Chapter 7: Methodology and system design

62-71

7.1 Introduction

Neural network approaches to WSD have been suggested (Cottrell and Small, 1983; Waltz and Pollack, 1985)[65]. These models consist of networks in which the nodes ("neurons") represent words or concepts, connected by "activatory" links: the words activate the concepts to which they are semantically related, and vice versa. In addition, "lateral" inhibitory links usually interconnect competing senses of a given word. Initially, the nodes corresponding to the words in the sentence to be analyzed are activated. These words activate their neighbours in the next cycle in turn, these neighbours activate their immediate neighbours, and so on. After a number of cycles, the network stabilizes in a state in which one sense for each input word is more activated than the others, using a parallel, analog, relaxation process.

Neural network approaches to WSD seem able to capture most of what cannot be handled by overlap strategies such as Lesk's. However, the networks used in experiments so far are hand-coded and thus necessarily very small (at most, a few dozen words and concepts). Due to a lack of real-size data, it is not clear that the same neural net models will scale up for realistic application. Further, some approaches rely on "context setting" nodes to prime particular word senses in order to force 1the correct interpretation° But as Waltz and Pollack point out, it is possible that such words (e.g., writing in the context of pen) are not explicitly present in the text under analysis, but may be inferred by the reader from the presence of other, related words (e.g., page, book, inkwell, etc.). To solve this problem, words in such networks have been represented by sets of semantic " micro features " (Waltz and Pollack, 1985; Bookman, 1987) which correspond to fundamental semantic distinctions (animate/inanimate, edible/ inedible, threatening/safe, etc.), characteristic duration of events (second, minute, hour, day, etc.), locations (city, country, continent, etc.), and other similar distinctions that humans typically make about situations in the world. To be comprehensive, the authors suggest that these features must number in the thousands. Each concept in the network is linked, via bidirectional activatory or inhibitory links, to only a subset of the complete microfeature set. A given concept theoretically shares several microfeatures with concepts to which it is closely related, and will therefore activate the nodes corresponding to closely related concepts when it is activated :itself. However, such schemes are problematic due to the difficulties of

designing an appropriate set of microfeatures, which in essence consists of designing semantic primitives. This becomes clear when one examinees the sample microfeatures given by Waltz ~md Pollack: they specify micro.f carfares such as CASINO and CANYON, but it is obviously questionable whether such concepts constitute fundamental semantic distinctions. More practically, it is simply difficult to imagine how vectors of several thousands of microfeatures for each one of the lens of thousands of words and hundreds of thousands of senses can be realistically encoded by hand.

7.2 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)[66] 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 denned how many hidden nodes the network would have. In addition, information between these Start-End markers denned 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).

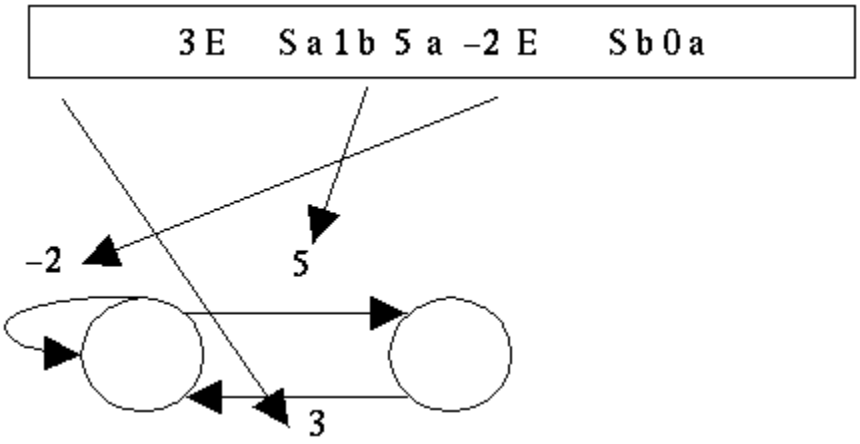


Figure 7-1: example of coding used by Fullmer & Miikulainen (Fullmer and Miikkulainen (1991)).

The network evolved by this process was used to control a virtual creature's movements in a square field, avoiding bad objects and coming into contact with good objects. The GA continued to run until a network that could solve the problem evolved. The number of generations needed until this network was found varied between 7 and 304 for objects that could be identified before hitting them, and between 15 and 414 generations when recognizing the object required travelling around it looking for a characteristic view[67].

Kitano (1994) used GA to evolve a sequence of graph generation rules, as opposed to directly coding network topology. Each genome denned a sequence of rules used to rewrite an element of the graph. When these rules were applied until only terminal symbols remained, the graph denned a connectivity matrix which was then used to configure a NN. For example, if we were developing a network with two nodes, a genome might code rules [S ! AB][A ! 01][B ! 10]. When these three rules are applied we end up with a 2*2 matrix than dense the connectivity between the two nodes in the network.

To be sure, the above examples do not represent a complete list of researchers who have used GA to optimize NN. A more complete review can be found, for example, in Yao (1993)[68].

7.3 A Natural Language Processing Task for a NN

This task was originally presented in D_avila (1999). 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[69]. 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, and at the same time activating those semantic nodes that select the meaning of the word being entered. 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 john from other words that might have the same semantic identity, such as girl . An example of such activation is shown in figure 2.

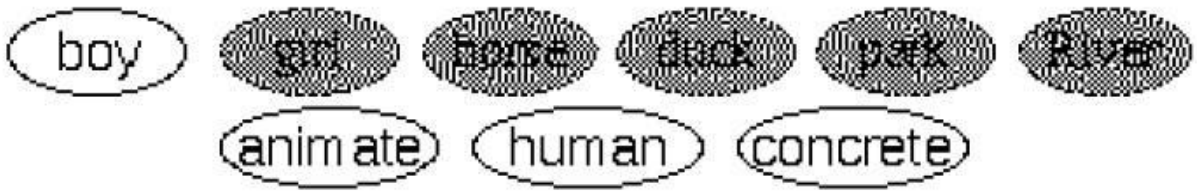


Figure 7-2: Input layer presentation of boy (not all input nodes are shown).

The language used in this research is composed of ten nouns: boy, girl, john, mary, horse, duck, car, boat, park, 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)[70]. In total, there are twelve main verbs

(saw, swam, swimming, ran, runs, running, is, was, raced, floated, said, heard) and two auxiliary verbs (is, was). For example, figure 3 shows how the verb runs is entered in the network; the individual node for runs is activated, as well as semantic nodes representing a verb of movement and a present tense verb. 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

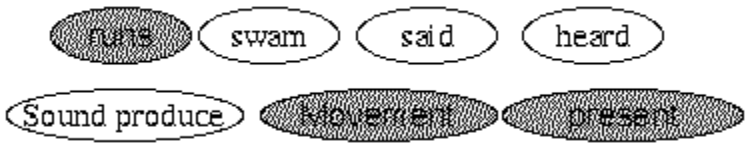


Figure 7-3: Input layer presentation of runs (not all input nodes are shown).

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). An example of such an activation is shown in figure 4.

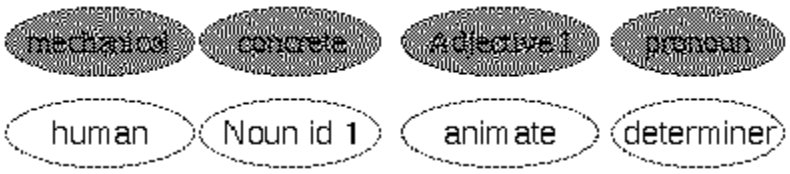


Figure 7-4: output layer presentation of the boy (not all output nodes are shown).

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.

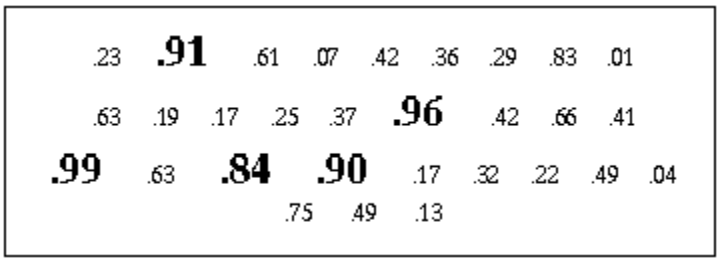
In the manner described above, then, the network should continue to indicate its understanding of the sentence being entered until an end-of-sentence marker is seen. The fitness of a network is determined by computing the sum of squared errors for all output nodes during the processing of all sentences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

7.4 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.

Figure 7-5sample of genes 1-30.



For example, if a genome had genes 1-30 as illustrated in figure 5, 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 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 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.

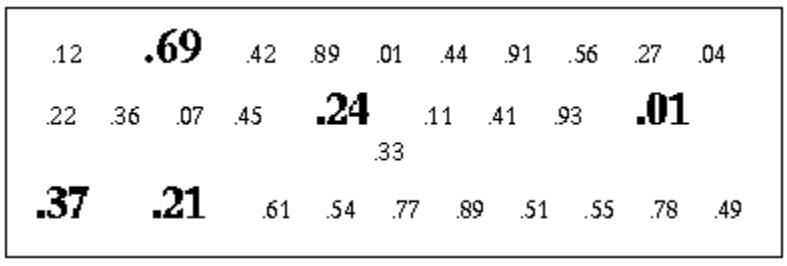


Figure 7-6: sample of genes 31-60.

For example, if the same genome used for the example above had genes 31-60 as illustrated in figure 6, we would look at the corresponding 5 takes-input-from genes, shown in bold in figure 6. 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. Results for the experiment described above are presented in table 1. The four topologies evolved by the GA system described above outperform commonly used NN topologies such as

Topology Average Best Worst

Type I 90.21 92.09 88.97

Type II 95.99 97.62 84.91

Type III 87.59 88.84 86.20

Type IV 82.69 85.97 80.39

SRN 70.58 90.40 17.69

fully connected 72.95 90.41 17.79

FGS 71.85 90.40 33.29

N-P 72.95 90.41 17.79

Table 1: Percent of language correctly processed after training with 20% of complete language, for both evolved and commonly used topologies.

Simple Recurrent Networks (SRN), Frasconi-Gori-Soda networks, and Narendra-Parthasarathy networks. Although some of these commonly used topologies managed to outperform some evolved topologies in the best of cases, on average the evolved topologies performed better by more than 10%. In addition, previously used topologies demonstrate a higher sensitivity to initial conditions.

The worst performance for previously used topologies is more than 45% lower than the worst performance for evolved topologies. Details about the characteristics of the evolved networks can be found in D_avila (1999).

7.5 Schema Disruption Computation

If we view the evolution of NN 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 gi and gj , then the bigger the distance between gi and gj , 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

$$\sum_{k=0}^{C}\sum_{i=0}^{N}\sum_{j=0}^{N}(|i-j| * DC(k,i) * DC(k,j))$$

where N is the number of genes, and DC(k, x) equals to a number between 0 and 1 which indicates what is the probability that gene x is involved in defining connection k. Notice that this number rejects a global probability of disruption for the complete network, as opposed to for any particular connection. This is different from what was originally presented in D_avila (2000)[71], and is motivated by the fact that the more connections a network has, the more likely it is to suffer disruptions.