A Combination of Neural Network and Low-Level AI-Techniques to Transcribe Speech into Phonemes

Kari Torkkola [1]

Helsinki University of Technology
Laboratory of Information and Computer Science
TKK-F, Rakentajanaukio 2 C, SF-02150 Espoo, FINLAND
email: Kari.Torkkola@hut.fi

Abstract

An approach to automate knowledge acquisition from natural signals, such as speech, is presented. The knowledge is extracted in the form of context-sensitive production rules that can be used to map a signal or a sequence of events into another one, e.g., for correction or enhancement purposes. The rules are constructed automatically from examples by using a concept of Dynamically Focusing Context (DFC), which is an extension of Dynamically Expanding Context (DEC) introduced in [7].

An application is presented, in which vector quantized (VQ) speech is transcribed into phonemes. As vector quantizers, Kohonen's Feature Maps have been used. The proposed method is used to generate rules from VQ-codes that enhance phonemic differences and cancel coarticulation effects. A 38 per cent reduction in phonemic transcription error rate has been achieved.

1 The problem and an application

The problem addressed in this paper is to transform a signal into a desired one aided by local context. The transformation is not algorithmic but has to be learnt from a set of examples. We can thus talk about data-dependent filtering opposed to algorithmic fixed-window filtering common in signal processing. This problem is approached by using a principle developed by [7] to automatically derive a large set of context sensitive transformation rules. This principle has been named Dynamically Expanding Context or the DEC. Since these rules are automatically inferred from examples, we could talk about low level AI.

The approach in this paper is close to that of the DEC, except that the formulation of context is more general and is based rather on the relevance of the context to a segment to be transformed than on the distance of the context from this segment. Of course, there are cases in which these two formulations are equal.

To illustrate this approach, an application in phonemic speech recognition is presented. The goal is to transcribe spoken utterances into strings of phonemes as accurately as possible. The proposed method is used to transform vector quantized speech in order to enhance phoneme differences and to cancel coarticulation effects. The symbols for the rules are provided by using artificial neural network models called the Self-Organizing Feature Maps [11].

2 Transforming streams of events by local context

The general problem statement can be formalized as follows: As examples we have a source signal $S(t)$ (or several samples of disjoint $S_i(t)$), which we would like to map to another signal $T(t)$

[1] Current affiliation: IDIAP, C.P. 609, CH-1920 Martigny, Switzerland, email: karit@idiap.ch

(or $T_i(t)$), the target signal. The signals can be discrete or continuous in time, and the signal values can be continuous, discrete, or symbolic. Around each $S(t_m)$ to be mapped to corresponding $T(t_m)$ local context $C(S, t_m)$ is taken advantage of. This context can be, for example, the range $[S(t_m - l(t_m)), S(t_m + r(t_m))]$, where $l(\cdot)$ denotes the left context and $r(\cdot)$ denotes the right context. In order for this scheme to be meaningful, it is assumed that $C(S, t_m)$ bears a relationship to $S(t_m)$ meaning that the signal is not entirely independent of its local context. Note that the amount of context used at each instant tm is not fixed but depends on the data. The dimension t from which the context is extracted can be time but it is not limited to that only. Consider multidimensional signals, for example, one pixel or any segment in a sequence of video frames. The context can be taken from the same frame only, or from several consecutive frames but from the same pixel position only, or it can be any combination of these two alternatives.

When the examples are numeric, there are many possible ways to create a parametric mapping from a set of source examples to target examples. The mapping is usually restricted to a certain fixed model, e.g., linear in the simplest case, the parameters of which are estimated from examples by minimizing some kind of a matching error criterion [2]. The problem lies then in choosing a model that corresponds well enough to the examples.

Pattern recognition algorithms can also be viewed from the mapping point of view. These methods map a feature vector into class membership. By extending the interpretation of output from strict class membership into any arbitrary value (discrete in the easiest case), and using the local context $C(S, t_m)$ as features, these algorithms can also be interpreted as arbitrary mapping functions.

However, if the examples are non-numeric, streams of symbols, other means must be used. There are not too many approaches described in the literature, that are able to learn this kind of a transformational mapping. One example are the transformational grammars, see, e.g., [18]. From the artificial neural network (ANN) literature we can find one such method very popular lately, namely multilayer feed-forward ANNs trained using error back-propagation [19] or some other variant of gradient descent. These networks consist of simple units that compute a weighted sum of their inputs, and deliver a single output limited with a sigmoid function. These units are arranged in a layered feed-forward fashion. The units accept continuous values, but symbolic inputs are possible by appropriate, usually binary coding schemes.

In context-aided mapping, the network has a fixed-width (w) input window covering $[S(t_m - w/2), S(t_m + w/2)]$. $S(t_m)$ denotes the source value to be transformed to $T(t_m)$, which can in this case be a vector with continuous values, a continuous scalar value, or a symbol. One such application is given in [21], in which English text is converted into phonemes using a context window of seven letters.

A completely different method to learn a transformational mapping from examples is the Dynamically Expanding Context (DEC) introduced in [7]. This method builds automatically context-sensitive rules from examples to transform examples into targets. The input signal is required to be discrete in time with discrete values or discretized continuous values. The output can have discrete (symbolic) or continuous values. A merit of the method is that the input window coverage is variable. Rules are generated that use as narrow context as possible. A more complete description is given in the next section.

3 Dynamically expanding context

3.1 The algorithm

The central idea behind the method is to derive unique symbol to symbol(s) transformation rules from two streams of symbols, the source stream, and the desired target stream. The two streams must first be aligned symbol by symbol. To create initial context-free transformation rules, these aligned streams must be segmented somehow, for example, by symbols. One segment of the source stream acts as the condition part of a rule, and the corresponding segment of the target string constitutes the production part. Now, these initial rules may conflict each other, which means that two rules with equal condition parts may have different productions. These cases are specialized by dynamically expanding the discrete context around the initial condition parts by just a sufficient amount to make the rules unique (see fig. 1). The rules will then have condition parts of variable length. In this way the generality of the rules will be optimum, and still, all conflicts within the training cases are solved. For details of the algorithm, see [7]. In practice, the context could be limited to a certain maximum amount. This limit should be chosen specifically for each application so that context beyond this limit should have no effect on the centre segment anymore. The remaining conflicts are then handled by the estimation procedure described below.

context level	condition part
0	k
1	nk
2	nkh
3	inkh
4	inkhi
:	:

Figure 1: *Expanding phoneme level context from string /elsinkhi/ around phoneme /k/.*

The whole set of rules can be stored as a vast tree structure. Each node consists of a rule, and is accessible by the condition part of the rule. A node has a branch for every different exception to the rule, i.e., every different context expansion. Hence, the final leaves of the tree constitute of rules for which no more context expansion was necessary. These are the valid production rules. Every intermediate node above a leaf is then a more general rule with less context used, and represents conflicting training cases.

To transform a new symbol string employing these rules, only a fast tree search has to be done for each segment of the string. The tree has to be traversed starting from the root and branched according to context in the string around the segment. If a leaf is reached, the segment to be transformed is replaced by the production. However, if the search terminates before reaching a leaf, a novel context has been encountered. In this case we have ended up at an intermediate node, which has no branch according to context expansion to be done next. These kind of cases can be handled by the following procedure. For every intermediate node of the tree, that is, for every conflicting rule, estimates of the most probable productions are constructed. To this end, all substrings that match the condition part of this node are searched from the training material, and corresponding productions are enumerated. The most frequent one represents the most probable estimate of the production, given the amount of context. If the search now terminates before a leaf, this estimate at the deepest level reached is used as the production. This procedure applies also to cases where the maximum allowed amount of context was reached.

3.2 Some applications of the DEC

Originally, the DEC-method was used to correct erroneous phonemic transcriptions [7]. These transcriptions are obtained from a speech recognition device not considering coarticulation effects at all, and therefore they contain about 10-30phonemic transcriptions or even orthographic forms of the utterances. Dynamic programming is used to align the two streams. By using the DEC-algorithm, a large number of context sensitive rules is derived that describe the structure of errors and coarticulation effects in example transcriptions. The rule set is thereafter used to correct similar errors from new transcriptions with high performance [7].

In this correction task the power of the DEC emanates from the fact that closer phonemic context contributes more to deterioration induced by coarticulation. The DEC automatically ignores more remote context when it is not necessarily relevant to a correction rule.

In addition to signal-to-signal mapping, the DEC can be used to relate one signal to itself, for example, to predict its future values. In this case the context is expanded only in the past, and the production will be the next value of the signal. The method has been used in the domain of music to build rules describing certain example musical pieces. The rule set has thereafter been used to generate new pieces of similar style [9]

3.3 Connection to decision trees

The DEC bears a close resemblance to decision trees. Decision trees are classifiers that proceed in a hierarchical fashion. The nodes in the tree are partial decisions, and the final leaves represent the classification. Decisions closer to the root are usually more obvious and natural than later decisions, which make subtle distinctions between classes. A partial decision at a node usually requires to examine information relevant to this decision only. This makes the decision trees appropriate to situations where all the features are not available at once, or where the cost involved in determining the values of certain features may be high. The difficulties in using decision trees lie in designing the tree structure, in choosing the features to be used in decision making at each node, and in designing appropriate decision functions for each node. All this must be based on the features of input examples. A thorough investigation of decision trees can be found from reference [3]. This reference concentrates on the domain of pattern recognition, where features are usually numerical values.

From the AI-literature one can find methods to build decision trees from inputs with symbolic features. One such method is the ID3 (Iterative Dicothomizer 3) [17] based on the CLS (Concept Learning System) by [4]. The algorithm starts from the empty root with all example cases. At each node one single feature is selected, and a new branch is generated corresponding to every different possible value of this feature. The examples remaining in the node are propagated into corresponding branches according to this feature. These steps are repeated for every node that still contains examples of two or more different classes. When the algorithm terminates the leaves contain only single class examples.

The basic question is how to select a feature to be used at each node. The criterion in ID3 is to select a feature which provides the greatest increase in information content among the training cases. The algorithm has found many uses, even commercial, and has done well in comparisons with other concept-learning algorithms [1]. In the AI-literature such an algorithm is called "concept-learning", since the decisions in the tree can be interpreted as a collection of descriptions of different classes or concepts.

Since the rule set built by the DEC-algorithm is actually a tree structure, the DEC can be viewed as a fast algorithm to build a decision tree from natural data. Local context now constitutes the features of a segment of the input stream, and these features are used in fixed expanding order as tree building proceeds. This is, of course, due to the fact that in speech application (and in general with natural signals) the relevance of local context to a certain part of the signal is usually proportional to its closeness to the part. With natural signals, features are in theory available from as far as we wish to go, but the DEC terminates when all conflicts among the examples are covered. The actual tree-building algorithm of the DEC will produce the same tree as ID3 would, if local context were used in ID3 as features in the same expanding order, and if the production part of a DEC-rule were interpreted as a classification decision. However, the algorithm of the DEC is faster, and requires only serial access to training cases. Moreover, ID3 has no provision for the estimation procedure employed in the DEC in case of a novel feature value.

4 Dynamically focusing context

Consider applying the DEC to data that is so fine-grained that expanding context by a single discrete step does not add much relevant information to condition parts of the rules. One such example is a digitized image. Suppose we have images corrupted by some process and we desire to build a context-dependent transformation from examples to do the restoration. Straightforward application of the DEC-algorithm would start with initial rules of pixel-to-pixel correspondence between a corrupted image and its original version. In case of a conflict these would be specialized by adding a new pixel value farther away to the condition part.

In order for the DEC to be an appropriate solution to a problem, the initial context-independent rules should be as general as possible. Further context expansions should then handle exceptions to these general rules. In the case of images, single pixel value to pixel value mappings do not represent much generality, and taking another nearby pixel values into account does not much specialize them. Therefore, for the case where there is a lot of fine-grained data, a new approach to using context in the DEC-algorithm is presented. It is based on context relevance, not on context distance. Rather than to expand the context area in size, the idea is to focus the details of a larger, possibly fixed-size area of context during the specialization process. Hence the expression, Dynamically Focusing Context or DFC.

As initial context, a larger section of data is taken including and surrounding the area to be transformed. A single discrete symbol is then computed to characterize this fine grained data, and this symbol is taken as the condition part of the rule. In case of a conflict, that is, among the training examples there are two rules with equal condition parts but different productions, the contexts have to be specialized. The initial section of data is then divided into smaller parts, from which their own characteristic symbols are computed and concatenated to construct the condition part of a rule. This context focusing can be continued until no conflicting rules remain, or no more data division is possible. A maximum depth to stop the focusing can also be defined. The remaining conflicts beyond this depth are then taken care of using the estimated most probable productions (cf. sec. 3.1). With this approach the original input stream does not have to be discrete valued, and not even discrete in time, as long as the actual symbols computed from the stream are discrete.

In the following section an application is presented that employs this technique in phonemic speech recognition. In this approach speech frames are vector quantized (VQ) into code sequences. Code sequences deteriorated by coarticulation effects and by an unideal acoustic processor of a speech recognizer are transformed closer to ideal ones using a rule set built by the DFC. In such ideal sequences code symbols remain constant inside phoneme areas and change sharply at the borders facilitating easier and more accurate conversion into phonemic transcription.

5 Application of the DFC to phonemic speech recognition

5.1 Background

Word-based approaches to speech recognition are almost impossible in many languages, because the words may have numerous different inflected forms. To obtain sufficient coverage for dictation tasks in, e.g., office environments, all these forms should be included in the dictionary as distinct entries. This would expand its size too much. Therefore, in these languages the first stage of a speech recognizer should distinguish units smaller than words. *Phonemes* act as these units in the recognizer employed here [8]. The goal is to transcribe spoken utterances into phoneme strings as accurately as possible using methods that learn from examples. These phoneme strings can then be the basis for further, e.g., linguistic processing.

The language used in this work is Finnish. The following characteristics of this language support the choice of phonemes as the basic recognition unit: 1) it contains only 21 phonemes, 2) the correspondence of written and spoken forms is almost identical, 3) the words are highly inflected, for example, a single verb can have over 1000 different inflected forms. The methods described are not limited to Finnish only, however. Their applicability to Japanese has already been demonstrated partly [12].

One major difficulty in phoneme recognition is the effect of coarticulation. The same ideal phoneme can have completely different appearances in different contexts, but it should always be recognized as the same symbol. One possible way to overcome this is to explicitly model phonemes in these different contexts, e.g., by triphone-based hidden Markov models (HMM) [20], [14]. Another way is first to recognize phonemes according to their actual appearance ignoring the coarticulation effects, and to correct the results afterwards aided by local context on the basis of the systematicity of the errors. The DEC-method can be used to construct correction rules for this case [7].

5.2 The baseline system

The speech recognition system, that is the starting-point of this research has been described in detail in references [7], [12], [8]. In this system artificial neural network models called the Self-Organizing Feature Maps (SOFMs) [11], [8] are utilized as vector quantizers to produce a so called quasiphoneme sequence (a phoneme code every 10ms) from the input speech. The SOFMs have been tuned using Learning Vector Quantization (LVQ) to differentiate phonemic classes [10], not only to represent the input feature space with minimum distortion as with conventional VQ-techniques [15]. Context-independent duration-based rules are then used to compress the quasiphoneme sequence into a sequence of phonemes [11]. The coarticulation effects and other systematic errors are corrected at phoneme symbol level by rules created using the DEC-method. The error rate of the system as the sum of inserted, deleted, and misrecognized phonemes varies between 5% and 15% depending on the speaker and on the speech material.

5.3 Application of the DFC to vector quantized speech

In research reported in [23] and [13] an approach was taken to handle coarticulation effects on a level below phonemes, namely on the level of VQ-codes. In the former reference, the idea of the DEC, automatic generation of context-sensitive rules, was applied rather straightforwardly to code sequence obtained from vector quantizer. The purpose was to use the rules to transform smooth or blurred transitions between phonemes to be sharper and more distinct, in addition to cancelling the effects of coarticulation. The latter paper tried to cope with the same problem using back-propagation networks. The approach taken here extends that research. An alternative to context expansion, namely context focusing, is explored at grammar building stage. To provide symbols from input speech signal for the rules, different methods based on SOFMs to discretize sections of speech are studied. Some strategies to employ the transformation rule set are experimented with, too. The best one turned out to be to recover the ideal code sequence from the one that has been deteriorated by coarticulation and by the acoustic processor of a speech recognizer. The result will then be a kind of a phonemic restoration filter.

As initial context, a section of vector quantized speech data is taken, represented by a single discrete symbol, which is computed as described in sec. 5.7. The production part of the rule can be, for example, the symbol of the correct phoneme which is supposed to lie in the centre of this section of data. These target phoneme symbols are given by an automatic system that aligns the speech waveform with the corresponding phonemic transcription [22].

The purpose is, then, to transform the section of speech characterized by the condition part of the rule into the specified target symbol. A certain amount of training examples, for example, five minutes of vector quantized speech is scanned and such rules are recorded. In case of a conflict, the initial sections of speech data are divided into smaller parts, from which their own characteristic symbols are computed and concatenated to the condition parts of the rules. This context focusing is continued as explained in section 4.

Three strategies to build and utilize the transformation rule set were explored with. For all of the strategies, the condition parts of the rules are computed in the same way, as will be explained later in sec. 5.7 more accurately. The strategies are:

1. By separate means locate first "significant speech events" (e.g. centre points of phonemes) and build a collection of rules that only classifies these events into phoneme classes or discards them as transitions between phonemes. The burden of locating phonemes is thus relieved from the set of rules.
2. Build a rule set that produces phoneme labels only from the centre parts of phonemes. Between them the target to be produced is a special transition symbol. In this case our aim is to have a set of rules to locate and classify phonemes automatically.

3. Build a rule set that transforms the code sequence given by the vector quantizer closer to the ideal quasiphoneme sequence. Phoneme symbols are used as targets. The corrected quasiphoneme sequence will then be the starting point for duration-based rules to be converted into phonemes.

5.4 The setup of experiments

Three male speakers were available for experiments. The database consisted of four repetitions of 311 utterances totalling 1897 phonemes. Each utterance was less than two seconds of duration. All results given are averages of four leave-one-out tests, that is, three repetitions were used for training and one for testing, rotating the test set. All results are also speaker-dependent unless otherwise mentioned. Unvoiced plosives /k/, /p/, and /t/ have been treated in the results as a single class, since they are to be recognized using a separate transient map method [6].

5.5 Experiments with phoneme detection and classification

To evaluate the first strategy, a grammar was built only from the centre points of phonemes. Applying the rules likewise only at the phoneme centres of the test data gives a phoneme recognition accuracy of 92%. Had we a perfect method just to locate these points, 92% accurate phonemic transcriptions could be produced with these rules.

It could, however, be possible to cope with less ideal phoneme locator methods as long as the rule set is also trained with phoneme positions found by the method. The rules could then be able to learn the unidealities or systematic context-dependent errors of the phoneme locator. The locator should be slightly over-sensitive producing more extra phoneme positions than failing to find a phoneme, since rules can be generated to discard such positions by using a null output symbol as a production. No such experiments were made, however.

The aim of the second strategy is to have a set of rules to transform the VQ-code sequence so that phoneme symbols are produced close to phoneme centre areas only, and special transition symbols are produced at transitions between phonemes. It would thus be a relatively easy task to pick the correct phoneme symbols from the stream of productions and to build the corresponding phonemic transcription of the utterance.

Since we have a system that points out which parts of the VQ-code sequence belong to which phonemes [22], we can freely select what to use as target symbols for different instants of the phoneme (they need not be the correct phoneme symbols). Figure 2 displays several alternatives for target symbols. In the experiments, it turned out to be disadvantageous to produce transition symbols at all. Only phoneme symbols are worth producing. Especially in case of some short phonemes, e.g., voiced stops the production of transition symbols reduces the reliability to recognize this phoneme. This is due to decreased amount of successive correctly produced phoneme symbols that are needed to make an accurate decision.

5.6 Experiments with different speech signal representations

Using SOFMs as vector quantizers gives several alternatives to construct symbols characterizing sequences of quantized speech frames. If a SOFM has been after-tuned by LVQ to optimize phoneme recognition accuracy, the code sequence will be a sequence of phoneme labels. To characterize such a sequence, one could use either simple or weighted majority voting over the code symbols appearing in the sequence. Another approach is to take advantage of the local topology -preserving property of SOFMs [8]. This means that on a two dimensional code vector grid of a SOFM, two adjacent vectors are also physically alike. Now, consecutive speech frames (represented by feature vectors) are similar due to continuous nature of speech. As these frames are quantized by a SOFM, the corresponding code vectors are usually close to each other on the map grid, and constitute thus a path or a trajectory on the SOFM. A sequence of speech frames can then be discretized as mean or

```
A: vvvvvvuuoooaaaaaaaaaaaaaääeäeiiiiiiiiinnnnnnnnnnnnnnaae
B: vvvvvvaaaaaaaaaaaaaaaaaaaaaiiiiiiiiiiinnnnnnnnnnnnnnnnn
C:  vvvv    aaaaaaaaaaaaaaa     iiiiiii     nnnnnnnnnnn
D: -vvvv----aaaaaaaaaaaaaaa-----iiiiiii-----nnnnnnnnnnn---
E: --!!--------!!!!!!!!!----------!!!---------!!!!!!!-----
```

Figure 2: *Alternatives for target productions. Row A: The original code sequence produced by word /vain/. Row B: The ideal code sequence (that also displays phoneme borders) can be used as such as the target. Row C: Use only centre parts of the phonemes for training and discard transitions. Row D: At transitions use a special symbol as the target production. Row E: Only two different symbols are used as targets; one for the transitions and another for the centres. This alternative can be used only to locate phonemes.*

median position of corresponding trajectory on the two dimensional SOFM space. For illustrations of speech trajectories on a SOFM see [11] or [8].

However, the comparison of pure SOFM-coordinates with phoneme codes as the basis for condition parts showed a clear advantage to phoneme codes. With both alternatives, the rules will explain the training data perfectly, but rules with SOFM-coordinate-based symbols are not able to generalize well enough for test material. It seems that with these symbols the training examples are too specific, and cannot cover all the necessary cases. Thus, when searching the rule base, the exact match requirement for condition parts is not enough. One should search for the most similar rule according to some measure. The latter case would make both the building and correction stages very slow, because the similarity measure should be evaluated to every existing rule.

It seems natural that phoneme codes contain the necessary information for coding, because the SOFMs have been particularly tuned with LVQ to clarify the differences between phonemic classes. Further generalization by using phoneme codes is due to the fact that a SOFM has many codevectors carrying the same phoneme code. These codevectors represent different allophones of the same phoneme class. It is obvious that if a separate code for each codevector is used, more specific rules will be produced. Utilization of the map topology by medians or means of trajectories did not help. Thus, the rest of the experiments were made using phoneme codes, that permits the usage of exact rule match with the fast rule tree structure.

5.7 Experiments with quasiphoneme transformation

The best phonemic recognition results were achieved with the third strategy in which the rules are used to transform the VQ-code sequence closer to the ideal quasiphoneme sequence. This sequence is then processed by duration-based rules to be converted into a phoneme sequence. Correct phoneme symbols function as target productions of the rules. No special transition symbols are used.

The exact way by which the condition parts are gradually specialized is as follows: The initial condition part at context level zero is based on a 90 msec window of codes (9 symbols) centred at the code symbol to be transformed. The symbol to be the condition part will be the simple majority of codes in this window. At the next two specializing stages the context expands 160 msec to both sides of the initial window. Thereafter only focusing is used to specialize the rules so that these three parts are gradually decomposed into smaller ones, from which the majorities are again computed. Figure 3 displays a part of the quasiphoneme sequence of word /voidaan/, and how the condition parts will be specialized at conflict situations. The latest specialization at each level is indicated by heavy lines. The production part of the rule is the target quasiphoneme symbol in the centre of the window (/d/), which is equal to the correct phoneme symbol. The rule derived from this point states that the centre symbol, which in this case is /v/, is to be replaced by /d/ in the context represented by the condition part. Notice that small variations in the code sequence are smoothed out due to majority computation, contributing to some amount of generality.

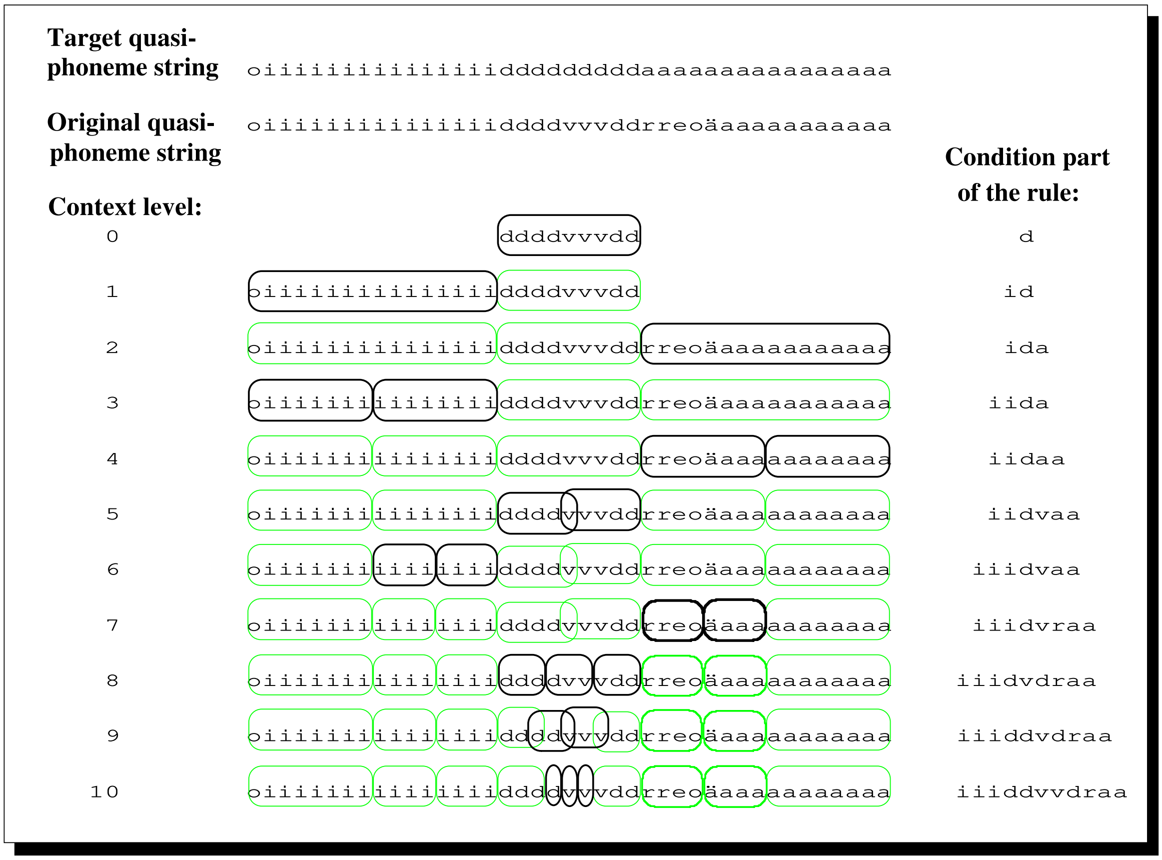

Figure 3: *An example of gradually specializing the condition part of a rule. The first two levels of specialization are done by expansion, the rest by context focusing. Here ten has been the maximum depth of focusing.*

Instead of simple majority voting to determine the symbol to describe a window, several kinds of weighted majorities were experimented with. Raised sine -like overlapping windows were tried in the middle windows and decaying windows at the edge areas. None of these, however, outperformed the simple majority computation.

Table 1 displays the effect of the DFC-correction to phonemic recognition rate and to error rate. On the average 38% of the errors in the phonemic transcriptions can be handled with the method. All the results given are speaker-dependent. However, it is also possible to generate a multi-speaker rulebase. No change in the recognition accuracy was detected with a combined three-speaker rulebase, that is, the rules of different speakers neither disturb nor help each others.

We also compared the difference between pure context expansion and focusing for one of the speakers (kt). For context expansion, we used the same symbols as for context on level 10 in fig. 3; the context only expanded gradually around the centre symbol. We found out that expansion produced more rules (21000 vs. 15000), and corrected only 33% of the errors (vs. 42% of the DFC). A comparison was also made to the context expansion scheme studied in reference [23], in which the context was expanded by quasiphoneme labels. Long sequences of equal consecutive labels were compressed to one or two symbols to construct more general condition parts. This scheme could only handle 25% of the errors by using the same amount of context as the DFC-scheme (400 ms in this case).

An experiment was also conducted to determine how much there is overlap between the corrective effects of quasiphoneme-level DFC and phoneme-level basic-DEC. The combination experiment was conducted so that three data sets were used to produce a DFC-rulebase. The fourth data set was

	errors				correct		
speaker	orig	DFC	diff.	%corr.	orig	DFC	diff.
kt	14.30	8.26	6.04	42	87.82	93.03	5.21
pu	18.80	12.09	6.71	37	87.90	92.01	4.11
jk	18.35	11.70	6.65	36	89.42	93.20	3.78
average	17.15	10.68	6.37	38	88.38	92.75	4.37

Table 1: *The effect of DFC-correction to error rate. Column 'orig' denotes the result derived from the original quasiphoneme sequences. 'DFC' displays the results derived from the quasiphoneme sequences after DFC-correction, 'diff' shows their difference, and '%corr' states the percentage of errors in the original strings that have been corrected by the DFC.*

corrected by using the rulebase. Duration-based rules were used to convert this set into phonemes. The procedure was repeated four times leaving always a different data set aside to be corrected producing thus four sets of phoneme strings for the DEC-experiments. In these experiments, three sets were used for training and one for testing by the leave-one-out principle. The results are given in table 2 as averages over all speakers. We can see that alone the DEC or DFC give very similar results. However, using first DFC, and DEC thereafter, gives results in which about 20% of the errors left by pure DEC are corrected.

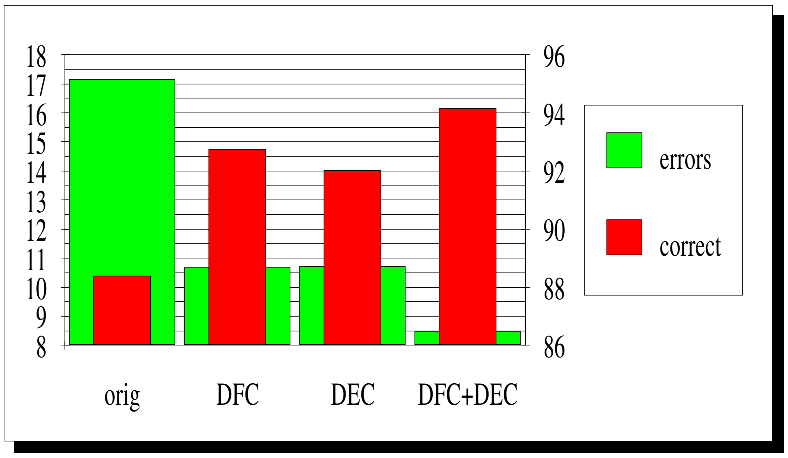

Table 2: *The overlap between phoneme-level DEC and quasiphoneme-level DFC. The left vertical axis shows the error rate, and the right one the amount of correctly recognized phonemes.*

6 Conclusions

In this paper an approach for automatic knowledge acquisition is presented that is applicable to data originated from natural signals. The knowledge is presented in the form of context sensitive transformation rules. These rules are generated automatically by using a principle named Dynamically Expanding Context (DEC) [7], but with dynamically focusing rather than expanding context (hence expression DFC). This new feature makes the algorithm more suitable to fine-grained data. The rule set can be used, for example, to transform signals in a way that is learnt from examples.

The property of being able to take into account only important context gives the DFC (and the DEC) an advantage over transformation methods that have to take a fixed width context into account with equal importance [21]. For one comparison, see [24]. Thus, the methods are feasible when it is possible to specialize transformation rules gradually so that early condition parts are composed of really more important and more general conditions that affect the symbol to be transformed (or the data represented by the symbol).

An application for phonemic speech recognition is described, that operates on vector quantized speech. Its purpose is to strengthen the phoneme differences so that coarticulation-based smooth transitions from one phoneme to another are sharpened. To this end, transformation rules using VQ-codes as input are generated by the DFC. Kohonen's SOFMs tuned by LVQ are used for vector quantization. In experiments, a 38% decrease of the phonemic recognition error rate was achieved. It thus seems worthwhile to incorporate the method into our speech recognizer [8].

In speech recognition literature, opinions have been given that recommend not to make any classification decisions too early, e.g., [16]. According to our experiments it seems to be advantageous to emphasize phoneme class differences early in the recognition process, on the contrary of these opinions. Same kind of conclusions have also been drawn elsewhere recently in the context of HMM-recognizers [5].

For each different application one has to study carefully how to define the initial contexts for the rules and how to specialize them in descending order of importance. For phoneme strings, the right way seems to be expansion, since closer phonemic context evidently contributes more to pronunciation of a phoneme. For sub-phonemic symbols, such as VQ'd speech, the answer is not that obvious, since we are actually trying to model two separate phenomena with the rule set: the acoustic processor including vector quantization, and the coarticulation effects. For effects due to purely coarticulation, context expansion would possibly suffice for the same reasons as for phoneme strings. However, due to our unideal acoustic processor and vector quantizer, the code sequences are not that smooth. They may contain abrupt variations partly because in speech there exist phenomena shorter than the used frame length (26,5 msec). The result of the vector quantizer may then depend on the position of these phenomena within a frame. Using longer windows with the context focusing approach these variations are smoothed out.

The experiments conducted with the combination of VQ-code level DFC and phoneme-level DEC suggest that integration of both DFC and DEC could be useful. This integrated system should use also segmental context at some level in addition to frame based context as used in the DFC now.

References

[1] A. Bundy, B. Silver, and D. Plummer. An analytical comparison of some rule-learning programs. *Artificial Intelligence*, 27:137–181, 1985.

[2] B.Widrow and S.D.Stearns. *Adaptive Signal Processing.* Prentice-Hall, 1985.

[3] G.R. Dattareya and L.N. Kanal. Decision trees in pattern recognition. In L.N. Kanal and A. Rosenfeld (eds.), editors, *Progress in Pattern Recognition 2.* Elsevier, 1985.

[4] E.B.Hunt, J.Marin, and P.J.Stone. *Experiments in Induction.* Academic Press, New York, 1966.

[5] H. Iwamida, S. Katagiri, E. McDermott, and Y. Tohkura. A hybrid speech recognition system using HMMs with an LVQ-trained codebook. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP90)*, volume 1, pages 489–492, Albuquerque, NM, USA, April 3-6 1990.

[6] Jari Kangas and Teuvo Kohonen. Using transient maps in classification of voiceless stop consonants. In *Proceedings of the First Expert Systems Applications World Conference*, pages 321–326, Paris, October 1989.

[7] Teuvo Kohonen. Dynamically Expanding Context, with application to the correction of symbol strings in the recognition of continuous speech. In *Proceedings 8th International Conference on Pattern Recognition (8th ICPR)*, pages 1148–1151, Paris, France, Oct. 27-31 1986.

[8] Teuvo Kohonen. The 'Neural' Phonetic Typewriter. *IEEE Computer*, 21(3):11–22, March 1988.

[9] Teuvo Kohonen. A self-learning musical grammar or associative memory of the second kind. In *Proceedings of the IJCNN, Int. Joint Conf. on Neural Networks*, volume I, pages 1–5, Washington, 1989.

[10] Teuvo Kohonen, Gyorgy Barna, and Ronald Chrisley. Statistical pattern recognition with neural networks: Benchmarking studies. In *Proceedings of the IEEE International Conference on Neural Networks*, pages I 61–68, San Diego, July 1988.

[11] Teuvo Kohonen, Kai Mäkisara, and Tapio Saramäki. Phonotopic maps – insightful representation of phonological features for speech recognition. In *Proceedings of the 7th International Conference on Pattern Recognition (7th ICPR)*, pages 182–185, Montreal, Canada, July 1984.

[12] Teuvo Kohonen, Kari Torkkola, Makoto Shozakai, Jari Kangas, and Olli Ventä. Phonetic Typewriter for Finnish and Japanese. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP88)*, pages 607–610, New York City, USA, April 1988.

[13] Mikko Kokkonen and Kari Torkkola. Using self-organizing maps and multi-layered feed-forward nets to obtain phonemic transcriptions of spoken utterances. *Speech Communication*, 9(5-6):541–549, December 1990.

[14] Kai-Fu Lee. *Automatic Speech Recognition: The Development of the SPHINX System.* Kluwer Academic Publishers, Boston, 1989.

[15] Y. Linde, A. Buzo, and R.M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Comm.*, COM-28(1):84–95, January 1980.

[16] J. Mariani. Recent advances in speech processing. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP89)*, pages 429–440, Glasgow, U.K., May 23-26 1989.

[17] J.R. Quinlan. Discovering rules by induction from large collections of examples. In *Expert Systems in the Micro Electronic Age (ed. D.Michie)*. Edinburgh University Press, 1979.

[18] R.C.Gonzalez and M.G.Thomason. *Syntactic Pattern Recognition.* Addison-Wesley, 1978.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In J.L. McClelland, D.E. Rumelhart, and PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 318–362. MIT Press/Bradford, Cambridge, 1986.

[20] R. Schwartz, Y. Chow, O. Kimball, S. Roucos, M. Krasner, and J. Makhoul. Context-dependent modeling for acoustic-phonetic recognition of continuous speech. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP85)*, pages 1205–1208, Tampa, Florida, March 26-29 1985.

[21] Terrence Sejnowski and Charles R. Rosenberg. Parallel networks that learn to pronounce english text. *Complex Systems*, 1(1):145–168, 1987.

[22] Kari Torkkola. Automatic alignment of speech with phonetic transcriptions in real time. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP88)*, pages 611–614, New York City, USA, April 11-14 1988.

[23] Kari Torkkola and Teuvo Kohonen. Correction of quasiphoneme strings by the dynamically expanding context. In *Proceedings of the 9th International Conference on Pattern Recognition (9th ICPR)*, pages 487–489, Rome, Italy, November 14-17 1988.

[24] Kari Torkkola and Kimmo Raivio. Comparison of symbolic and connectionist approaches to eliminate coarticulation effects in phonemic speech recognition. In *Proceedings of the European Conference on Speech Communication and Technology (Eurospeech89)*, volume 1, pages 9–12, Paris, France, September 26-28 1989.