# Different techniques to Evolve Neural Network for Language processing

Rakesh Kumar
Department of computer science
Assam University
Silchar
rakesh_rbl@rediffmail.com

B.S.Purkayastha
Department of computer science
Assam University
Silchar

**Abstract**

One approach used by researchers trying to develop computer systems capable of understanding natural languages is that of training a neural network (NN) for the task. For this type of approach, one of the key questions becomes how to best configure NN parameters such as topology, learning rates, training data, and other. How to choose values for these parameters is still an open question, especially since the effect these variables have on each other is not completely understood.

Genetic algorithms (GA) are particularly well suited for finding optimal combinations of parameters, since they make no assumption about the problem being solved. Different NN configurations are coded as genomes, which have a fitness function based on how well they can solve a particular task. Genomes are paired and recombined in the hope that the offspring of good solutions will be even better.

**Keyword**

Evolving neural network, Word sense disambiguation , Neural Network.

## GA for NN Optimization

In recent years researchers have used genetic algorithm techniques to evolve neural network topologies. Although these researchers have had the same aim in mind (namely, the evolution of topologies that are better able to solve a particular problem), the approaches they used varied greatly.

For example, de Garis (1996) evolved NN by having a series of growth commands give instructions on how to grow connections among nodes. Each node in the network processed signals that told it how to extend its synapses. When two different synapses reached each other, a new node was formed. The genetic algorithm was responsible for evolving the sequence of growth commands that controlled how the network developed.

Fullmer and Miikkulainen (1991) developed a GA coding system where pieces of a genotype went unused, imitating biological DNA processing. Only information stored between a Start marker and an End marker was used to generate networks. The number of correctly configured Start-End markers defined how many hidden nodes the network would have. In addition, information between these Start-End markers defined how the nodes were connected to each other. The meaning conveyed by each position in the used part of the genome depended on its distance from its corresponding Start symbol.

For example, the genome shown in figure 1 would generate two nodes, one for string S,a,1,b,5,a,-2,E and another for string S,b,0,a,3,E , which wraps around the end of the genome. Node a had an initial activation of 1(because of substring S,a,1), is connected to node b with a weight of 5 (because of substring b, 5), and to itself with a weight of -2 (because of substring a, -2). Node b had an initial activation of 0 (because of substring S,b,0) and a connection to node a with a weight of 3(because of substring a,3).

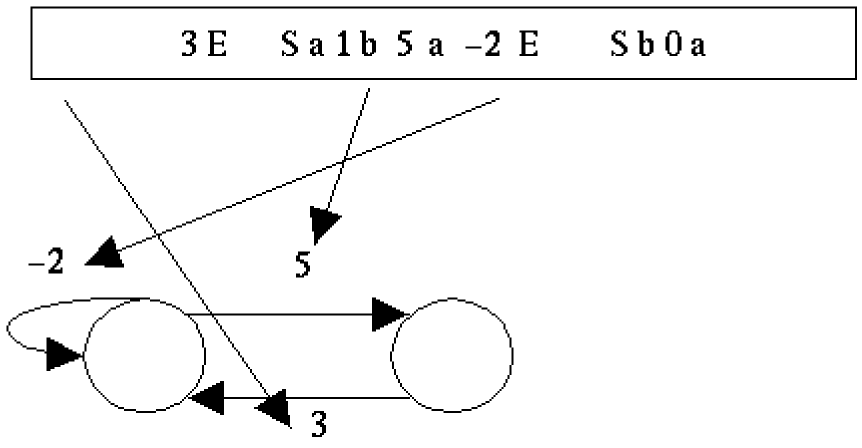


Figure 1

## Genetic Definition of NN Topology

Each NN in this system has 75 hidden nodes between the input and output layers. These 75 nodes are divided into N hidden layers, where N is a number between 1 and 30. The exact number of hidden layers is determined by the first gene of the corresponding genome. This position stores a random floating point number, with a value between 0 and 1. To determine how many hidden layers a network has, the value of this gene is multiplied by 30, and rounded to the next highest integer. If the result of this rounding up is 31, the network uses 30 hidden layers.

The number of hidden nodes in each of these hidden layers is also determined by the network's corresponding genome. The genome has 30 genes used to code the relative worth of each of the possible hidden layers. Once the number of hidden layers is determined to be N using the process described above, the N layers with the highest relative worth are identified. The 75 available hidden nodes are distributed among each of these N hidden layers according to each layer's worth relative to the sum of all N worth values.

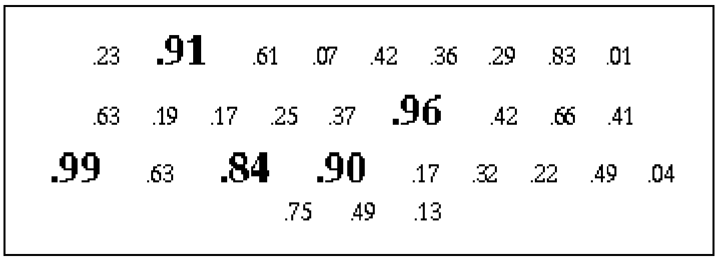

.23 **.91** .61 .07 .42 .36 .29 .83 .01
.63 .19 .17 .25 .37 **.96** .42 .66 .41
**.99** .63 **.84** **.90** .17 .32 .22 .49 .04
.75 .49 .13

Figure 2

For example, if a genome had genes 1-30 as illustrated in figure 2, and it had already been determined that it would have five hidden layers (as described above), the five layers to use are those indicated in bold. Since the sum of these five genes is 4.6, the first hidden layer would have (75*.91/4.6 = ) 14 nodes. The other four hidden layers would be allocated hidden nodes in the same way.

The connections between layers are also determined by the network's genome. For each of the thirty possible layers, there is a gene that indicates where the layer takes its input from. Each of these genes stores a random oating point value between 0 and 1. To determine where each hidden layer takes its input from, its takes-its-input from gene value is multiplied by N+2 (where N is the number of hidden layers this network will have, as determined by the procedure outlined previously), and rounded to the nearest integer. The resulting number points to which layer this one takes its input from. We multiply by N+2 to allow a hidden layer to take its input from any of the N hidden layers, as well as either the input or the output layer. A value of 1 would mean the layer takes its input from the input layer. A value of N+2 would mean the layer takes its input form the output layer. For values between 2 and N+1, the layer would take its input from the layer with the (N-1)th highest relative worth.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| .12 | **.69** | .42 | .89 | .01 | .44 | .91 | .56 | .27 | .04 |
| .22 | .36 | .07 | .45 | **.24** | .33 | .11 | .41 | .93 | **.01** |
| **.37** | **.21** | .61 | .54 | .77 | .89 | .51 | .55 | .78 | .49 |

Figure 3

For example, if the same genome used for the example above had genes 31-60 as illustrated in figure 3, we would look at the corresponding 5 takes-input-from genes, shown in bold in figure 3. Multiplying each of the selected genes by 6, we would obtain 4.14, 1.44, .06, 2.22, and 1.26. This would mean that hidden layer 1 would take its input from hidden layer 4, hidden layer 2 would take its input from hidden layer 1, hidden layer 3 would take its input from the input layer, hidden layer 4 would take its input from hidden layer 2, and hidden layer 2 would take its input from hidden layer 1.

Where each layer sends its output is determined in a similar way, using positions 61-90 of the genotype. Each of these genes stores a random floating point value between 0 and 1. To determine where each layer sends its output, its sends-output-to gene value is multiplied by N+1 and rounded to the nearest integer. The resulting number points to which other layer this one will send its output to. We multiply by N+1 to allow for hidden layers sending their output to any of the N hidden layers, as well as to the output layer. A value of N+1 would mean the layer sends its output to the output layer.

For values between 1 and N, the layer sends its output to the layer with the Nth highest relative worth. No layer sends its output back to the input layer.

## Word Sense Disambiguation

Their tolerance for noise, their ability to generalize, and their well-known suitability for classification tasks [1] make NNs natural candidates for approaching WSD. A large number of successful applications demonstrates that NN design is improved by considering it in conjunction with EAs, since neural and evolutionary techniques can be combined in a synergetic way [14]. The most promising approach to using EAs to design and optimize NNs is the one which jointly evolves network architecture and weights: a completely functioning network can be evolved without any intervention by an expert [1].

The algorithm presented in Section 3 evolves a specific NN specializing in disambiguating one given target polysemous word. The same process must be repeated for each polysemous word one is interested in disambiguating. Since our aim here is to demonstrate the feasibility of this approach, we will focus on a small, but representative, set of target polysemous words. We used IXA Group's web corpus [2], which comprises all WordNet noun senses. Its construction is inspired by the "monosemous relatives" method [10]. The method usually relies on information in WordNet in order to retrieve examples from large corpora or the web. The advantage of using this corpus instead of other traditional corpora specifically designed for WSD is that, due to the way the corpus is constructed, we get an extensive coverage of word usage in all domains. Furthermore, since the corpus is not tagged by hand, it gives wider guarantees of accuracy and objectivity.

Two kinds of representations commonly used in connectionism are distributed [9] and localist schemes [7]. The major drawback of the latter is the high number of inputs needed to disambiguate a single sentence, because every input node has to be associated to every possible sense; in fact, this number increases staggeringly if one wants to disambiguate an entire text.

Examples of distributed schemes are microfeatures [13] and the word-space model [12]. Distributed schemes are very attractive in that they represent a context as an activation pattern over the input neurons. Therefore, unlike with localist schemes, the number of input neurons required does not have to equal the size of the vocabulary. On the contrary, there is an intrinsic idea of compression, whereas each input neuron encodes a given feature of a word or sentence and thus can be reused to represent many different words or senses. Of course, the more the input patterns are compressed into a low-dimensional space, the more information is lost. However, despite such information loss, enough information may still be there to allow meaningful processing by the NN.We used two distributed representation schemes, both based on the way words are written, corresponding to two different degrees of compression: (i) a positional scheme, whereby 156 input neurons, divided in six groups of 26 input neurons each, encode the first six letters of a word, after deleting as many vowels, starting from the last one but except the first one, as required to reduce the word to six letters, because consonants carry a heavier distinctive load; if, even after deleting all the vowels but the first, the word is still more than six letter long, only the first six letters are actually represented (thus representation would be encoded as REPRSN, but cotton would be encoded as COTTON); if the word is shorter than six letters, the representation is padded with blanks; and (ii) a letter count scheme, which brings the distributed representation idea to one extreme, by using the number of occurrences of the 26 letters of the alphabet as features. In both cases, the activations of the input neurons are obtained by summation of the activation patterns representing the words occurring in a given context, excluding the target word, after removing stop words and stemming the remaining words. Additional fields of the training set (one for each output neuron) correspond to the n senses of the target word. They are all set to zero except the one corresponding to the correct sense. For example, starting from a sentence 'part aqueduct system', where the target word, tunnel, has two senses, (1) "a passageway through or under something" and (2) "a hole made by an animal", the associated record would be • for the letter count encoding,

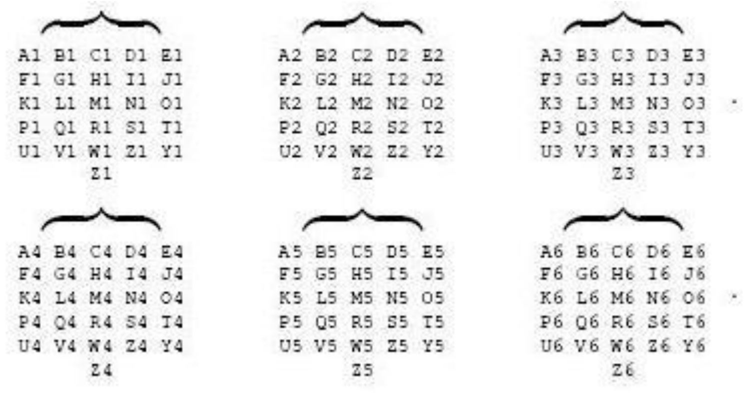
2 0 1 1 2 0 0 0 0 0 0 0 1 0 0 1 1 1 2 3 2 0 0 0 1 0
->1 0,

in which the first 26 numbers represent the occurrences of the letters of the alphabet, and the last 2 the two output senses (here n = 2); • for the positional encoding, after reducing all words

to their 6-letter representatives 'PART ', 'ACQDCT', and 'SYSTEM',

$$
\begin{array}{ccc}
\overbrace{\begin{matrix}1&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\1&0&0&1&0\\0&0&0&0&0\\&&0&&\end{matrix}} &
\overbrace{\begin{matrix}1&0&1&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&1\\&&0&&\end{matrix}} &
\overbrace{\begin{matrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&1&1&1&0\\0&0&0&0&0\\&&0&&\end{matrix}} \\
\overbrace{\begin{matrix}0&0&0&1&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&2\\0&0&0&0&0\\&&0&&\end{matrix}} &
\overbrace{\begin{matrix}0&0&1&0&1\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\&&0&&\end{matrix}} &
\overbrace{\begin{matrix}0&0&0&0&0\\0&0&0&0&0\\0&0&1&0&0\\0&0&0&0&1\\0&0&0&0&0\\&&0&&\end{matrix}} \\
& \rightarrow 1\ \ 0, &
\end{array}
$$

where the numbers on the left-hand side of the arrow represent the occurrences of the letters in the six positions, i.e., A1, . . . , Z6, here displayed according to the template



The rationale for this type of sentence encoding is that, since every word can be regarded as an activation pattern of the input layer of a NN, the overlap of different patterns from the same sentences allows the network to detect significant information about the context. The success of the experiments presented in Section 4 will serve as an empirical proof that even such extremely compressed representations preserve enough context information to allow disambiguation of word sense

## Conclusion

This paper has presented a mathematical methodology for predicting the effectiveness of a GA in processing NN topology. The methodology is used to predict the effectiveness of several GA coding schemes. These predictions are shown to correspond with actual results. In addition, different ways of representing sentences at a NN output later are presented. Advantages of using non-binary representations are discussed, both from the point of view of expanding sentences that can it can process and efficiency of GA used to evolve them.

## References

de Garis, H. (1996). CAM-BRAIN: The Evolutionary Engineering of a Billion Neuron Artificial Brain by 2001 Which Grows/Evolves at Electronic Speeds Inside a Cellular Automata Machine (CAM), In Lecture Notes in Computer Science - Towards Evolvable Hardware, Vol. 1062, pages 76-98. Springer Verlag, pages. 76-98.

Davila, J. (1999) Exploring the Relationship Between Neural Network Topology and Optimal Training Set by Means of Genetic Algorithms. In International Conference on Artificial Neural Networks and Genetic Algorithms, pages 307-311. Springer-Verlag.

Daavila, J. (2000) A New Metric for Evaluating Genetic Optimization of Neural Networks. In Proceedings of the First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks. pages 52-58. IEEE

Fullmer, B., Miikkulainen, R., (1991). Using marker- based genetic encoding of neural networks to evolve _nite-state behavior. In Proceedings of the first European Conference on Artificial Life, pages. 253-262.

Kitano, H., (1994). Designing Neural Networks using Genetic Algorithm with Graph Generation System. In Complex Systems, 4. pages 461-476.
Weiss, S., Kulikowski, C. (1991). Computer Systems that Learn. Classification and Prediction Methods from Statistics. In Neural Nets, Machine Learning, and Expert Systems. Morgan Kaufmann.

Yao, X. (1993). A review of evolutionary artificial neural networks. In International Journal of Intelligent Systems, 4. pages 203{222.