SELF-ORGANIZING LETTER CODE-BOOK FOR TEXT-TO-PHONEME NEURAL NETWORK MODEL

Kåre Jean Jensen and Søren Riis

Nokia Mobile Phones, Frederikskaj, Denmark
email: kaare.jensen@nokia.com and soren.riis@nokia.com

ABSTRACT

This paper describes an improved input coding method for a text-to-phoneme (TTP) neural network model for speaker independent speech recognition systems. The code-book is self-organizing and is jointly optimized with the TTP model ensuring that the coding is optimal in terms of overall performance. The code-book is based on a set of single layer neural networks with shared weights. Experiments show that performance is increased compared to the NETTalk and NETSpeak models.

1. INTRODUCTION

In a phoneme based speaker independent speech recognition system with a dynamic vocabulary, a strategy is needed for defining the vocabulary in terms of phonetic transcriptions. The basis for defining the vocabulary is in most cases written text, which means that some method for converting written text to phonemes is required. For many languages like, e.g. Japanese and Finish, there is a simple set of non-ambigous rules for converting graphemes (letters) to phonemes. However, for some languages like, e.g. English and Danish, no simple set of translation rules can govern the mapping at a satisfactory level of accuracy. For such languages, the mapping is often solved by combining a lexicon lookup with a statistical model that is able to generalize to words that are not present in the lexicon. The better mapping accuracy the statistical model provides, the smaller the lexicon needs to be.

Speech recognition applications implemented on embedded portable devices like mobile phones inherently suffer from limited memory and computational resources. Even though these resources are expected to increase for next generation of portable devices, the number of applications required to run simultaneously are also very likely to increase. It is therefore very important to minimize resource consumption of any application intended for a portable embedded device. Thus, for a speaker independent phoneme based recognition system, the TTP mapping module should be kept as small as possible.

A highly compact and well performing implementation of a phoneme based speaker independent speech recognizer is possible to realize for dedicated tasks. An example of this is the British English speaker independent phoneme based recognizer presented in [8, 6]. As shown in [8], this recognizer only needs about 6 kB of memory for storing phoneme models to yield the same performance as a standard Hidden Markov Model based recognizer requiring 10–14 times more memory. This is far below the memory required by most TTP systems, especially if a lexicon is included as part of the system. Thus, in terms of memory usage the bottleneck in implementing a phoneme based recognizer on a portable device is the TTP module.

A widely used approach for statistical TTP models are the so-called decision trees [5]. However, if the vocabulary in the application is unconstrained, the decision trees will typically be very large in order to provide an acceptable mapping accuracy. Neural networks provide an alternative solution to model based TTP mapping. Contrary to a decision tree, which can be viewed as an efficient way of storing (compressing) the mappings in a lexicon, the neural network aim at learning a functional relation between written text and the corresponding phonetic transcription. As shown in [2], a neural network based TTP model requires a factor of 2–4 times less memory to obtain the same performance as a decision tree based TTP model on words that are not present in the data used for generating the models. For an embedded system, where the available memory may prohibit using a lexicon lookup, a neural network might thus provide a small well generalizing TTP model.

2. DATABASE

The TTP models described in this paper are for American English and the dictionary used for training and testing the models is the freely available Carnegie Mellon University (CMU) pronunciation dictionary. In order to prepare the dictionary for training, the letters and the phoneme transcriptions have to be aligned such that there is a one-to-one correspondence between letters and phoneme symbols. The alignment is based on the Viterbi algorithm and aims at minimizing the cost of inserting null phonemes and pseudo phonemes in the phoneme transcriptions in the dictionary. The null phonemes are inserted to handle cases where a letter does not map to a phoneme and the pseudo phonemes are used when one letter maps to more than one phoneme (like the ‘x’ in fax). This is the same approach as used in [5].

To make the CMU dictionary as unambiguous as possible, all multiple pronunciations have been removed before the alignment is performed. The CMU dictionary (with a single pronunciation for each word) has been split in two parts by randomly selecting 20% of the entries for an independent test set. The remaining part of the dictionary is used for training the neural network based TTP models.
In order to present a letter to the network it has to be converted to some numeric quantity. In NETTalk [9] an orthogonal code-book was used as shown in Table 1. The last row in the table is the code for the graphemic null. For the English alphabet this results in a code size of 27. An important property of the NETTalk coding scheme is that it does not introduce any correlation between different letters.

<table>
<thead>
<tr>
<th>letter</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1 0 0 0 ... 0 0 0 0</td>
</tr>
<tr>
<td>b</td>
<td>0 1 0 0 ... 0 0 0 0</td>
</tr>
<tr>
<td>c</td>
<td>0 0 1 0 ... 0 0 0 0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>y</td>
<td>0 0 0 0 ... 0 1 0 0</td>
</tr>
<tr>
<td>z</td>
<td>0 0 0 0 ... 0 0 1 0</td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0 ... 0 0 0 1</td>
</tr>
</tbody>
</table>

Table 1: Orthogonal letter code used in NETTalk.

Letters are presented to the network one at the time and for each presented letter the network gives an estimate of the most probable corresponding phoneme. In order to take the grapheme context in the word into account, a number of letters on each side of the letter in question can also be used as input to the network. Thus, the network input is a window of letters. This is shown in Figure 1 as the letters $L_2 \ldots L_1$ for a context size of two letters. The center letter $L_0$ is the letter that corresponds to the output of the network. A graphemic null is defined in the character set and is used for representing letters to the left of the first letter and to the right of the last letter in a word. Since the neural network input units are continuous valued, the letters in the input window need to be translated to a specific code vector. Various methods for generating the code vector for each letter in the input window will be described in more detail in Section 4.

The TTP neural networks in this work are all fully connected multi-layer perceptrons, which use hyperbolic tangent sigmoid shaped functions in the hidden layer and a softmax normalization function in the output layer. The softmax normalization ensures that the network outputs are in the range $[0; 1]$ and sum to unity. It can be shown that a neural network with softmax normalization will approximate class posterior probabilities when trained for $1$ out of $N$ classification and when the network is sufficiently complex and trained to a global minimum [1].

4. INPUT LETTER CODING

In order to present a letter to the network it has to be converted to some numeric quantity. In NETTalk [9] an orthogonal code-book was used as shown in Table 1. The last row in the table is the code for the graphemic null. For the English alphabet this results in a code size of 27. An important property of the NETTalk coding scheme is that it does not introduce any correlation between different letters.

It may, however, not be optimal to fully decorrelate the different inputs. From a pronunciation point of view, it is evident that some correlation does actually exist between the different letters in the English alphabet. In fact, introducing some correlation between the letters might actually be beneficial for learning the functional relation between written text and phoneme transcriptions. Thus, instead of an orthogonal encoding of the letters, a better approach might be to let the code-book be adaptable and to jointly optimize parameters of both the code-book and the TTP model so as to maximize overall mapping accuracy. Furthermore, if the memory available for storing the TTP module is fixed a priori, the self-adapting code-book may also provide a good solution for designing a neural network based TTP model, which gives a good compromise between complexity and mapping accuracy.

For the NETTalk coding scheme, the neural network based TTP model size can only be limited by reducing the number of hidden units or the number of consecutive letters to use in the input window.

The NETSpeak [4] approach addresses the issue of correlation by grouping the 26 letters into five different closely related categories (vowels, consonants, etc.). Each group contains a maximum of six members. Five elements of the code vector are used for coding the group and six elements are used for coding each member within the group. This results in a code size of 11 elements. This coding reduces the size of the input layer with almost 60% compared to the orthogonal code but it also introduces correlation between members of different groups which may not necessarily be beneficial for learning the functional relation between written text and phoneme transcriptions.

5. SELF-ORGANIZING CODE-BOOK

In order to find an optimal coding scheme an extra layer, the code layer, is added before the input layer of the TTP neural network during training. After training the extra layer is decoded into a code-book and the TTP neural network (without the extra layer) can now be used together with the generated code-book.

5.1. Outline of the Full Network

Figure 2 shows an outline of the full neural network based TTP model during training and generation of the self-organizing code-book (SOC). In addition to the basic TTP network described in Section 3, it contains a number of parallel blocks C, the code layer, inserted between the letter input window and the TTP network. These coding-blocks are fully connected single layer per-
ceptrons which act as an adaptable preprocessor that generates the inputs for the basic TTP network. The outputs of each of the code-block networks are fully connected to the input layer of the basic TTP network.

The inputs for the code layer networks are the orthogonal code in Table 1 and the weights of the code-block networks are shared such that a weight between input unit \( i \) and output unit \( j \) is the same for all code-block networks. This means that the code-block networks are actually multiple replica of the same network. This method is known as weight sharing [1] and has been applied successfully to other similar tasks like e.g. prediction of protein secondary structures [7] and written character recognition [3].

By minimizing one global error function for all weights in this combined network it is ensured that the outputs of the code layer will have an optimal correlation in the input space for the given training data, model size and configuration.

5.2. Generation of the Code-book

When the model has been trained (as will be described in Section 6) the SOC-book is generated by detaching one of the C-blocks in Figure 2, applying the orthogonal letter code in Table 1, and reading the SOC at the output of the block. This code can then be used as a code-book in the way as the orthogonal code-book used in NETTalk.

6. NEURAL NETWORK TRAINING

The TTP network extended with the code-layer is trained by the standard back-propagation algorithm augmented by a momentum term. The aligned dictionary is used as training data for the neural network. Each letter with context and the corresponding phoneme make up one training example. One pass through the training set is denoted an epoch. Weight update is performed in a stochastic on-line fashion, i.e., the weights are updated after presentation of each training example picked at random from the training set which has been shuffled. The training set contains almost 650,000 examples and it is highly redundant as there are many identical examples of mappings between a specific context and a specific phoneme. For such large redundant training sets, we have found that stochastic online training provides faster training compared to batch training where the weight update is based on the average gradient for all training examples.

Before testing the models, all parameters were rounded off to eight bits as this was found sufficient for representing model parameters without a significant loss in accuracy. The number of parameters in the models therefore equals the required memory for storage in bytes.

The outputs of the TTP network approximate the phoneme class a posteriori probabilities and the estimated phoneme transcription for a word is simply generated by choosing the most probable phoneme for each letter in the word.

7. EXPERIMENTS

To estimate the optimal code size five different models have been trained with code sizes from 10–30 in steps of five. All models were kept equal in size by adjusting the number of hidden units. The optimal code size was estimated based on the performance on the training set. Figure 3 shows the results for a model size of 10 kB and a letter context of three. Results for the NETTalk and NETSpeak models have been included for reference. The figure shows that a code-size of 15 is optimal for this model. Similar experiments for a 40 kB model show results consistent with this.

Even though the NETTalk model and the network with a SOC code-book of the same size (27) as the orthogonal code-book contains the same number of hidden units, the network with SOC coding gives the best performance. This indicates that the SOC coding captures correlations between letters that are beneficial for the TTP neural network.

To investigate the effect of the SOC for different model sizes, four different networks have been trained for both the NETTalk, the NETSpeak, and the SOC code-book of size 15. Figure 4 and 5 show the results for the training and test data respectively. The model sizes are 10, 20, 30, and 40 kB. As seen from figure 4, the SOC improves performance on the training data regardless of model size. Interestingly, the NETSpeak code does not seem
to offer any performance improvement over the NETTalk code.
For the 40 kB models, the SOC scheme provides a relative error rate reduction of more than 10% compared to the NETTalk and NETSpeak approaches. Figure 5 shows performance on the independent test data. The performance improvement of the SOC is decreasing with model size. This indicates that introduction of some regularization or early-stopping method might be applied with success. As for the performance on the training data, it is again observed that the NETSpeak code does not offer any improvements over the NETTalk code on the test data.

### 8. CONCLUSION

In this paper a self-organizing code-book for input coding of letters for a compact text-to-phoneme neural network model is presented. The code-book is generated by adding an extra layer to the network during training which utilizes weight sharing. This method has previously been applied for similar tasks with success.

The method is compared to the NETTalk and NETSpeak approaches on the pronunciation dictionary from Carnegie Mellon University and results showed that error rate can be reduced by more than 10%.

### 9. REFERENCES