Speech Synthesis Using Neural Networks Trained by an Evolutionary Algorithm

Prof. Dr. Trandafir Moisa[1], Dan Ontanu [2], and Adrian Horia Dediu [3]

[1] CSE department, Politehnica University, Splaiul Independentei 313, 77206, Bucharest, Romania
tmoisa@cs.pub.ro

[2] CST, Tolstoi 4B, 712893, Bucharest, Romania,
dano@starnets.ro

[3] CST, Tolstoi 4B, 712893, Bucharest, Romania
hd@ziplip.com

Abstract. This paper presents some results of our research regarding the speech processing systems based on Neural Networks (NN). The technology we are developing uses Evolutionary Algorithms for NN supervised training. Our current work is focused on Speech Synthesis and we are using a Genetic Algorithm to train and optimize the structure of a hyper-sphere type Neural Network classifier. These Neural Networks are employed at different stages of the Speech Synthesis process: recognition of syllables and stress in the high level synthesizer, text to phonemes translation, and control parameters generation for the low level synthesizer. Our research is included in a pilot project for the development of a bilingual (English and Romanian) engine for text to speech / automatic speech recognition and other applications like spoken e-mail, help assistant, etc.

Keywords: Evolutionary Algorithms, Neural Networks, Supervised Training, Speech Synthesis

1 Introduction

In order to generate faster and more accurate results in different applications, many researchers use Neural Networks (NN) in conjunction with Evolutionary Algorithms (EA). Thus, evolving weights and thresholds, the network topology and node transfer functions, as well as evolving learning rules or finding a near optimal training set are examples of using EA in the NN training process. A "State of the Art" presentation of Evolving Neural Networks can be found in [Yao 1999].

Our approach is based on developing techniques for simultaneous optimization of both weights and network topology.

It is well-known that the coding scheme used by an EA is crucial for successfully solving a given problem. There have been some debates regarding the cardinal of the

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2074, pp. 419–428, 2001.

alphabet used for the coding scheme. According to Goldberg [Goldberg 1989], the lower the number of symbols in the alphabet, the higher implicit parallelism can be achieved. This way, a genetic algorithm should work with binary genes. However, other studies revealed unexpected advantages of using high cardinality alphabets [Michalewicz 1992]. Genetic operators (average crossover) dedicated to solve specific problems can benefit from such a coding scheme.

Among the EA used for NN training, such as - Genetic Algorithms (GA), Evolutionary Strategies (ES), Evolutionary Programming (EP), etc. – we focused on GA with real number chromosomes and special genetic operators like average crossover [Beasley 1993] and rank-space selection [Winston 1992].

We are using NN trained by GA for solving two practical problems: prediction and text-to-speech (TTS) engine.

As known, a text-to-speech engine is a system aimed to convert a source text into its spoken (i.e. voiced) equivalent [Moisa 1999a], [Moisa 1999b], [Moisa 2000]. The source text is generally supposed to be ASCII encoded, preferably without any spelling or lexical restrictions. Its spoken counterpart is a voice stream, that is, a sequence of voice samples, with a specified sampling frequency and dynamic range. We have developed a complete text-to-speech engine for the English and Romanian languages. The general structure of the TTS engine is outlined in **Fig. 1**:

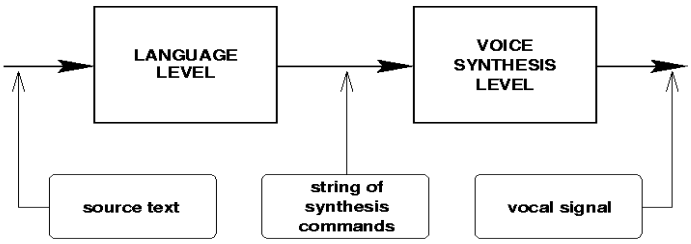

Fig. 1. General structure of the TTS engine

2 Parallel Training of Neural Networks Using Genetic Algorithms

We used the Soft Computing Genetic Tool [Dediu 1996], [Dediu 1999], for the parallel training of the neural networks. Having an Excel type fitness function evaluation interface, we used on a given row the input data set, the neural network architecture, the output computed value, the desired output value, and the error for the current data set. The neural network uses the genes values as weights and thresholds. Using the copy formulas capability, we extended the described structure for the whole data set available. Normally, when referring a data element, we used a relative addressing mode, and when we referred to a gene value, we used an absolute address mode. The fitness function computes the average error for the training data set available. The algorithm works and minimizes the fitness function, thus the global error for the whole data set.

From formal point of view, we modified the well-known "Back Propagation" training algorithm [Lippman 1987]:

Repeat
  select one input data set;
  compute NN output;
  adjust weights and thresholds;
until stop criteria achieved;

And we obtained the following algorithm:

Repeat
  parallel compute NN outputs for whole input data set;
  adjust weights and thresholds;
until stop criteria achieved;

The stop criteria might be err < □ or we can stop after a certain number of training iterations or after a given amount of time.

The following picture (**Fig. 2**) might create an idea of what is happening during the training phase of the neural network using Soft Computing Genetic Tool.

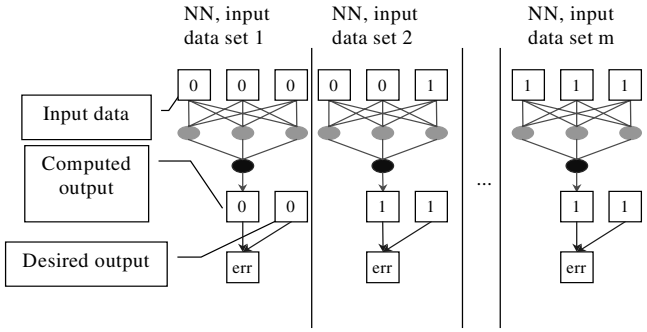

Fig. 2. Parallel training of neural networks using genetic algorithms

Behind the interface of the Soft Computing Genetic Tool, there is a genetic engine that proposes values for genes, than the interface evaluates the fitness function and sends back to the genetic engine the fitness function value.

We should remark the higher degree of parallelism achieved by our algorithm. Thus, the GA contains a parallel population of individuals, every individual having a parallel set of neural network structures, every such structure being, as known, a massively parallel computing system.

3 Neural Networks Trained by Genetic Algorithms – First Results

We tested our method on several data sets and we studied the behavior of neural networks depending on different transfer functions. We realized that the genetic algorithm needs some hints coming from the neural processing elements transfer functions, in order to be able to find out weights and thresholds. Thus, a step transfer

function is practically useless since, after a slightly modification of a gene (weight or threshold) only the luckiest mutations can reduce the global error.

We used for tests the "Game of life" problem where a cell surrounded by eight neighbors survives if it has between 2 and 3 living neighbors. SEQARABISCHOur neural network structure has 9 inputs, one output and 19 hidden neurons fully interconnected. We run the tests and we compare the results of our GA method with a BP training algorithm. We used as genetic operators shuffle crossover, average mutation [Michalewicz 1992] and rank-space selection [Winston 1992]. The following tables summarize algorithms' parameters and comparative RMS errors for GA and BP methods (**Table 1** and **Table 2**). Analyzing the results, we can conclude that the GA used for NN training can find fast a satisfying solution, not so well tuned but well suited for generalization.

Table 1. Algorithms' training parameters

BP parameters		GA parameters	
Max Iterations:	10000	Max Generations:	1000
Learn Rate (Eta):	0.01	Population size:	10
Learn Rate Minimum:	0.001	Crossover Probability	80%
Learn Rate Maximum:	0.3	Mutation Probability	0.48%
Momentum (Alpha):	0.8	Nr. of genes:	210

Table 2. Comparative results BP versus GA

		BP RMS ERROR		GA RMS ERROR	
	Patterns	2330 iterations	10000 iterations	100 generations	1000 generations
Training data set:	452	0.022	0.019	0.2214	0.147
Test data set	60	0.028	0.026	0.016	0.0012

4 Neural Networks Trained by Genetic Algorithms – a Text-to-Speech Engine

The language level comprises several natural language processing stages ([Moisa 1999a], [Ontanu 1998]) such as: brief lexical analysis, exception treatment, conversion to phonological notation (at both rough and fine scale), detection of prosodic features responsible for rhythm and intonation, as well as the generation of the sequence of commands for driving a voice synthesis machine. This processing chain eventually translates the source text in a sequence of lexical tokens, each of these being, in turn, a sequence of phonemes, labeled with prosodic markers. On this basis is finally generated the input for the next stage, namely the sequence of commands. At the voice synthesis level we find a concrete voice generation system, based either on a model for the human vocal tract or, on concatenation of small voice

units, such as diphones, half-syllables and the like, extracted by processing human voice recordings. Our TTS engine employs the latter approach, one based on a TD-PSOLA recent development ([Dutoit 1996]). Such a synthesis machine has the advantage of providing a simple command which, essentially specifies the fundamental frequency and duration for every phoneme that is to be played.

4.1 The Neural Model for the TTS Engine

The neural structure presented in **Fig. 3** is a 2-layer hypersphere classifier, as described in [Ontanu 1993] or [Ontanu 1998]. Each node on the hidden layer (H_i) is related to a n-dimensionally hypersphere in the problem space, centered in C_i and having the radius r_i. If an input pattern belongs to such a domain (say, to the one of center C_k and radius r_k), the associated node (H_k) will fire.

The decision region for a binary (2-class) problem is the union (set theoretic) of all the domains represented by the nodes H_i. This is achieved by putting the final neuron (Y) to perform a "logical OR" among all H_i outputs. The training algorithm is basically a covering procedure in the problem space, trying to approximate the distribution of "positive answer" patterns. As opposed to [Reilly 1982], such a procedure, described in section 2 of the paper, effectively builds the structure by growing the hidden layer accordingly to problem complexity. When compared to multi-layer perceptrons, trained by backpropagation, the overall advantages of such networks are: the speed of the training process, and better generalization capabilities

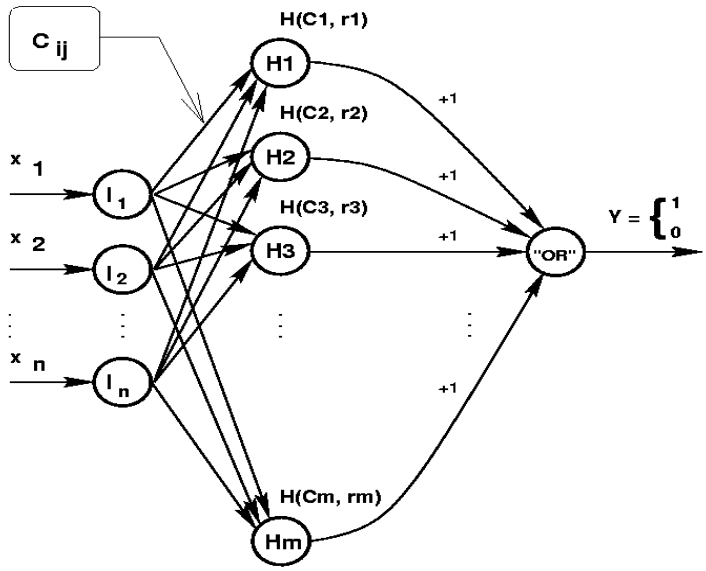

Fig. 3. The structure of a 2-layer binary hypersphere classifier: Hi - active nodes associated to hyperspheres of center Ci and radius ri in the problem space, Ii – passive input nodes

4.2 Lexical Analysis and Exception Processing

This process first looks for the segmentation of the source text in two roughly defined categories: words and punctuation marks. By "word" we mean in this context any sequence of alphanumeric characters and/or usual punctuation marks, separated by white spaces (i.e. tabs, carriage return, line feed, or the space character itself). Next, every scanned word is classified as "expandable" and "not expandable". The first class comprises various numeric expressions, i.e. numbers (*23, 3.14*, etc), date and time marking conventions (*07-26-1999, 13:23*, etc), acronyms like *IBM* or *HTML*, short-cuts for usual words like *ASAP* ("as soon as possible"), *etc* ("etcaetera"), measurement units and so on. The class also includes geographical terms (cities, countries, rivers, mountains), names for well-known companies and institutions (*FMI, Microsoft, Xerox*), foreign currencies (*Euro, USD, DM*), etc. Every such word is replaced by an appropriate expansion, which is a conversion to its spoken form. For instance, *23* becomes, after expansion, *twenty-three*. We say that some of the words are directly "expandable", meaning that the spoken form is found by using some appropriate procedure; these are the various numerical expressions we've talked above. On the contrary, for any other terms, we consider using an exception dictionary, in fact, an associative and continuously extendable database, containing the proper expansion for every entry it has. For some word categories, the dictionary gathers not only the corresponding spoken string, but also items like the plural and singular forms or the word's gender. This helps solving situations like *30 Km/h*, which obviously require the plural form for *km/h* ("kilometers per hour" and not "kilometer per hour"). The same dictionary includes some typical word contexts for solving most verb/noun formal ambiguities; in other words, homonymic (or homographical) pairs, like *lives* in "he lives the house" vs. *lives* in "they fear for their lives". The second class, the "not expandable", includes all the other "normal" words (that is, anything not belonging to the first class). Such words are checked against a pronounciability test, in fact, a heuristic meant to detect a proper blend of vowels and consonants within the word's structure. We allow for a maximum of 3 vowel successions as well as for a maximum of 4 consonants in a row. If the test fails, we choose to spell the word. Finally, we take into account some punctuation marks, namely: *{.}, {,}, {:}, {;}, {?}*. We call these as having prosodic value, meaning the fact that, at the prosodic analysis stage, these marks will indicate several types of intonation. As we'll explain latter, we only consider local intonation which consists in the alteration of the voice fundamental frequency only in the neighborhood of such a punctuation mark, accordingly to three "laws": a descending contour (for statement-type clauses), an ascending contour (for interrogation-type clauses), and a "neutral" contour, characteristic for compound statements. The whole lexical analysis process is performed by a bottom-up parsing procedure whose "semantic actions" consist in creating new lexical tokens, on the basis of the above discussed expansion operations. The output of this stage is a string of tokens, made exclusively of letters and/or prosodically valued punctuation marks.

4.3 Translation to Phonemic Notation

As seen from figure **Fig. 4**, this process involves two distinct stages. The first one (the primary translation), converts each token produced by the lexical analyzer to a string of phonemes. This is accomplished by means of using a neural network (set of hypersphere classifiers as illustrated in **Fig. 5,** trained over a 120,000 items pronunciation dictionary. As a result, a letter is usually replaced by a sound (phoneme) or a pair of sounds. Stressed vowels are also marked accordingly. The notation employs one character per phoneme and is inspired by a convention proposed at the LDC (Linguistic Data Consortium). The second stage takes into account some allophonic variations, like the aspired counterparts for [t], [p], [k] (cf. *pit*, *cut*, *task*) or the [o] variant in words like *sort*. The output of either two levels is a string of converted tokens. The main difference is that the "fine" process outputs also the inherited prosodic markers (i.e., for rhythm and intonation).

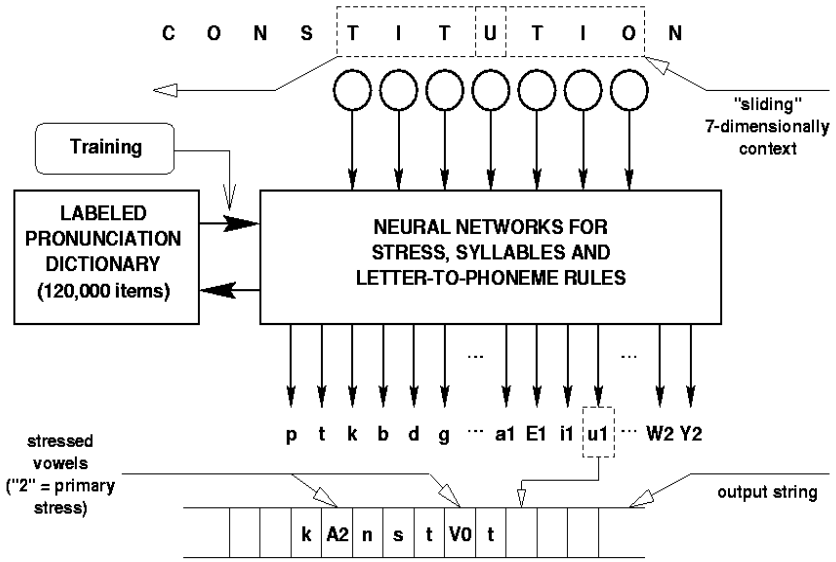

Fig. 4. Translation to phonetic notation and detection of prosodic features (such as stress) using neural networks (hypersphere classifiers)

4.4 Driving the Voice Synthesizer. Neural Networks for Generating Duration and Pitch

In this section we briefly present the main aspects of a neural principle for generating the command in TD-PSOLA like voice synthesizers. As known, these voice

generators are driven at phoneme level, by means of specifying the duration and the pitch (fundamental frequency) curve. Specifying fundamental frequency amounts to giving a discrete approximation of it, that is several "location-value" pairs spread over the phoneme duration. The TD-PSOLA algorithm uses then interpolation in order to achieve a continuous pitch variation. This forms the basis for the intonation control. Given the above hypotheses, we consider labeling vowel-centered phonemic contexts with duration and pitch tags. These contexts are obtained by sliding a 7-characters window over the string of phoneme tokens (interspersed with stress markers), representing the output of the former text processing stages. We gather only those windows centered on a vowel. The net eventually recognizes the attributes for the central vowel of the current context, namely its duration (in *ms*) and the fundamental frequency value, say, in the middle of its length (50%), expressed in Hertz. **Fig. 5** reveals a set of binary neural classifiers, trained and built using the algorithm presented in this paper. The duration and pitch variation ranges are conveniently sampled into a set of discrete values. Thus, we allow for vowel duration starting at 30 ms (short, unstressed vowels) and ending at 200 ms (for example, a long stressed final vowel). As for the pitch, the range covers the typical domain for male voice, i.e. 80-400 Hz. Both cases use a variation step of one unit (1 ms, respectively 1 Hz).

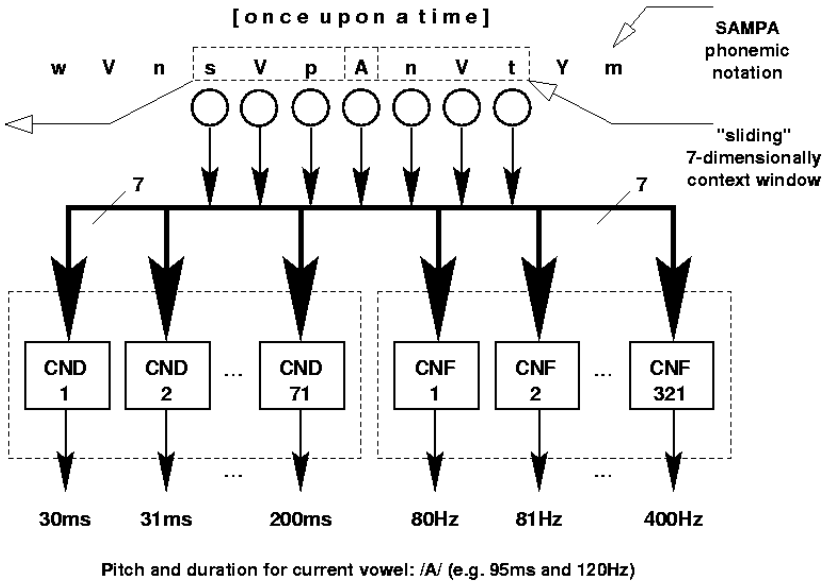

Fig. 5. Set of neural classifiers for recognizing duration and pitch. (CND = Duration Neural Classifier, CNF = Frequency Neural Classifier)

For every discrete value, within the specified ranges, we associate a 2-layer hypersphere neural classifier aimed at recognizing contexts having that value as its attribute. We come, therefore, to a set of 71 binary networks for duration and 321 for frequencies, as pointed out in the figure below.

It becomes obvious that the training process for the above nets requires the evaluation (measurement) of both parameters for a large set of vowel contexts. We recorded and used a dictionary of about 4000 English basic words. From the corresponding voice recordings, we further extracted by hand more than 8000 phonemic contexts and measured for each one the duration and the pitch values at three different knots (15%, 50%, 85%). Training sets were created for each of these measurement points. Then, the learning process built the appropriate net structures, as presented above. The low-levels synthesis engine is based on diphone concatenation. A diphone can be viewed as the transition region between two successive phonemes. A large base of such elements is used for the synthesis process. For the English language, it roughly includes about 1600 diphones and amounts to 7 MB of storage. The output of the **CND**s and **CNF**s (**Fig. 5**) is translated into commands with the following syntax:

<phoneme_name> <duration> {<position> <pitch value>}.

Curly brackets indicate that the synthesizer accepts several pairs "duration-pitch", as mentioned earlier, but we confine ourselves to a simpler command structure, comprising a single pitch specification. The synthesizer generates vocal signal at 16,000 samples per second, with 16 bits per sample. These values express, in fact, the recording properties of the diphones database.

5 Concluding Remarks

Our experience in using NN models (particularly hyperspehere classifiers) to develop TTS and ASR engines, revealed the strength of the method and the benefits of this approach.

To further improve the results, we investigated different NN training methods using EA. In this respect, we have developed a GA employing high cardinality alphabets for the coding scheme and a rank space selection operator. This way, a higher parallelism has been achieved, speeding up the training process.

The results of this approach have been emphasized by an increased efficiency of our bilingual (English and Romanian) engine for text to speech / automatic speech recognition, as well as related applications such as: spoken e-mail, help assistant, etc.

References

[Beasley 1993] David Beasley, David R. Bull, Ralph R. Martin, *An Overview of Genetic Algorithms, Part 2, Research Topics,* Univerity Computing, 1993, 15(4) 170-181

[Box 1970] G.E. Box and G.M. Jenkins, *Time Series Analysis: Forecasting and Control*, Holden Day, 1970.

[Dediu 1996] A.H. Dediu, D. Mihaila, *Soft Computing Genetic Tool, Proceedings of First European Congress on Intelligent Techniques and oft Computing EUFIT'96,* Aachen 2-5 September 1996, Vol. 2,pp. 415-418.

[Dediu 1999] Dediu Adrian Horia, Agapie Alexandru, Varachiu Nicolae, *Soft Computing Genetic Tool V3.0 – Applications*, 6-th Fuzzy Days in Dortmund-Conference on Computational Intelligence; 1999

[Dutoit 1996] T. Dutoit, V. Pagel, N. Pierret, F. Bataille, O. Van der Vrecken, The MBROLA project: *Towards a Set of High Quality Speech Synthesizers* Free of Use for Non Commercial Purposes Proc. ICSLP 96, Fourth International Conference on Spoken Language Processing, vol.3, p. 1393-1396, Philadelphia, 1996

[Goldberg 1989] D. E. Goldberg. *Genetic Algorithms in search, optimization and machine learning*. Addison-Wesley, 1989

[Gori 1998] M. Gori and F. Scarselli, *Are Multilayer Perceptrons Adequate for Pattern Recognition and Verification*?, IEEE Transactions on Patern Analysis and Machine Intelligence, November 1998, Volume 20, Number 11, pp. 1121-1132.

[Lippman 1987] Richard P. Lippman, *An Introduction to Computing with Neural Nets*, IEEE ASSP MAGAZINE APRIL 1987.

[Michalewicz 1992] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 1992

[Moisa 1999a] Trandafir Moisa, Dan Ontanu, *Learning Romanian Language Using Speech Synthesis*, Advanced Research in Computers and Communications in Education, New Human Abilities for the Networked Society, Edited by Geoff Cumming, Tashio Okamoto, Louis Gomez, Proceedings of ICCE'99, 7th International Conference on Computers in Education, Chiba, Japan, pp. 808-815, IOS Press, Volume 55, 1999

[Moisa 1999b] Trandafir Moisa, Dan Ontanu, *A Spoken E-mail System,* 12th International Conference on Control Systems and Computer Science – CSCS12, May 26 – 29, 1999, Bucharest, Romania.

[Moisa 2000] Trandafir MOISA, Adrian Horia DEDIU, Dan ONTANU, *Conversational System Based on Evolutionary Agents*, CEC 2000, International Conference on Evolutionary Computation USA 2000

[Ontanu 1993] Onţanu, D.-M. *"Learning by Evolution. A New Class of General Classifier Neural Networks and Their Training Algorithm"*, Advances in Modelling & Analysis, AMSE Press, Vol. 26, nr. 2/1993, pp. 27-30

[Ontanu 1998] Dan Ontanu, Voice Synthesis, referate for the PhD. thesis *"Neural Models for Specch Recognition and Synthesis"*, under the direction of prof. dr. Trandafir Moisa, author, "Politehnica" University, Bucharest, July 1998

[Reilly 1982] Reilly, Cooper, Elbaum, *"A Neural Model for Category Learning"*, Biological Cybernetics, 45, p. 35-41, 1982

[Winston 1992] Patrick Henry Winston, *"Artificial Intelligence"* third ed., Addison-Wesley, June 1992

[Weigend 1994] A.S. Weigend, N.A. Gershenfeld (Eds.), *Time Series Prediction: Forecasting the Future and Understanding the Past*, Addison Wesley, 1994.

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