Genetic Evolution of Neural Networks that Remember

Jaime J. Dávila
Hampshire College
School of Cognitive Science
Amherst MA 01002

Abstract - The GENDALC system has been previously used to evolve NN topologies for natural language tasks. This paper presents results on additional tasks that require remembering and processing of previous input patterns. These results indicate that GENDALC is particularly well suited for tasks that require remembering.

I. INTRODUCTION

The GENDALC (Genetic Evolution of Neuron Distribution and Layer Connection) system has been previously used to evolve topologies for natural language processing tasks [1]. Neural networks evolved in those experiments performed better than other commonly used topologies such as fully connected networks, Simple Recurrent Networks, Frasconi-Gori-Soda networks (FGS), and Narendra-Parthasarathy networks [2]-[4]. In these experiments words were entered one word at a time, thus requiring that the NN remember words no longer at the input layer in order to correctly process a sentence. In addition, the GENDALC system can also be used to optimize other NN parameters such as learning function to be used, values for different learning parameters, and type of training data to use during training.

In order to determine the general ability of the GENDALC system to evolve solutions to different problems, it was presented with a problem in a different domain. In this case, the evolved networks were required to imitate the behavior of infants described by Spelke, Breinlinger, Macomber and Jacobson [5]. This task is discussed in more detail in section III of this paper.

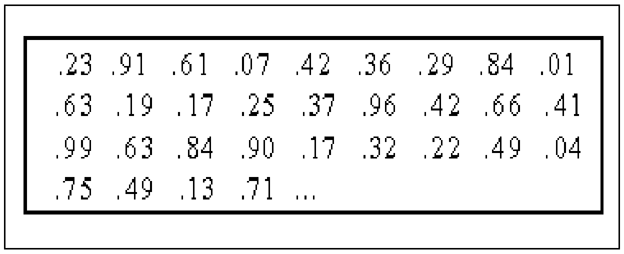
II. DESCRIPTION OF THE GENDALC SYSTEM

The GENDALC system 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: Genes 1-31 for a sample genome.

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.

III. DESCRIPTION OF THE TASK FOR THE NN

The task for the NN is based on experiments done by Spelke, Breinlinger, Macomber and Jacobson on children [5]. In their experiments, they habituated 2.5 to 4 month old babies to a ball being dropped to the floor behind an occluding screen. After the ball fell to the floor, the screen was removed, revealing the ball on the floor. Once the babies were habituated to this falling ball, a second horizontal object was placed above the floor. Then both the floor and the horizontal barrier were occluded by a screen, and the ball was once again dropped from above. When the screen was removed, the ball was revealed to be either on the floor or on the horizontal barrier. By measuring how long the infants looked at the scenes, as well as other physical cues, the researchers estimated the level of surprise of the children. Experiments showed children found it surprising to see the ball below the barrier, even though that was more consistent with the habituating scenes. This seemed to indicate that children at a very early age recognize the solidity constraints of physical objects.

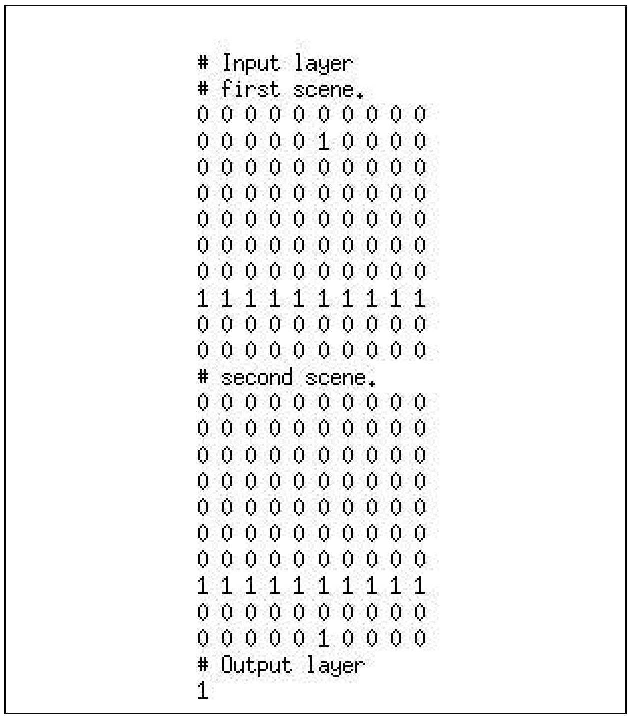
The task for the NN in this experiment was to imitate the behavior described above. The input to the NN was a representation of the initial and final position of the ball and barrier it ‘saw.’ The output of the NN needed to be ‘1' if the scene sequence is surprising (i.e., the ball seems to have gone through the barrier) or ‘0' when it was not surprising. In order to better corroborate GENDALC’s ability to evolve topologies for different tasks, three different input formats were used.

A. Format A

The first format used represented the position of the ball and barriers with simple coordinates. Therefore, only four numbers were needed to represent a scene. Both the initial and final scene were shown at the same time, for a total of eight input nodes.

B. Format B

The second input format represented each scene as a 10 by 10 matrix into which the position of the ball and the barrier were drawn. The ball was represented by a single input set to 1 at the ball’s position. The barrier was represented by a horizontal row of 10 input nodes set to 1's at the barrier’s position. All other input nodes were set to 0. Once again, both the initial and final scene were presented simultaneously, which required 200 input nodes. An example of this format is shown in figure 2. Lines starting with ‘#’ are ignored by the NN simulator used.

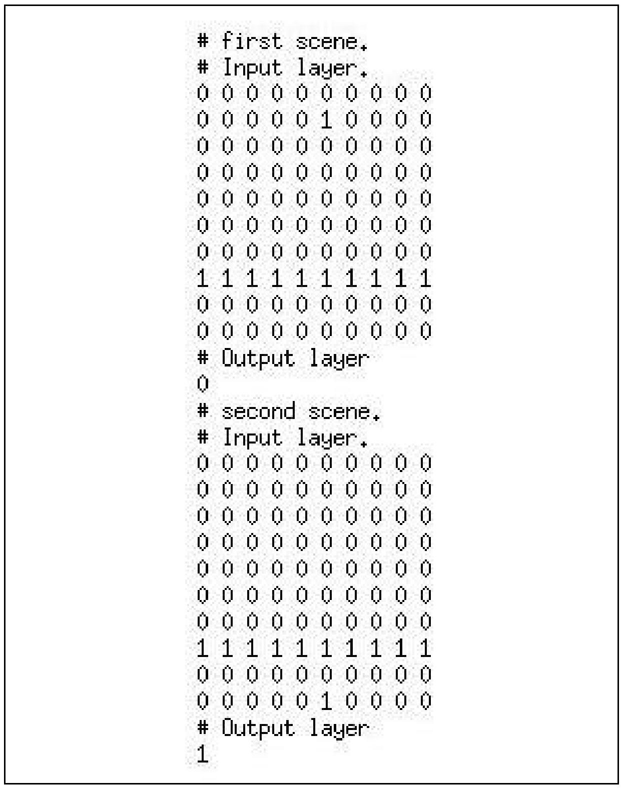


```
# Input layer
# first scene.
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
# second scene.
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
# Output layer
1
```

Figure 2: Example of format B input.

C. Format C

The third input format represented the balls and barriers the same way as format B, but only one scene was presented at a time. The output after the first scene was always of no surprise (‘0'). The output of the NN needed to be ‘1' if the scene sequence was surprising (i.e., the ball seemed to have gone through the barrier). When it was not surprising, the output of the NN needed to be ‘0.’ Figure 3 shows an example of input format C.



```
# first scene.
# Input layer.
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
# Output layer
0
# second scene.
# Input layer.
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
# Output layer
1
```

Figure 3: Example of format C input.

IV. EXPERIMENT PARAMETERS

The NN described in section II were evolved using a basic genetic algorithm (GA). Each network was trained for 500 epochs with 400 scenes of the type described in section III. Then its fitness was determined by evaluating its performance when processing 2000 scenes. The sum of square errors for the single output was computed and assigned as the network's fitness.

The NN aspect of the software runs was performed using the batchman language in the SNNS neural network simulation package. The genetic algorithm aspect was performed using my personal modifications to the GENITOR system developed by Darrell Whitley at Colorado State University. During each generation of the GA, all networks were selected a single time for mating. During mating, two NN were combined by selecting two crossover points and switching the part of their corresponding genomes that lied between them.

With a random probability of 4% a single gene on a network's genome was altered, getting assigned a new random value between 0 and 1. All networks were trained with the batch backpropagation through time algorithm, with a learning parameter of 0.2. All nodes used a squashing logistic function defined as $f(x) = 1/(1+e^{-x})$, where x is the total weighted input to the node.

After evolving NN configurations, and in order to verify consistent performance, networks were validated by performing bootstrap validation [6]. During this process, a set of 2000 input patterns was randomly generated, of which 400 were used for training. Trained networks were then tested with the other 1600 patterns. This process was repeated 42 times. Numbers reported as network performances are the average for all cross validation runs.

V. RESULTS

Table 1 outlines the performance of the top topology evolved by the GENDALC system, and compares it against the performance of other commonly used topologies. The values reported are average errors for each of the patterns presented during validation.

TABLE 1: COMPARISON OF AVERAGE OUTPUT ERROR FOR DIFFERENT TOPOLOGIES

Topology	Format a	Format b	Format c
Evolved by GENDALC	0.499	0.499	0.355
Elman (SRN)	0.37	0.040	0.003
Jordan [8]	0.355	0.044	0.004
Feed forward, single layer	0.38	0.004	0.003
Fully connected	0.574	0.640	0.538

These values show the GENDALC system to be highly inefficient in finding solutions to the intended problem. This is particularly true in light of the evidence that other methods find better solutions. These results are specially disappointing given that the GENDALC system achieved error rates of less that 6% in natural language tasks that appeared to be harder than the one presented here [1].

A closer look at the data utilized in the experiments used here reveals how other topologies might be achieving their performance values. Since the child development experiments here imitated always had an initial scene with a ball being placed higher that a barrier, a NN simply has to determine if the ball ends below the barrier in order to determine if the sequence is surprising or not. That is, there is no need for the NN to pay attention to the relative position of the ball and the barrier in the initial scene, since the ball is always higher than the barrier in the initial scenes. If and only if the ball ends lower than the barrier, the sequence is surprising. This method of solving the problem presented is confirmed when the topologies are analyzed while processing input patterns. For example, weights going to the context layer in Elman networks have been set to very low values by the training algorithm. This basically turns these networks into feed forward networks.

In order to test all topologies for their ability to remember, a new set of data was designed. In it, the initial position of the ball could be above the barrier, or below it. In the cases where the ball started above the barrier, the final position could be either above or below. If it was below, the output of the NN should be of 1, indicating surprise. Otherwise it should be 0. In those cases where the initial position of the ball was below the barrier, the final position of the ball was also below the barrier. For these cases, the output of the NN should be 0, indicating no surprise. This mix of data requires a NN to pay attention to both the initial and final ball position before determining if the sequence was surprising or not.

An additional change to the data was that networks only received a single scene at a time. Under what is labeled on table 2 as input format A', the position of the ball and barriers are represented once again with simple coordinates. Networks receive two inputs representing the coordinates of the ball and two inputs representing the coordinates of the barrier. At any particular instant, though, either the initial or the final scene is being presented, but not both. In format C', scenes are represented as in format C before, but with the characteristics for the ball positions described in the previous paragraph. Results for experiments that used this data are presented in table 2.

TABLE 2: AVERAGE OUTPUT ERRORS FOR NEW INPUT SETS.

Topology	Format A'	Format C'
Evolved by GENDALC	0.259	0.279
Elman (SRN)	0.322	0.809
Jordan	0.473	0.468
feed forward, single layer	0.75	0.812
fully connected	0.47	0.502

As we can see, the performance of the GENDALC system is better than those of the other topologies tried. In fact, it is better than its own performance for data where remembering was not important.

VI. COMMENTS ON THE GENDALC SYSTEM

It is by now well known that no search algorithm is optimal for all problem spaces [7]. It is therefore important to identify what type of problem GENDALC might be best suited for.

The topologies evolved by the GENDALC system have proven to be successful at tasks that require remembering input patterns they have seen in the past. This is true not only for the task presented here, but also for natural language processing task it has been used for before [1]. In that task NN received a sentence one word at a time and had to incrementally build a description of the sentence in its output nodes. For example, if the sentence "the boy ran in the park" was entered, the network needed to respond by indicating that "the boy" was a noun phrase, and it acted as the agent of verb "ran". The network also needed to indicate that "in the park" was a prepositional phrase modifying the verb "ran".

The language the NN dealt with is illustrated in figure 4. This grammar, which includes relative clauses, was used to generate sentences with up to three noun phrases and two verb phrases.

S→NP VP　　S→NP VP that S
NP→N　　NP→NP PP
NP→DET N　　NP→DET ADJ N
NP→NP RC　　PP→P NP　　RC→that VP
VP→V　　VP→VP NP
VP→VP PP　　VP→AUX VP
VP→is ADJ　　VP→was ADJ
VP→VP ADV　　P→in|after|with
V→ran|runs|swam|swimming|raced|is|was
N→boy|girl|john|mary|horse|duck|river|park|
car|boat
ADJ→blue|red|fast　　ADV→fast　　DET→the

Figure 4: Grammar used for previous NLP experiments.

One interesting finding of this previous research was how much better the topology evolved by GENDALC was in comparison with other commonly used topologies. Table 3 presents a comparison of the performance of the best topology evolved by GENDALC versus four other (preset) topologies. While there are preset topologies that perform only 7 percentage points below those evolved by GENDALC in the child development task presented here, in the apparently harder NLP task the closest preset topology was 23 percentage points worse than the topology evolved by GENDALC in the NLP task. In addition, in absolute terms, preset topologies have a performance around 65-72% of the maximum on both tasks, while GENDALC's performance jumps from around 75% in the child development tasks to 96% in the NLP task.

Looking at these two tasks, both relative to each other and when changes were made to the child development task, seems to indicate that the performance of the GENDALC system increases directly proportionally with the need to remember the data presented to it.

TABLE 3: NETWORKS PERFORMANCE IN NLP TASK.

Topology	Ave. correct outputs
Evolved by GENDALC	95.99
SRN	70.58
Fully connected	72.95
FGS [3]	71.85
N-P [4]	72.95

This might be caused by the type of topologies it can easily evolve. The algorithm presented in section II above lends itself to evolving topologies with many connections between hidden layers. These layers are then available for storing information being presented sequentially at the input layer. Notice, for example, the topology evolved for the NLP task shown in figure 5. The three different paths leading to layer 2 allowed it to look at a sequence of three consecutive words from the sentence being presented. Given the grammar used, this was enough to determine the syntax of the sentences being presented.

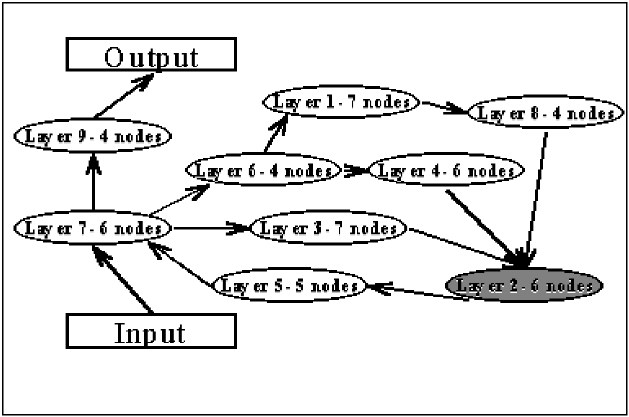


Figure 5: Evolved language processing topology .

VII. FUTURE RESEARCH

I am currently analyzing the type of NN evolved by GENDALC for the child development task discussed in this paper. Looking at that topology and how it performs its task can confirm how important it was to have the type of feedback loop seen in the NLP experiments.

In addition, new experiments are already underway to test GENDALC's ability to evolve topologies for other tasks that require remembering. One of these requires a NN to read a musical composition one note at a time and then determine if it is written using a major or a minor key. Since there is no minimum number of notes across compositions required to make this distinction, only NN that can remember well should be successful.

On the other end of the spectrum, GENDALC will be used to evolve networks for child development tasks that do not require remembering. The task will be modeled after research performed by Goren, Sarty, and Wu [9]. In it, the authors found that newborn babies as young as one hour old visually follow images of human faces longer than other images. To imitate this process, NN will be presented drawings with human faces and with other types of images. Their ability to correctly categorize them will be measured. Given that a complete drawing will be shown simultaneously at the input layer, this experiment will measure the ability of the GENDALC system in a cognitive domain that does not require memory.

VII. CONCLUSIONS

This paper has presented a method of evolving neural networks that appears to be better suited to solve problems that require remembering the patterns it has previously seen. Networks evolved with this method outperform other commonly used topologies in these tests. This shows promise for neural network simulations of human cognition tasks that typically require the ability to remember.

REFERENCES

[1] J. Davila, "Genetic Optimization of NN Topologies for the Task of Natural Language Processing," *Proceedings of the International Joint Conference on Neural Networks,* Washington, D.C., 1999.

[2] Elman, J. L., Distributed Representations, Simple Recurrent Networks, and Grammatical Structure. *Machine Learning, 7,* 1991, pp. 195-224.

[3] Frasconi, P., Gori, M., and Soda, G., (1992) Local feedback multilayered networks. *Neural Computation, 4(1),* 1992, pp. 120-130.

[4] Narenda, K., Parthasarathy, K. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks, 1(1),* 1994, pp. 4-27.

[5] E. Spelke, K. Breinlinger, J. Macomber, and K. Jacobson, Origins of Knowledge, *Psychological Review, 1992, Vol. 99, No. 4,* pp. 605-632.

[6] S. Weiss, and C. Kulikowski, Computer Systems that Learn. Classification and Prediction Methods, in *Statistics, Neural Nets, Machine Learning, and Expert Systems,* 1991. Morgan Kaufmann. San Mateo, California.

[7] W.G. Macready and D.H. Wolpert, No Free Lunch Theorems for Search, *IEEE Transactions on Evolutionary Computation, vol. 1 , no. 1*, pp. 67-82, 1997.

[8] M. I. Jordan, Attractor Dynamics and Parallelism in a Connectionist Sequential Machine. *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pp. 531-546, New Jersey, 1986.

[9] C.C. Goren, M. Sarty, and P.Y.K. Wu. Visual Following and Pattern Discrimination of Face-like Stimuli by Newborn Infants. *Pediatrics, 56*, pp. 544-549.