formal language.*
*b) If d = 0, then the OUT value contains only terminal symbols, the given target is correct and NLG is in accepting state.*

*c) If d < 0, then the OUT value is treated as new input of NLG.*

*Step 6: Apply new weight production from the training unit.*

*Step 7: Training process continues and step 2 to step 7 are repeated.*

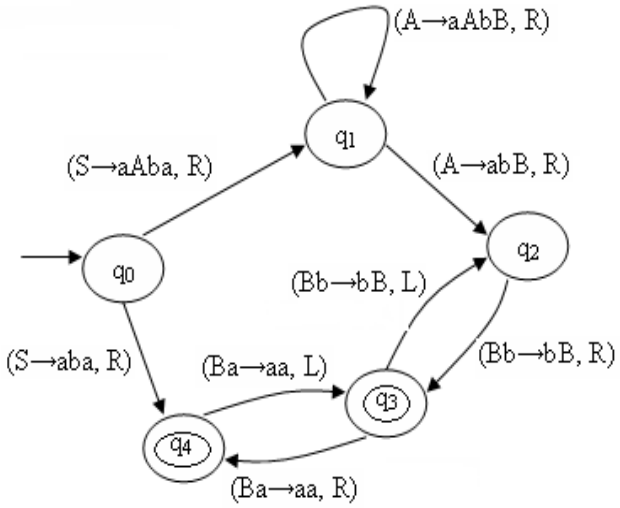


Figure 3. State transition diagram of TM for the language $\{ a^n b^n a^n \}$

The adjustment of synaptic weights during the learning reduces the output error as the learning trails increase. This interaction is externally imposed rather than occurring within the neural structure. When a new input pattern is presented, the neuron outputs are computed as usual, but because these outputs are fed back as inputs to the system, the activations of neurons get subsequently modified, leading the network to a new system. The next state of the network is thus a function of the current input and the present state of the network.

*E. Output unit*

The output unit is the result of NLG. The output unit consists of two outputs either 'ACCEPT' or 'REJECT'. If the output unit produces the result 'ACCEPT', then the NLG recognizes the target pattern. If the output unit generates the result 'REJECT', then the NLG does not accept the target pattern. Therefore, NLG generates recursive language (recursive language produces either accept or reject).

## VI. EXPERIMENTAL RESULTS

The stability and dynamic nature of network which are achieved through the architecture of NLG are proved by an example. The structure of NLG also generates recursive language. The simulation result of the NLG is produced using the formal language $L=\{a^n b^n a^n\}$. For experimental purpose a set of correct patterns and a set of incorrect patterns are taken. The set of correct patterns {aba, aabbaa, aaabbbaaa, aaaabbbbaaaa, aaaaabbbbbaaaaa } are taken from L and set of incorrect patterns {abba, aaabaabb, bbbaaabbb, abbababababbaaab, aabbaabbaabbaabbaabba} are taken from L' where L' is a complement of L. If the correct input pattern $x_1$ is applied for training it accepts $x_1$ and halts where $x_1 \in$ L. If the incorrect input pattern $x_2$ is applied for training it rejects $x_2$ and halts where $x_2 \notin$ L ($x_2 \in$ L'). The language is recursive if there exists a Turing machine that accepts every word in L and rejects every word in L'.

The NLG architecture consists of training data unit, network unit, objective function unit, training unit and output unit. The training set is taken for network training process. The training set consists of input pattern X = {S}, weight vector W={ $w_1$=(S → aAba, R), $w_2$= S → aba, R),$w_3$ = (A → aAbB, R), $w_4$ = (A → abB, R), $w_5$ = (Bb → Bb, R), $w_6$ = (Bb → Bb, L), $w7$ = (Ba → aa, R), $w_8$ = (Ba → aa, L)} and target pattern T = { $a^3b^3a^3$}. The training data unit consists of input pattern {S} and target pattern (desired output) $\{a^3b^3a^3\}$ for training. If the input pattern X={S} and target pattern T=$\{a^3b^3a^3\}$ are applied to the network unit in NLG. Network unit structure is feedback network which has dynamical property. The network unit works until it reaches the equilibrium point using the training parameters, input patterns, weight vector and OUT value. The structure of the network unit is designed using the state transition diagram of Turing machine. The state transition diagram is shown in figure 3.

The network unit can be demonstrated by means of a state transition diagram which gives the states of the network and their training parameter together with the target pattern from one state to another. The network either remains in the same state or moves to the next state. The function of training unit is based on the weight vectors and algorithm in the training unit. The training algorithm is constructed by the transition function of the language $L = \{a^n b^n a^n\}$. The figure 4 demonstrates architecture of NLG for the language $\{ a^n b^n a^n \}$.

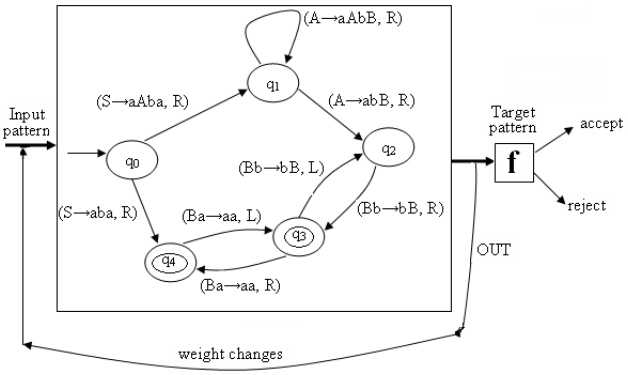


Figure 4. Architecture of NLG for the language { $a^nb^na^n$}

Initially, the input pattern X={S} and the target pattern T = {$a^3b^3a^3$} are applied to the network unit. This unit determines the OUT value. If objective function f compares the length of OUT and length of target pattern, then d = length (OUT) – length (target pattern) where d is the difference between OUT and target pattern. If d<0 then the OUT value is taken as input pattern and the weights are adjusted and trained in the network unit using training algorithm from training unit. This process continues until the target pattern and the OUT value is same. If d=0 and if the OUT value contains non-terminal with terminal symbol then the training process continues until the non-terminal becomes terminal. If d>0, the NLG goes to wrong direction as the given input pattern is an incorrect pattern which does not belong to L. Thus the training process is terminated and the NLG halts.

TABLE I. SIMULATION OF CORRECT WORD PATTERN IN NLG

| Iteration | Input/current pattern X | OUT (desired output) | Weight adjusted W | value of d | Remarks |
|---|---|---|---|---|---|
| 1 | S | aAba | aAba | d = - 5 | d < 0 |
| 2 | aAba | aaAbBba | aAbB | d = -2 | d < 0 |
| 3 | aaAbBba | aaabBbBba | abB | d = 0 | d = 0, OUT contains terminal with non terminals |
| 4 | aaabBbBba | aaabbBBba | bB | d = 0 | d = 0, OUT contains terinal with non terminals |
| 5 | aaabbBBba | aaabbBbBa | bB | d=0 | d = 0, OUT contains terinal with non terminals |
| 6 | aaabbBbBa | aaabbbBBa | bB | d=0 | d = 0, OUT contains terinal with non terminals |
| 7 | aaabbbBBa | aaabbbBaa | aa | d=0 | d = 0, OUT contains terinal with non terminals |
| 8 | aaabbbBaa | aaabbbaaa | aa | d=0 | d=0, OUT contains all terminal symbols, OUT and target are equal, NLG halts |

The correct word pattern T = {$a^3b^3a^3$} is taken from L = {$a^nb^na^n$} for simulation. In table I, iteration$_1$ denotes that the input pattern {S} is applied to the network unit. {S} is adjusted by $w_1$ = {aAba}. The network unit produces OUT = aAba. In table I, if d < 0, then the OUT value is fed back again considered as input pattern and the training process continues from iteration$_1$ to iteration$_2$. Iteration$_3$ shows d=0 and OUT value has both terminal and non-terminal. Since the OUT value contains both terminal and non-terminal the training process continues till the OUT values becomes terminal. The training continues from iteration$_3$ to iteration$_7$. In iteration$_8$, the OUT value contains only terminal symbols (OUT = aaabbbaaa). The OUT value and target pattern T = {aaabbbaaa} are equal. Thus the NLG accepts the given correct word pattern {aaabbbaaa} and halts.

The incorrect word pattern {$b^3a^3b^3$} is taken for simulation. The incorrect pattern {$b^3a^3b^3$} is fixed as target pattern T. The training procedure is same as training procedure of the correct word pattern. The input pattern X={S} and the target pattern T = {$a^3b^3a^3$} are applied to the network unit. In table II, iteration$_1$ shows that the input pattern {S} is applied to the network unit. {S} is adjusted by $w_1$={aAba}. The network unit produces OUT = aAba. In table II if d<0, then the OUT value is given as input to the network and the training process continues from iteration$_1$ to interation$_2$. Iteration$_3$ shows d=1 means d>0. When d>0, then the training algorithm of NLG implies that the training process leads to indefinite loop. Therefore the training process of NLG should be terminated and the NLG rejects the given incorrect word pattern {$b^na^nb^n$}.

TABLE II. SIMULATION OF INCORRECT WORD PATTERN IN NLG

| Iteration | Input/current pattern X | OUT (desired output) | Weight adjusted W | value of d | Remarks |
|---|---|---|---|---|---|
| 1 | S | aAba | aAba | d = - 5 | d < 0 |
| 2 | aAba | aaAbBba | aAbB | d = -2 | d < 0 |
| 3 | aaAbBba | aaaAbBbBba | aAbB | d = 1 | d > 0, OUT contains terinals with non terminals, rejects and halt |

From the above discussion it is concluded that if correct input word pattern is given to the NLG it accepts and halts. If incorrect word pattern is given to the NLG it rejects the incorrect word pattern and halts. Since the NLG halts for both correct and incorrect pattern applied to it the NLG achieves stability and it accepts recursive language. The NLG is efficiently trained by single layer feedbackward neural network. As it is a dynamical system, changes are made at the run time of the machine. It solves halting problems and never goes to indefinite loop or cycle. The most important notable thing is that the machine achieves stability. As it solves most of the problems of the formal language, it is considered better than any other conventional feedforward networks.

## VII. CONCLUSION

Conventional dynamical language generators are neural finite automata and neural pushdown automata. These conventional methods are feedforward neural networks. The lack of feedback network ensures that the network is unconditionally stable. Non-recurrent networks have a repertoire of behavior that is limited compared to their recurrent structure. Neuro Language Generator overcomes the defects faced by the traditional neural finite automata and neural PDA. Neuro Language Generator is based on neural network with a finite-state machine which is a Turing machine. It is feedback neural networks which is dynamic or recurrent networks. The Neuro Language Generator can offer great computational advantages over purely static neural networks. The NLG is a language generator.

## REFERENCES

[1] Alquezar, R. and Sanfeliu, A. (1995). An algebraic framework to represent finite state automata in single-layer recurrent neural networks. Neural Computation, 7(5):931-949.

[2] Blair, A. and Pollack, J. B. (1997). Analysis of dynamical recognizers. Neural Computation, 9(5):1127 - 1142.

[3] 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.

[4] Draye, J., Pavisic, D., Cheron, G., and Libert, G. (1995). Adaptive time constants improve the prediction capability of recurrent neural networks. Neural Processing Letters, 2(3):12 - 16.

[5] 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.

[6] Goudreau, M., Giles, C., Chakradhar, S., and Chen, D. (1994). First-order vs. second-order single layer recurrent neural networks. IEEE Transactions on Neural Networks, 5(3):511 - 513.

[7] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. Neural Networks, 2(5):359 - 366.

[8] Hopcroft, J. E. and Ullman, J. D. (1979). Introduction to automata theory, languages, and computation, Addison, Wesley, Reading, MA.

[9] Kolen, J. F. (1994). Fool's gold: Extracting finite state machines from recurrent network dynamics. In Cowan, J. D., Tesauro, G., , and Alspector, J., editors, Advances in Neural Information Processing Systems 6, pages 501 - 508, San Mateo, CA. Morgan Kaufmann.

[10] Kremer, S. C. (1999). Identification of a specific limitation on local-feedback recurrent networks acting as mealy-moore machines. IEEE Transactions on Neural Networks, 10(2):433 - 438.

[11] Peter Tino, Bill G. Horne, C. Lee Giles (1998). Finite State Machines and Recurrent Neural Networks Automata and Dynamical Systems Approaches. Technical Report, UMIACS-TR-95-1 and CS-TR-3396, Institute for Advanced Computer Studies, University of Maryland.

[12] 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.

[13] 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.

## AUTHORS PROFILE

**Dinadayalan. P.** obtained his M.C.A degree in the year 1996 and M.Tech degree in Computer Science and Engineering in the year 2000 from Pondicherry University, India and M.Phil. degree in Computer Science from Manonmaniam Sundaranar University, Tirunelveli, India in the year 2002. He is presently pursuing his Ph.D in Computer Science from Vinayaka Missions University, Salem, India under the guidance of Dr. R. Vasanthakumari. He has published papers in national and international conferences conference and in many international journals in the area of Artificial Neural Networks and Theory of Computer Science. He is working as Assistant Professor in the Department of Computer Science, Kanchi Mamunivar Centre for Postgraduate Studies, Puducherry, India.

**Gnanambigai Dindadyalan** obtained her M.Sc. degree in Computer Science from Pondicherry University, India in 1999 and M.Phil. degree in Computer Science from M.S. University, Tirunelveli, India in 2002. She is presently pursuing her Ph.D in Computer Science from Vinayaka Missions University, Salem, India under the guidance of Dr. R. Vasanthakumari. She has published papers in national and international conferences in the area of Neural Networks and Automata theory. She is working as Assistant Professor in the Department of Computer Science, Indira Gandhi College of Arts and Science, Puducherry, India.

**Dr. R. Vasanthakumari** obtained her Ph.D. degree from Pondicherry University, in 2005. She is working as a Principal in Government College, Puducherry, India. She has more than 27 years of teaching experience in P.G. and U.G colleges. She is guiding many research scholars and has published many papers in national and international conference and in many international journals.