DISRUPTION ANALYSIS FOR NEURAL NETWORK TOPOLOGY EVOLUTION SYSTEMS

Jaime J. Dávila

School of Cognitive Science
Hampshire College
Amherst, MA 01002
jdavila@hampshire.edu
413-559-5687

ABSTRACT

This paper presents a method for analyzing GA effectiveness for the evolution of neural networks. The analysis is based on the schemata of the (phenotype) neural network being evolved, as opposed to the traditional method of analyzing schemata disruptions at the genotype level. Comparisons between the two types of analysis are made. Empirical data is presented that indicates the greater validity of the analysis at the phenotype level.

1. INTRODUCTION

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.

Evolution of neural networks (NN) by genetic algorithms (GA) involves two steps. In the first one, GA performs genetic manipulation of a genotype. In a second step a NN is created based on a genotype, and evaluated for its ability to solve a predetermined problem. How well the NN can solve the problem is typically used as the fitness of the genotype that defined it.

A wide number of evolution strategies have been used in order to optimize NN topologies. Since discussing even a small percentage of these strategies is outside the context of this paper, the reader is referred to [3] for an overview.

Given that each of these methods involves allowing time for the generation and evaluation of a high number of NN, experimenting with a high number of GA methods can be too computationally expensive. It becomes important, then, to be able to estimate the effectiveness of different evolutionary strategy before making use of them. In this way both the applicability of the method and parameters such as population size and mutation rates can be set according to how each method affects the evolution of NN.

This paper presents a formalism for analyzing GANN systems based on what they do to phenotype schemata. This differs from the most commonly used method used in the GA literature of analyzing disruptions at the genotype level. A mathematical definition of how easy it is to destroy phenotype schemata is presented and used to make predictions. These predictions are then corroborated empirically on variations of the GENDALC system [1]. This system has proved to be successful in evolving NN for context sensitive tasks.

2. PHENOTYPE-LEVEL ANALYSIS OF GANN SYSTEMS

If we view the evolution of NN topologies as a process with the main goal of defining connections between any two nodes, then we can determine their ability to combine building blocks by estimating how likely it is for evolutionary operations to disrupt connection definitions; the less likely it is for connection definitions to be disturbed, the easier it is for the algorithm to combine building blocks present in the current population. An operation like crossover can disrupt a connection definition every time a crossover point is selected between two genes that, taken together, define a connection between nodes of a network. Therefore, how likely it is for crossover to cause this disruption can be estimated by the distance between genes that combine to define any particular connection. If a particular connection is defined by alleles in genes g_i and g_j , then the bigger the distance between g_i and g_j , the bigger the chance that the connection will be disrupted by a crossover operation. Taking a sum of the distance between genes that can define a connection we obtain a total disruption index (TDI) of

$$\sum_{k=0}^{C}\sum_{i=0}^{N}\sum_{j=0}^{N}\left(\left(i-j\right)\right)\times DC\left(k,i\right)\times DC\left(k,j\right) \quad (1)$$

where N is the number of genes, and DC(k, x) is a number between 0 and 1 which indicates the probability that gene x is involved in defining connection k. Notice that this number reflects a global probability of disruption for the complete network, as opposed to for any particular connection. This is motivated by the fact that the more connections a network has, the more likely it is to suffer disruptions.

3. PREDICTIONS ON THE GENDALC SYSTEM

In order to test the formulations presented in this paper, I will be using them to make predictions for three variations of the GENDALC system [1]. These three variations are then used to solve the same task, and their ability to take advantage of characteristics in their initial populations is measured. Empirical results are reported in section 5 of this paper.

3.1 GENDALC system-a

GENDALC SYSTEM-A determines NN topologies by distributing 75 hidden nodes among up to 30 hidden layers, and then determining how these hidden layers are connected to each other. Each topology in the GENDALC system is configured by a genome in a genetic algorithm population. The exact number of hidden layers is determined by the first gene of the corresponding genome. This position stores a 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.

Take, for example, the partially illustrated genome in figure 1, here showing genes 1 through 31. The first gene, with a value of .23, determines that this network will have 7 hidden layers. The relative worth for all 30 layers are represented here by genes 2 through 31. Among these, the seven with highest worth value are layers 18, 14, 1, 21, 7, 20, and 27 (with relative worth values of .99, .96, .91, .90, .84, .84, and .75). The sum of these relative worth values is 6.19, which means that each of these seven layers will have 75*X/6.19 nodes, where X is each layer's relative worth. Therefore, for this example layers 18, 14, 1, 21, 7, 20 and 27 will have 12, 12, 12, 10, 10, 10, and 9 nodes, respectively.

.23	.91	.61	.07	.42	.36	.29	.84	.01
.63	.19	.17	.25	.37	.96	.42	.66	.41
.99	.63	.84	.90	.17	.32	.22	.49	.04
	.75	.49	.13	.71	...			

Figure 1: sample of genes 1-31

The connections between these seven 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 floating 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 with 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 for hidden layers taking their input from any of the N hidden layers, as well as either the input or the output layer. A value of 0 means the layer takes its input from the input layer. A value of N+2 means 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.

Where each layer sends its output to is determined in a similar way, using positions 62-91 of the genotype. Each of these genes stores a floating point value between 0 and 1. To determine where each layer sends its output to 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 means 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.

3.2 GENDALC system-b

In GENDALC-SYSTEM-B each network still has 75 hidden nodes, but they are always divided into 30 hidden layers (that is, there is no gene used to determine how many hidden layers to use). The distribution of these 75

nodes into 30 layers is done based on 30 relative worth values in the genome, as in SYSTEM-A. Connections between these 30 hidden layers is also done as in SYSTEM-A.

If we assume that, in (1), DC(k,x) = 1 for all values of k and x, then TDI(SYSTEM-A) = TDI(SYSTEM-B). In reality, DC(k,x) does not always return 1; It tends to return smaller values under SYSTEM-B than under SYSTEM-A. Notice, for example, that under SYSTEM-A the set of gene sequences that would allow node n to be in layer L has a higher cardinality than under SYSTEM-B, given that SYSTEM-B always has 30 hidden layers, while under SYSTEM-A the number of hidden layers is determined by a gene. This will, in turn, affect which genes are involved in defining a connection between two given nodes, which affects DC(k,x). Under standard random distributions, considering actual values for DC(k,x) would give TDI(SYSTEM-A) > TDI(SYSTEM-B). What this would mean is that it is easier for a particularly good schema to be disrupted by crossover under SYSTEM-A than under SYSTEM-B.

3.3 GENDALC system-c

SYSTEM-C uses the same genes as SYSTEM-B, but the position of the genes has been altered. Instead of having 30 worth values followed by 30 takes-input-from values and then 30 sends-output-to values, SYSTEM-C arranges genes so that the worth, takes-input-from and sends-output-to genes for any one particular layer are in three consecutive positions (positions 3*L, 3*L+1, and 3*L+2, where L is the hidden layer number). Because a gene with a particular functionality will have the same effect on resulting phenomes regardless of its position in the genome, we can discard terms DC(k,i) and DC(k,j) in equation (1), and obtain

$$TDI'() = \sum_{k=0}^{C} \sum_{i=0}^{N} \sum_{j=0}^{N} \left| \left(P(g_i) - P(g_j) \right) \right| \quad (2)$$

where $P(g_a)$ is the position of gene a. Under this definition, TDI(SYSTEM-B) > TDI(SYSTEM-C), since SYSTEM-C minimizes the distance between the relative-worth and takes-input-from genes. This means that crossover is less likely to disrupt a useful schema under SYSTEM-C.

4. A TASK FOR THE GENDALC SYSTEM - NATURAL LANGUAGE PROCESSING

This task was originally presented in [1]. As an overview, a network is asked to receive a sentence one word at a time, and to incrementally build a description of the sentence in its output nodes. For example, if the sentence *the boy ran in the park* is entered, the network should respond by indicating that *the boy* is a noun phrase, and it acts as the agent of the verb *ran* . The network should also indicate that *in the park* is a prepositional phrase modifying the verb *ran*.

Entering a word into the network amounts to activating a single node that represents the given word at the input layer. At the same that this word is entered, semantic nodes that reflect the meaning of the word are activated. For example, to enter the word *boy*, a node that represents that word is activated, as well as nodes that indicate that the word being entered is a proper noun, singular, concrete, and human. In addition, an ID node is set to a value that would allow the network to distinguish *boy* from other words that might have the same semantic identity, such as girl.

The language used in this research is composed of ten nouns: *boy, girl, john, mary, horse, duck, car, boat, park,* and *river*. Available semantic nodes are: human, animal, or mechanical (three mutually exclusive nodes); animate or inanimate (represented by one node, active if the noun is animate); proper (active if true, inactive otherwise); and one ID node. Verbs are entered in a similar way; the node representing that verb is activated, simultaneously with semantic nodes that convey the meaning of that verb. Semantic nodes available for verbs are: present or past tense (two mutually exclusive nodes), auxiliary verb, movement verb, sound producing verb, sound receiving verb, visual receiving verb, and a verb ID node used to distinguish verbs that would be identical otherwise (for example, *ran* and swam would be identical without this last node). In total, there are twelve main verbs (s*aw, swam, swimming, ran, runs, running, is, was, raced, floated, said,* and *heard*) and two auxiliary verbs (*is, was*).

In addition to nouns and verbs, the language has three adjectives (*fast, blue, red*), one article (*the*), one adverb (fast), and three prepositions(*with, in, after*). Each of these is entered in the network by activating an individual node for the word, plus an additional node that indicates which type of word (adjective, article, adverb, or preposition) is being entered.

After each word is entered, the NN is expected to produce a representation of what it understands about the sentence up to that point. For example, after the network sees *the boy* (entered one word at a time; first *the* and then

boy) it should indicate that it has detected a noun phrase that uses an article, and that the noun of this phrase is *boy*. *Boy* in this case is represented by activating output nodes that code human, animate, concrete, and an additional ID node that distinguishes *boy* from other words that otherwise would have identical semantics (such as *girl*).

If the network, immediately after having been shown *the boy* at the input layer, is shown *runs*, it should respond by indicating that it has detected a verb phrase with *runs* as its verb. Indicating that the verb of this verb phrase is runs is done by activating semantic nodes for verb of movement and present tense. In addition, a node ID is activated so that runs can be differentiated from other verbs that would otherwise have identical semantics. At this point the network should also indicate that the first noun phrase detected is the agent of the first verb phrase detected. This is done by activating a np1-agent-of-vp1 node in the output layer.

In the manner described above, then, the network should continue to indicate its understanding of the sentence being entered until an end-of-sentence marker is seen. The fitness of a network is determined by computing the sum of squared errors for all output nodes during the processing of all sentences in the language.

5. EMPIRICAL RESULTS

5.1 Parameters used in evolutionary runs

To verify the effect different coding schemata have on the evolution of NN topologies, I have used the three GENDALC versions outlined previously to evolve NN that are presented the NLP task mentioned above. In order to better measure the effect of crossover, computations were performed with no mutation, and populations of 21 elements. Network fitness values were computed by taking the sum of square errors for all output nodes throughout the presentation of a language of 413 sentences. Training is performed on 20% of this same language. In order to verify consistent performance, all evolved networks were validated by performing 144 bootstrap validation runs per network [2]. Evolutionary runs were repeated 48 times.

5.2 Phenotypic disruptions is the correct point of analysis

The first empirical result to be presented in this section demonstrates that phenotypes (NN) are the correct level at which to analyze schemata disruption, as opposed to the more traditionally used genotypical analysis. To do so, two genomes were identified which used the same genes to define network topologies, but with different alleles. Both of these genomes defined networks with nine hidden layers. The first genome (GENOME-A) defined a NN with 4475 connections, and had a fitness of 37575. The other genome (GENOME-B) defined a NN with 1660 connections, and had a fitness of 73536. Since both of these genomes use the same genes to define their respective networks, genotype level schemata should be equally affected by crossover. The TDI formulations presented in this paper, however, predict that it will be harder for the GA to utilize GENOME-A in its processing, since it has more connections. To test this, genetic runs were seeded with one of these two genomes as one of the elements of the initial population. The parent selection algorithm was biased so that it would always choose the seed element, and crossover was biased so that it would always chose crossover points in part of their genotype level schemata. The chance of mutation was set to 0%. Results of these runs are presented in figure 2.

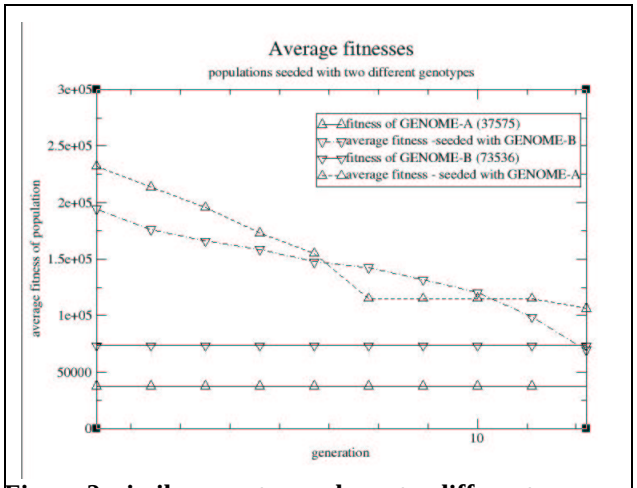

Figure 2: similar genotype schemata, different phenotype schemata.

As can be seen, characteristics of GENOME-B, with fewer connections, spread across the population more effectively than those of GENOME-A.

5.3 Effect of the total number of genes involved in defining a connection

As previously outlined, the only difference between methods SYSTEM-A and SYSTEM-B is the number of genes used. SYSTEM-B reduces the number of genes involved in defining connections by one. SYSTEM-B also has the side effect of reducing the distance between the genes being used, since the additional gene in SYSTEM-A is always in position one. Figure 3 shows the increased effectiveness of SYSTEM-B over SYSTEM-A, as predicted by the smaller total TDI value.

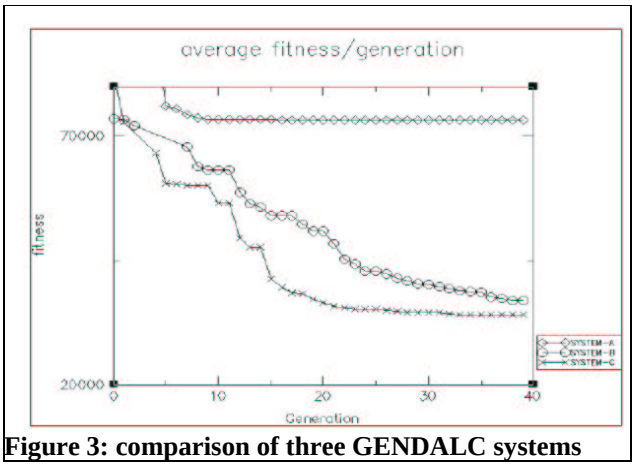

Figure 3: comparison of three GENDALC systems

5.4 Effect of the distance between the genes involved in defining a connection

As seen in figure 3, both VERSION-B and VERSION-C reach comparable average fitness values after 40 generations, but VERSION-C reaches that value in little over 50% the number of generations required by VERSION-B. This is explained by an even lower TDI for VERSION-C, caused by the fact that the genes involved in defining any one connection are now closer to each other, which allows the evolutionary computation to make the best use of building blocks present in the original population.

5.5 TDI estimates can help in selecting evolutionary parameters.

Figure 3 can also be used as an example of what the TDI computation reveals about the relationship between crossover and mutation for different coding schemes. The evolution carried out by VERSION- C has a lower chance of building block disruption by the crossover operation. This means that evolution will depend heavily on operations other than crossover in order to introduce new phenome characteristics into the population. In the absence of any such other operation, crossover will continue to exploit good solutions, but will perform little exploration. This can be seen in figure 3 by the fact that VERSION-C reaches the vicinity of the final average value fairly quickly, but then seems to fail to find anything significantly better.

The need for evolutionary exploration in a system like VERSION-C can be provided by operations such as mutation. To corroborate this, the experiments described above were repeated using a mutation rate of 1%. Results for those runs are presented in figure 4. As can be seen, VERSION-C still reaches better values for average fitness faster than VERSION-B, but now has enough mutation to allow it to explore options not present in the original population, and improve on previous results towards the end of the genetic run.

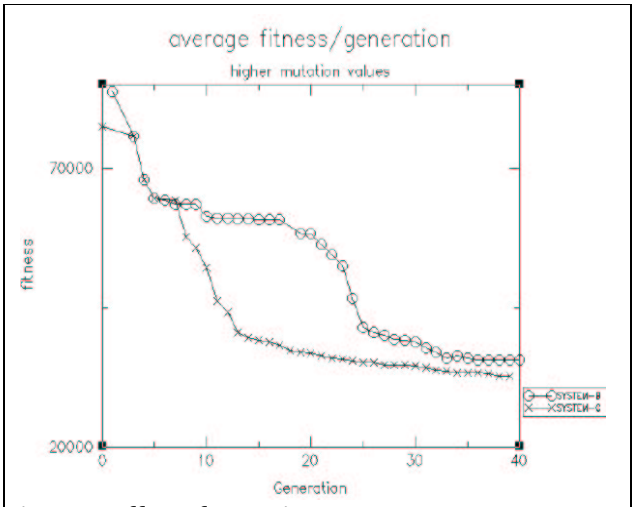

Figure 4: effect of mutation

5.6 The number of nodes in the input and output layers can affect the effectiveness of the evolutionary process

In terms of reducing the number of nodes in the output layer, good schemata spread across the population much quicker when using networks with fewer input and/or output nodes. This is predicted by the formulas presented previously in this paper. Since the total number of connections going to the input and/or output layer is smaller, term C for the first summation in equations (1) and (2) is smaller.

To verify this, the coding method at the output layer for the NLP task used here was modified as follows. Instead of having binary (and mutually exclusive) nodes representing noun properties such as human, animal, mechanical and inanimate, these four nodes are substituted with a single type-of-noun node, with activations ranging between 0 and 1. Something similar can be done to describe verbs, substituting nodes for verb-of-movement, verb-of-sound-producing, verb-of-sound-receiving, etc., with a single type-of-verb node. Another change similar to this can be done with the NP1-MODIFIES-NP2, NP1-MODIFIES-NP3, NP1-MODIFIES-VP1, NP1-MODIFIES-VP2 nodes; they can be substituted with a single node that indicates which phrase this one phrase is modifying. This reduces the number of output nodes from 106 to 36. Results of a run using this coding scheme are illustrated in figure 5. As can be seen, this coding allows the GA to spread the characteristics of the best individual much faster than the previous method.

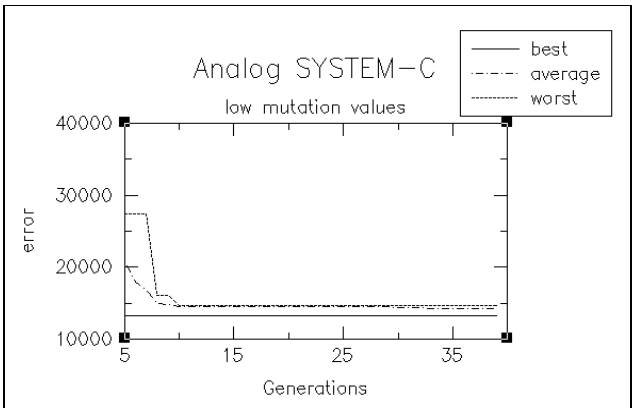

Figure 5: effect of input/output layer size

5.7 TDI formulas can be useful without requiring complete computations

It is important to note that there is no need to completely evaluate TDI values for particular GANN systems before the formulas presented in this paper become useful. For example, values for DC(k,i) in equation (1) do not need to be obtained before comparisons between two different systems can be made. As was done in this paper, knowing how these values differ relative to each other for two systems can be enough to determine the relative likelihood of schemata disruption. This becomes relevant in light of the fact that complete evaluation of TDI values might be complex.

6. FUTURE RESEARCH

Extended empirical tests of the model presented in this paper will allow to better understand the effect of using different operations during evolution for different representational models. For example, SYSTEM-C might benefit from even higher mutation values than used here, and this property should be related to actual values for TDI. Although, as mentioned in the previous section, predictions can be made without having absolute TDI values, a complete mathematical description for all terms in TDI formulas would allow to make better predictions regarding the feasibility of different coding schemes, and provide better guidance regarding the balance between crossover and mutation. This would allow for better comparisons between TDI values and disruptions observed in empirical tests.

7. CONCLUSION

This paper has presented a way of evaluating coding schemes for genetic algorithms used to optimize neural network topologies. The methodology is based on the likelihood that crossover operations can effectively construct on useful phenotype-level building blocks present in initial populations. By doing so it estimates how easy it is for an evolutionary model to find good solutions to the problem at hand, while at the same time providing information regarding how to best balance exploration with exploitation in a genetic system. The method presented here differs from other commonly used methods in that it analyses schemata disruptions at the phenotype level, as opposed to the typically used genotype level analysis. Empirical results show that the phenotype is the correct level for analysis.

This methodology results useful in designing a system that can effectively perform hard natural language processing tasks, and can in theory be extended to any GA/NN system.

8. REFERENCES

[1] 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.

[2] 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.

[3] X. Yao. “Evolving artificial neural networks.” *Proceedings of the IEEE*, 87(9):pp. 1423--1447, 1999.