A Language-Independent Neural Network-Based Speech Synthesizer

Mario Malcangi, David Frontini
Università degli Studi di Milano
DICo - Dipartimento di Informatica e Comunicazione
Via Comelico 39 – 20135 Milano - Italy
Laboratorio DSP&RTS (Digital Signal Processing & Real-Time Systems)
Laboratorio LIM (Laboratorio di Informatica Musicale)
malcangi@dico.unimi.it, d.frontini@acm.org

Abstract

The applicability of soft computing to implementing text-to-speech conversion is subject to debate. Using neural networks for phoneme-level, text-to-speech conversion has several advantages over hard computing. Soft computing's capacity to generalize makes it possible to map words missing from the database, as well as to reduce contradictions related to different pronunciations for the same word.

Neural networks have been shown to optimally solve a large class of applied pattern-matching problems, but very little research has been done to match the requirements of pattern generation in machine-to-human interaction.

An artificial speech synthesizer based on neural networks is being developed for application to deeply embedded systems for language-independent speech commands on hands-free interfaces. A feed-forward, backpropagation artificial neural network has been trained for this purpose using a custom-developed, regular expression-based, text-to-phones transcription engine to generate training patterns. Initial experimental results show the expected properties of language independence and in-system learning.

1. Introduction

Speech synthesis, the automatic generation of speech waveforms, has developed rapidly in last decade. Very good results have now been achieved in text-to-speech (TTS) synthesis technology, above all in terms of speech quality. These results have been attained with redundancy as the primary solution to the many hurdles to producing quality speech, thanks to the assumption that vast resources (memory, computing power, etc.) are available now (and in the future). Personal computers now meet these requirements, but speech-synthesis applications are mainly targeted at embedded and deeply embedded systems. Memory and processing-power resources are scarce in this type of system and the current approach to TTS synthesis does not suit them.

Speech synthesis becomes a very complex task if the main goal is to implement it on a deeply embedded system with real-time, unlimited-vocabulary, speaker-independent and language-independent specifications. Current speech-synthesis solutions cannot be scaled down to satisfy the emerging demands of embedded systems in every application field where human-to-machine interaction is required.

In the past, when desktop computing had meager computing power and memory storage, many research and development efforts sought optimal solutions to speech-synthesis problems, such as text-to-phoneme translators based on neural networks and phoneme-based speech-synthesizer circuits. When DSP (digital signal processor) chips were introduced in 1980, hardware implementation of speech synthesis was neglected and firmware-based speech synthesizing became a more interesting topic for researchers and developers. Because memory was a scarce resource, great effort was devoted to compressing speech data. In the last decade, the availability of memory in desktop and laptop computers has driven speech-synthesis research to aim for high-quality speech production, with engineers and system designers paying little attention to system optimization.

The next wave of computing and communication technology exhibits a clear trend towards embedded

computing solutions based on system-on-chip (SoC), system-on-package (SoP), etc. These emerging computing technologies are very powerful, but not redundant in terms of memory and computing power. Speech synthesis needs a systemic approach to achieve new results befitting this new scenario.

The artificial neural-network (ANN) [1][2] approach to speech synthesis [3] can optimally solve several implementation and application problems, above all because it is closer to the process to be emulated, the human ability to communicate by means of voice and language.

Sejnowski and Rosenberg [4] were the first researchers to demonstrate that ANN can be successfully applied to the speech-synthesis challenge. Since that demonstration, many new ANN approaches to speech synthesis have been proposed [5][6][7].

The main advantage of applying an ANN in speech synthesis is its ability to learn to speak just as a human does. This means that it can learn to speak any language, just as a human does. The ANN engine is the same for any language it is trained in, so there is no coding dependency because the ANN trained for a specific language is only data dependent.

The traditional approach to TTS requires a large amount of data to represent all the knowledge about how a word has to be converted into a correct speech-synthesizer control stream. The ability of an ANN to generalize reduces this amount of data and also performs a smoothing action on the output, resulting in more natural speech synthesis [8][9].

In the following sections, this paper presents the system framework, ANN architecture, training strategy, and performance evaluation. A brief concluding section summarizes the work and future plans.

2. System framework

The main idea in developing our system framework is to set up an almost fully automatic process to generate a ready-to-run, ANN-based speech synthesizer trained for a specific language, starting from ASCII text.

A special-purpose development environment (Figure 1) was designed for this purpose. It consists of four functional blocks, each executing a whole task useful in developing, training, evaluating, and implementing a complete text-to-speech application.

The functional blocks are:

- Regular expression-based, text-to-phones translator
- Training-set builder
- ANN engine
- Speech synthesizer

The regular expression-based, text-to-phones translator is a specially developed engine that can process ASCII text and convert it to the corresponding phonetic transcription. It is based on a language-specific rule set, which embeds all the phonetic information needed to correctly speak each word in the text.

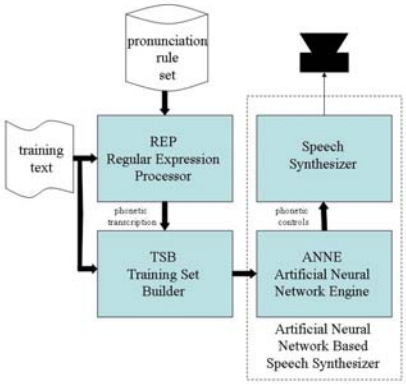

Figure 1. System framework.

The text and its phonetic transcription are the data input for the training-set builder. This functional block automatically processes a phonetic transcription of the text used to train the ANN, generating the training patterns to tailor the ANN for this specific application.

A critical task executed by the training-set builder for this ANN is aligning the text to its phonetic transcription. Manual alignment is avoided by using rule-based alignment between text and phonetic transcription.

The ANN engine is the core processor of a neural network-based speech synthesizer. It is based on JOONE (Java Object-Oriented Neural-Network Engine) [10]. This engine was trained using the patterns generated from the training-set builder. When training is complete, the ANN engine is ready to process a text string and generate a pattern to run the speech synthesizer.

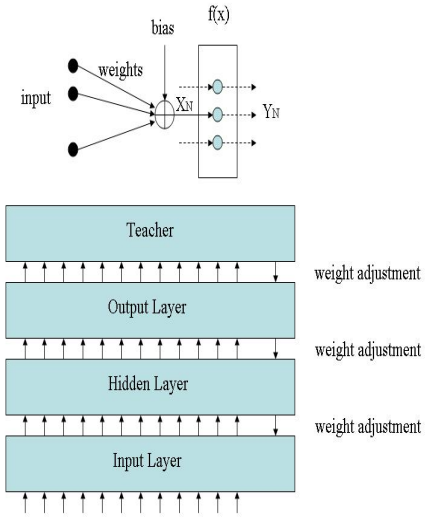

Figure 2. JOONE.

The speech synthesizer, not discussed in this paper, is a phoneme sequencer that allows software control of rate, pitch, amplitude, duration, movement rate, and articulation rate of spoken phonemes.

This framework allows the ANN to be trained automatically by the user (application engineer or end user) and applied to a specific task. The only data needed to build the ANN to drive the speech synthesizer for a specific language is a large text and the phonetic rule set for that language. Both these data sets are currently available for any language. Therefore, the neural network-based speech synthesizer is effectively language independent.

This architecture allows the ANN-based speech synthesizer to be updated to a different language simply by changing interconnection weightings, while leaving its programming code as is. This enables run-time, in-system updating of the speech-response application to the country where it is running.

3. Regular expression-based text-to-phones translator

Much of the computation in text-to-speech synthesis consists of a series of processing passes applied to the text to be uttered. Each pass serves to transform the input text string into an output control parameter string. To execute these tasks, special-purpose (language dependent) algorithms need to be developed.

Most of these algorithms apply context-sensitive rules such as "*if-then*," so our efforts have focused on defining a fully phonetic and linguistic data-driven (rules) text processor. The format of the rules is the following:

$$C(A)D = B \qquad (1)$$

which reads "*A is transformed into B, if the text to which it belong matches A in the sequence CAD*", where C is a pre-context string and D is a post-context string. B is the phoneme string (symbols or codes).

Because many rules share elements of pre-context and/or post-context, classes of elements were defined to make rule coding more compact and to reduce the number of rules.

For example, for Italian and English the following classes of elements, with the corresponding regular expressions, were defined:

(!)	| (^) | ($)	
(#)	| ([AEIOUY]+)	
(:)	| ([^AEIOUY]*)	
(+)	| ([EIY])	(2)
($)	| ([^AEIOUY])	
(.)	| ([BDGJMNRVWZ])	
(^)	| ([NR])	

Using these classes, a general rule (for English) such as the following can be coded (using X-SAMPA to encode phones):

$$!(BI)\# = /b/a/j/ \qquad (3)$$

The rule is thus valid for several text strings, such as BIO…, BION…, BIOG…, BIANNUAL, etc.

The rules are ordered with the most specific first, the most general last:

$$\begin{array}{l} !(B)! = /b/i:/ \\ \quad \ldots \\ !(BI)\# = /b/aj/ \\ \quad \ldots \\ (B) = /b/ \end{array} \qquad (4)$$

The regular expression-based, text-to-phones translator is a general purpose text-processor engine applicable to any written text in any language. It depends only on the rule set and on the classes defined for that language.

The algorithm is as follows, if ρ is the word to be translated, $\mathcal{R}$ the current rule, and ***cover*** is the size of the string to be converted:

```
size = size(ρ)
  for index=0 to size do
    c ← get(ρ, index )
    Rules ← GetRules( c )
    for ℛ in Rules do
      pre-Context← createPreContext(ρ,c)
      post-Context← createPostContext(ρ,c)
      if ℛ is valid c|ρ then
          result ← result + ℛ.phoneme
          cover ← ℛ.cover
          index ← index + cover
          break
      end if
    end for
  end for
```

4. ANN architecture

The ANN has a three-layer, feed-forward, backpropagation architecture (FFBP-ANN), similar to that used in NETtalk [4]. Its inputs are fully connected to all the nodes of the hidden layer, and the hidden layer is fully connected to the output nodes.

All the inputs and outputs have a linear activation function that controls the connection. A non-linear activation function (sigmoid) connects hidden-layer nodes to output-layer nodes.

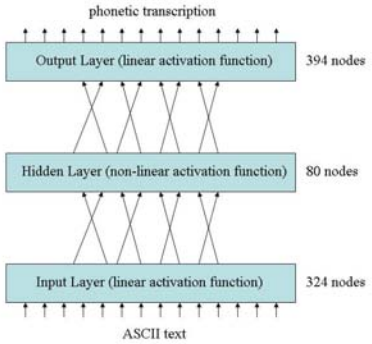

Figure 3. Architecture of the FFPB-ANN.

Input to the FFBP-ANN consists of nine consecutive characters of text to be phonetized. The output encodes the phone that corresponds to the character in the center position of the nine-character input window.

Each of the nine input nodes consists of 36 binary elements, one for each symbol in the character set (the complete alphabetic set A-Z plus a few control characters such as space, period, question mark, etc.). There are thus 324 (36 x 9) total input elements for the FFBP-ANN, enough to fully encode a complete pattern nine characters wide.

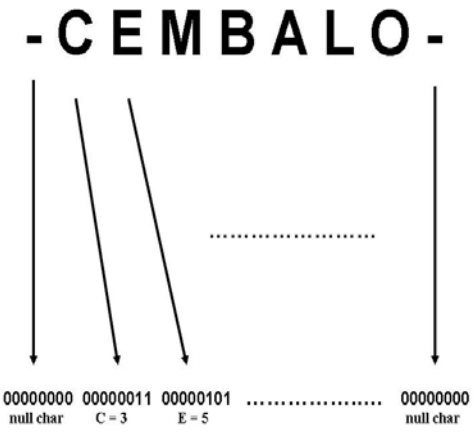

Figure 4. Input-layer data encoding.

The output consists of 394 nodes, one for each phone of the phone set to be used.

Current output encodes the phone that corresponds to the middle character in the input-layer string. The pre-context and post-context of the current input character help determine output.

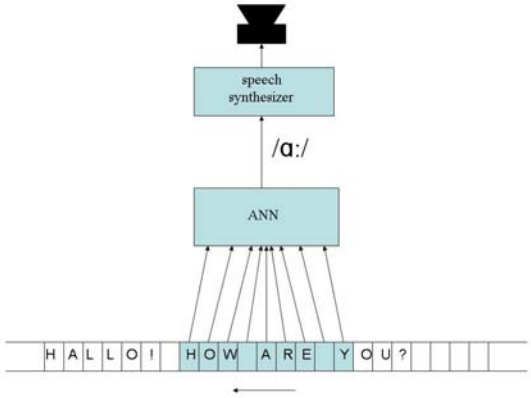

Figure 5. Sliding window.

The size of the layers depends on the number of phones that the ANN needs to represent and on the number of different characters that may occur in the text to be uttered. If the ANN is to be trained for a specific language, the size of the input layer can be fixed as the character set of that language.

The output layer cannot be fixed, because a given language exhibits many variations (phones) of basic speech sounds (phonemes). The ANN must be adapted to the specific speech synthesizer being employed. For this reason, the output layer is resized at training time to fit the applied synthesis model.

To fix output layer size, a binary coded solution can be implemented, but this leads to a lower learning rate compared to one-in-position, as reported in section 6.

5. Training strategy

FFBP-ANN training strategy is a key issue in our framework. One primary goal is to fully automate the training procedure so that the ANN can be trained for any language regardless of its specificity.

The backpropagation algorithm was used as the learning algorithm to train our FFBP-ANN. Successfully applying this algorithm required that training data (words and their pronunciations) be collected and that input and output data be converted into suitable binary form.

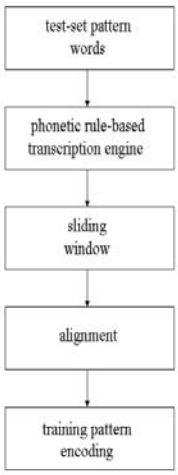

Figure 6. Training-set generation process.

Two main problems have to be solved: generating the set of training patterns and aligning them.

5.1. Training patterns

Rosenberg and Sejnowski developed a special-purpose dictionary of English words and their pronunciations. Each word consists of two strings, the phoneme string (transcribed in PronLex) and the stress string. Both strings are aligned character-by-character with the word as shown for the two words *agglutinate* and *aberration*:

```
a  →  x, 0;
g  →  g, <;
g  →  -, >;
l  →  l, >;
u  →  u, 1;
t  →  t, <;
i  →  -, 0;
n  →  N, <;
a  →  e, 2;
t  →  t, <;
e  →  -, <;

a  →  @, 2;
b  →  b, <;
e  →  x, 0;
r  →  r, <;
r  →  -, >;
a  →  e, 1;
t  →  S, >;
i  →  -, 0;
o  →  x, <;
n  →  n, <;
```

This training set is very large but not exhaustive. It cannot be automatically generated for any language because it requires a phonetic dictionary for the target language. It also needs manual alignment when word length fails to match phone-string length.

In an attempt to achieve fully automated training-pattern generation, the pronunciation rule set and the text-to-phones algorithm are used to generate the training pattern starting from the word list alone.

The pronunciation rule set embeds both alignment information and stress information in each rule, so alignment can be executed automatically during training-pattern generation. The stress information is encoded in the phone symbol as follows:

/a/	not stressed
/'a/	stressed
/'a:/	stressed and long

This solution allows any stress information to be encoded into the phone name, so the ANN can learn more about different pronunciations of each phone.

5.2. Aligning patterns

Pattern alignment [11] is automatically solved by looking up the pronunciation rules applied during the generation of the phonic transcription for each word in the training set.

Defined as the transformation *T* of the word string *(ρ)* into the phonetic string (p) by applying the rule $\mathcal{R}$ (see section 3):

$$p = T(\mathcal{R}(\rho))$$

the alignment problem between the word character string and the corresponding phone character string is implemented during run time when the rule is applied. Implementation is based on the following algorithm:

```
if sizeof(p) = sizeof(ρ)
    no alignment required
 else
    cover ← R.cover
    if cover > 1
          alignment required
    endif
endif
```

If alignment is not required, a one-to-one association between one character of the word ρ and one phone is established. If alignment is required, a many-to-one association between a number equal to *cover* characters of the word ρ and one phone followed by *cover-1* null phones (coded as -) is established.

Applying this method to the Italian word GOGNA, yields the following alignment:

```
G = /_g/
O = /'o/
G = /J:J/
N = /-/     ← null phone inserted
A = /a_/
```

The applied rules are the following:

```
!(G) = /_g/          cover=1
…
(O)$$# = /'o/        cover=1
…
(GN)#! = /J:J/       cover=2
…
(A)! = /a_/          cover=1
…
```

Once alignment is complete, the sliding window algorithm is applied to the transcribed word, generating the following training array (based on a nine-character window):

```
_ _ _ _ _   G O G N A   /_g/
_ _ _ _ G   O G N A _   /'o/
_ _ _ G O   G N A _ _   /J:J/
_ _ G O G   N A _ _ _   /-/
_ G O G N   A _ _ _ _   /a_/
G O G N A   _ _ _ _ _   /-/
O G N A _   _ _ _ _ _   /-/
G N A _ _   _ _ _ _ _   /-/
N A _ _ _   _ _ _ _ _   /-/
A _ _ _ _   _ _ _ _ _   /-/
```

5.3. Training

Training is performed by running the backpropagation algorithm. This algorithm is difficult to set up for application-specific purposes because several parameters must be trimmed for optimal learning, the number of hidden units, initial random weight, learning rate, the coefficient of momentum, and stopping point (how many training epochs to run) [12].

Several experiments were conducted to find the best combination of such setup parameters. The following setup was identified:

Input layer units: 324
Hidden layer units: 80
Output layer units: 394
Learning epochs: 400
Learning rate: 0.15
Momentum: 0.9

As a training set, 720 Italian words are used. Each input word is assigned a set of phone codes, each including all the parameters that make the word sound natural when played through the synthesizer. The network learns the relationship between input words and output parameters through iteration. After 50 passes (epochs) through the training set, the phonetized text will be understandable, and after 100 passes, the error rate will be less than 2%. After 400 training passes the error rate is below 1%. This represents an ANN stable state useful for synthesizing good-quality speech.

To test the ANN thus set up, an Italian text was conveyed into the system and the corresponding

utterance was produced by driving a concatenative speech synthesizer.

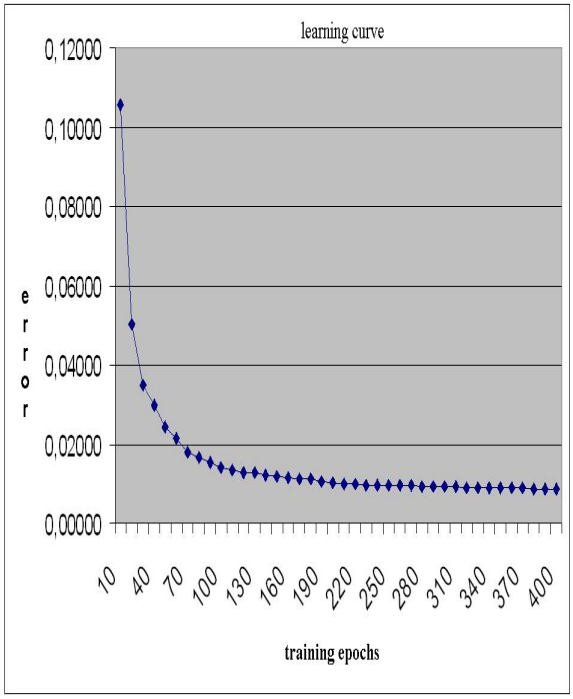

Figure 10. Learning curve after 400 epochs of 720 patterns.

6. Performance evaluation

To evaluate the trained FFBP-ANN's performance, a special-purpose environment that implements the soft-computing and hard-computing text-to-speech models (TXT2SP) was developed. Both implementations share the pronunciation rule set, allowing significant comparison.

Hard-computing (algorithmic) implementation of the text-to-speech uses a set of coded phonetic rules to execute the text-to-phones transcription. Soft-computing (ANN) implementation uses the trained knowledge obtained during learning, from the same set of coded phonetic rules used to run the hard-computing implementation.

The same word (text) can be sent as input to both transcription processes (soft- and hard-computing) and compared at the phonetic level. Utterance-level comparison is also available because the concatenative speech synthesizer is integrated into the TXT2SP developing environment.

The TXT2SP development environment also provides the functions needed to set up and train the ANN, as well as to compare ANN performance with rule-based implementation, using a text file as test input.

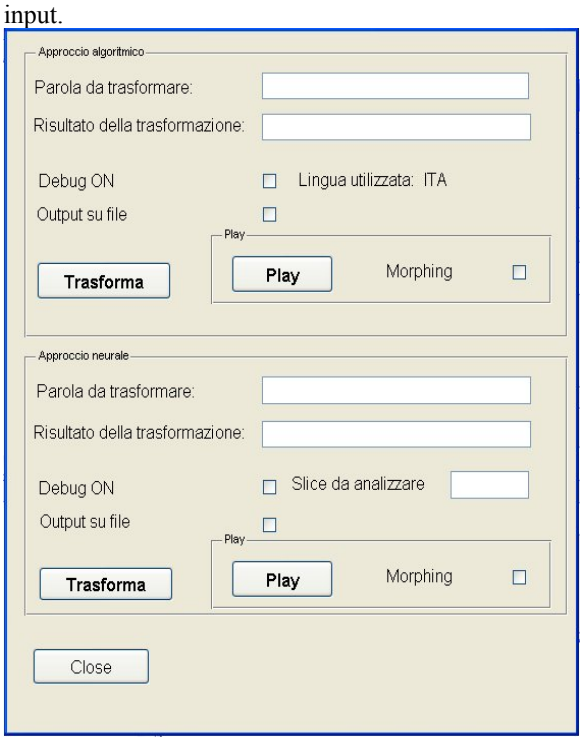

Figure 11. Graphic user interface of the special-purpose developed environment (TXT2SP) used to compare ANN implementation with rule-based implementation of text-to-speech synthesis.

The hard-computing implementation proves faster. But it fails if the word is not included in the pronunciation rule set. This happens when a proparoxytone word is tested.

For example, the proparoxytone word

CEMBALO = /_tS/'e/m:/b/a/l/o_/

is not in the pronunciation rule set, so the hard-computing implementation incorrectly transcribes it as follows (default):

CEMBALO = /_tS/e/m/b/'a:/l/o_/

However, the soft-computing implementation runs almost correctly:

CEMBALO = /_tS/'e/m/b/a/l/o_/

It fails only with regard to the duration of /m/ phone. Moreover, it identifies the correct stress position. This demonstrates the ANN's very high

generalization ability despite the limited number of exception words furnished in the 720 word-based training set. Increasing the number of exception words at training time, a more robust generalization ability can be embedded in the ANN.

Increasing training-set size implies increasing the ANN's learning time. This time can be trivially reduced by using a faster executing processor or a parallel executable software coding of the ANN. A smarter solution can be found in ANN development strategy.

A test was performed to compare the learning speed of an ANN with binary-coded output to that of an ANN with binary-decoded (one-in-position) output. This test showed that the one-in-position ANN learns faster, as shown in fig. 12.

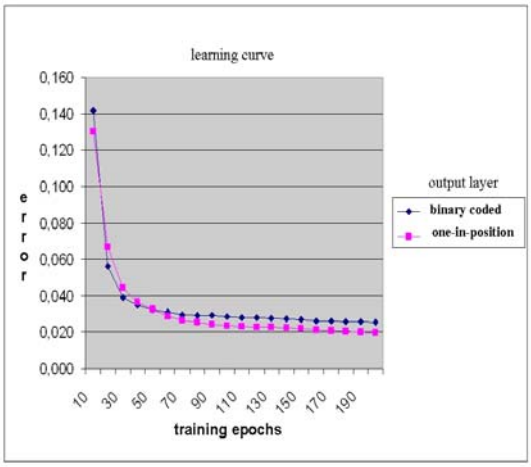

Figure 12. Learning speed is better for one-in-position coding at output layer level.

7. Conclusions

An FFBP-ANN-based engine for text-to-phone transcription is being developed to act as the back end for phonetic synthesizers. It can be trained automatically if pronunciation rules are available. Only a set of words is needed for training.

The ability of the trained FFBP-ANN to generalize shows that this approach to text-to-speech becomes advantageous compared to current TTS implementations when embedded systems are the target.

Future efforts will strive to solve the problem of automatically generating the pronunciation rule set for a given language starting from text only. A further aim will be to develop an ANN-based speech synthesizer that can be embedded in an SoC solution.

8. References

[1] C. Bishop, "Neural Networks for Pattern Recognition". Oxford University Press, 1995.

[2] S. Haykin, "Neural Networks A Comprehensive Foundation". Prentice Hall, 1999.

[3] D. P. Morgan, C. L. Scofield, "Neural Networks and Speech Processing", Kluwer Academic Publishers, 1991.

[4] T. J. Sejnowski, C. R. Rosenberg, "Parallel Networks that Learn to Pronounce English Text". Complex System Publications, 1987.

[5] M. Rahim, C. Goodyear, "Articulatory Synthesis with the Aid of a Neural Net", in Proceedings ICASSP, Glasgow, Scotland, May 1989, pp. 227-230.

[6] M.S. Scordilis, J.N. Gowdy, "Neural Network Based Generation of fundamental frequency contours", in Proceedings of ICASSP, Glasgow, Scotland, May 1989, pp. 219-222.

[7] F. Hendessi, A. Ghayoori, T.A. Gulliver, "A Speech Synthesizer for Persian Text Using a Neural Network with a Smooth Ergodic HMM", ACM Transactions on Asian Language Information Processing, Vol. 4, No. 1, March 2005.

[8] G. Bakiri, T. G. Dietterich, "Achieving High-Accuracy Text-to-Speech with Machine Learning", In R. I. Damper, editor, *Data Mining Techniques in Speech Synthesis*. Chapman and Hall, New York, NY, 2002.

[9] J. A. Bullinaria, "Representation, Learning, Generalization and Damage in Neural Network Models of reading aloud ". Edinburgh University Technical Report, 1994.

[10] JOONE, Java Object Oriented Neural Engine, http://www.joone.org.

[11] C. X. Ling, H. Wang, "Alignment Algorithms for learning to read aloud". IJCAI, 1997.

[12] M. Moreira, E. Fiesler, "Neural Networks with Adaptive Learning Rate and Momentum Terms". IDIAP Technical report, 1995.