Phoneme-to-grapheme conversion for out-of-vocabulary words in speech recognition

Bart Decadt, Walter Daelemans*
CNTS Language Technology Group, University of Antwerp
e-mail: {decadt,daelem}@uia.ua.ac.be

Deliverable WP2, March 31, 2001

Abstract

In this report, we show that Out-Of-Vocabulary items (OOVs), recognized using phoneme recognition, can be reasonably reliably transcribed orthographically using Machine Learning techniques. More specifically, (i) we show baseline performance of a machine learning approach to phoneme-to-grapheme conversion when different levels of artificial noise are added (simulating phoneme recognizer errors), (ii) we provide results on real phoneme recognition data, and (iii) we provide a detailed error analysis.

1 Introduction

One of the problems in speech understanding is the reliable recognition of words not present in the speech recognizer vocabulary (out-of-vocabulary items, OOVs). Current speech recognition technology makes use of (among other information sources) a restricted pronunciation lexicon (typically 40K words) to produce word graphs (lattices of possible sequences of words detected in the input) from which the most likely sequence is chosen. This approach cannot handle words not present in the restricted lexicon. A possible solution to this problem is to detect these OOVs in some way (using confidence scores provided by the speech recognizer), produce a phoneme string for them using a phoneme recognizer, and finally use a phoneme-to-grapheme converter to find a likely orthographic transcription of the OOV.

This report concentrates on the final step of this proposed solution: phoneme-to-grapheme conversion. A phoneme-to-grapheme converter takes as input a string of phonemes, generated by a phoneme recognizer, and gives as output a string of graphemes (the spelling of the word). However, the phoneme strings generated by a speech recognizer are not free of errors: a typical error rate for a phoneme recognizer is 25%.

The motivation for the experiments reported here is our hypothesis that machine learning techniques can adapt to the peculiarities of the errors made by a phoneme recognizer, and can provide the necessary robustness and accuracy to the phoneme-to-grapheme conversion task when provided with sufficient training data (pairs of words and corresponding output of the phoneme recognizer).

*Research funded by IWT in the STWW programme, project ATraNoS (Automatic Transcription and Normalisation of Speech). The research consortium consists of CCL and ESAT (K.U. Leuven), CNTS (U. Antwerpen), and ELIS (R.U. Gent). The project aims at generic basic research on speech recognition and normalization of unrestricted speech with automatic subtitling from speech as a case study. We would like to thank the project partners of ESAT for their cooperation in the work reported here. We would also like to thank our colleagues Véronique Hoste and Erik Tjong Kim Sang for their help.

In Section 2 we introduce the machine learning method we use throughout this report, and report on the results of applying it to a lexical database (a pronunciation lexicon) with varying levels of artificially generated noise. In Section 3 we report on experiments using a dataset of 129075 words of text with the corresponding output of the ESAT phoneme recognizer (Demuynck et al., 1998; Demuynck et al., 2000; Demuynck, 2001; Duchateau, 1998) [1]. Each of the following Sections also provides a thorough error analysis. Section 4 sums up and assesses the feasibility of the approach in practice.

2 Baseline Experiments

The experiments described here were carried out to find out what the accuracy of phoneme-to-grapheme conversion is in the ideal case (almost no errors in the phoneme transcription), and to find out in a controlled setting how much performance decreases when the error rate in the phoneme strings increases.

CELEX (Baayen, Piepenbrock, and van Rijn, 1993), a lexicon containing 173,873 Dutch words (or 1,769,876 graphemes) and their pronunciation, was taken as the dataset, and TIMBL (Daelemans et al., 2000), a memory-based learner developed at the University of Tilburg and the University of Antwerp, was used to carry out the task of phoneme-to-grapheme conversion. In the following subsections, the machine learning method will be introduced, and the design of the experiments and the method of artificial noise generation will be described. Finally, the results and an error analysis will be provided.

2.1 Machine Learning Method

For all the experiments reported here, we used TIMBL, a memory-based machine learner. Memory-based learning is based on the hypothesis that in domains like language processing, where relatively few regularities compete with many sub-regularities and exceptions, a *lazy* form of learning (keeping in memory all examples and using similarity-based reasoning on all examples at classification time) is superior to an *eager* learning approach (extracting rules or other abstractions from the examples and using these to handle new cases) (Daelemans, van den Bosch, and Zavrel, 1999). Furthermore, the results of research on a similar task (grapheme-to-phoneme conversion (Busser, Daelemans, and van den Bosch, 1999; Daelemans and van den Bosch, 1996; Hoste et al., 2000; Hoste, Gillis, and Daelemans, 2000; van den Bosch and Daelemans, 1993; van den Bosch and Daelemans, 1998)), suggest that memory-based learning may be very well suited for our similar task, phoneme-to-grapheme conversion.

TIMBL is a software package for memory-based learning implementing a wide range of algorithms, weighting metrics, and other parameters. It can take as input patterns (or instances) of feature values with a corresponding class symbol (supervised, example-based learning). During the learning phase, TIMBL stores all instances in memory and collects statistical data about these instances. To evaluate the performance of TIMBL on a task, a test set containing previously unseen instances is used: TIMBL predicts the class of these new instances by comparing them with the instances from the training set. The new instance gets the same label as the most similar instance(s) from the training set. We will describe here only the algorithms which we used in our experiments, for a full description of the implementation of all available algorithms and metrics, we refer to (Daelemans et al., 2000).

[1] The phoneme recognizer mentioned is not a separate component in ESAT's speech recognizer: for phoneme recognition, ESAT uses its speech recognizer with a vocabulary of 40 phonemes, instead of using it with a large vocabulary of 40K words. The context-dependent acoustic modelling and the statistical model of phoneme sequences (5-gram) were estimated on a dataset containing six hours of speech read aloud (different from the dataset used in our phoneme-to-grapheme conversion experiments).

The basic similarity between two instances is computed using an *overlap metric.* In the case of our symbolic, nominal data (phonemes as features), this means that similarity between two patterns is the number of features for which the two patterns have the same value. Obviously, this would in general give bad results as not all features are equally relevant for solving a particular task. We use an information-theoretic approach (*information gain* in its form normalized for number of values per feature; i.e. *gain ratio*, see (Quinlan, 1993)) to weigh the relevance of the different features. We will call this algorithm IB1-IG, introduced in (Daelemans and van den Bosch, 1992). Another factor of importance in memory-based learning is the number of neighbors that is taken into account to extrapolate from (the parameter k). Finally, we have used in our experiments the IGTREE algorithm (Daelemans, van den Bosch, and Weijters, 1997), a decision tree-based heuristic approximation of memory-based learning which is more efficient than IB1-IG.

2.2 Data Preprocessing

In the machine learning set-up we chose, each phoneme of each word is represented with its surrounding context as an instance or pattern that has to be classified with the grapheme corresponding with that phoneme (there are as many patterns to be classified as there are phonemes). This implies that to work properly, the grapheme and the phoneme strings for each word should be of equal length. As phonemic representations are rarely as long as their corresponding graphemic representation, they have to be *aligned.* When the string of graphemes is shorter than the string of phonemes, the null symbol '-' is inserted. An example is the alignment of the Dutch word *taxi* with its phoneme string /tAksI/:

$$/\text{t A k s i}/ \rightarrow \text{t a x - i} \rightarrow \text{taxi}$$

We inserted these null symbols with an *Expectation Maximization* (EM) algorithm (we used an already implemented version of EM, called ILKALIGN (van den Bosch and Daelemans, 1994, revised in 1999)): it starts by randomly inserting null symbols in the grapheme strings which are shorter than their phoneme string, and uses this initial alignment for computing probabilities for each phoneme-grapheme pair in the aligned strings (the *expectation* step). With these probabilities, ILKALIGN makes a more plausible alignment (the *maximization* step). This procedure is repeated as long as the total probability of the new alignment increases (Daelemans and van den Bosch, 1996).

In the reverse case, a shorter grapheme string, we used *compound graphemes.* For example, in the Dutch word *slaap* (sleep), with pronunciation /slap/, we replaced the graphemes *aa* with the *compound grapheme A*:

$$/\text{s l a p}/ \rightarrow \text{s l A p} \rightarrow \text{slaap}$$

The amount of context used during the experiments was set to three phonemes to the left and to the right of the phoneme in focus position. E.g., the Dutch word *kast* (cupboard) is represented as the four instances below:

Left context			Focus	Right context			Class
=	=	=	k	A	s	t	k
=	=	k	A	s	t	=	a
=	k	A	s	t	=	=	s
k	A	s	t	=	=	=	t

The last symbol in a pattern (i.e., the class to be output) always indicates the graphemic representation of the phoneme in focus, which here occurs in fourth position, while the other positions represent phonemes in the context of the focus phoneme, with the symbol '=' indicating a word boundary.

1. Compute for each phoneme the three nearest phonemes in terms of value difference.
2. Set a threshold between 0 and 1.
3. For the phoneme representation of each word
   - For each phoneme in the phoneme representation
     - Generate a random number between 0 and 1.
     - If the random number is below the threshold, replace the phoneme by a randomly chosen phoneme from the set of nearest phonemes for that original phoneme.

Figure 1: The algorithm used to add noise to the dataset

To avoid a possible influence of the alphabetic ordering of the CELEX data on the results, the order of the words was randomized at word level, before the patterns were created.

2.3 Adding Artificial Noise to the Data

In order to find out how much the performance of the machine learner decreases when the amount of noise in the data increases (in other words how robust and adaptive the machine learner is), the algorithm in Figure 1 was used to add a percentage of artificial noise to the data. Noise was added to CELEX at word level, before creating the patterns, and was restricted in this experiment to *substitutions* of phonemes with similar, or *nearest*, phonemes (and thus ignoring two other kinds of errors made by phoneme recognizers, insertions and deletions).

To compute the *nearest phonemes* of a particular phoneme, we used the Modified Value Difference Metric (MVDM) (Cost and Salzberg, 1993). This metric can be used to compute for each pair of values of a particular feature a specific distance, based on the similarity of their co-occurrence distributions with the different possible output classes. By computing this for the focus phoneme feature in our dataset, we can easily derive the three phonemes that are most "confusable" with each phoneme when doing phoneme-to-grapheme conversion. Table 1 presents a list containing the three nearest phonemes for some of the phonemes occurring in the CELEX data. With this method of noise generation, we simulate the type of confusion between sounds that a phoneme recognizer may make.

2.4 Experimental Results and Error Analysis

We used IB1-IG (k = 1, 3, and 5) and IGTREE with levels of generated noise ranging from 0% to 50% (in steps of 5%). The performance of TIMBL for each parameter setting

Phoneme	Nearest phonemes and their MVDM value		
a	‘ - 0.01008	A - 0.0171	D - 1.13
@	e - 0.1096	E - 0.1103	] - 0.1104
d	t - 1.631	T - 1.964	# - 1.964
n	1 - 0.7406	N - 1.331	m - 1.937
k	X - 0.7202	g - 0.9972	# - 1.516
E	e - 0.03827	] - 0.07266	@ - 0.1103
e	] - 0.03558	E - 0.03827	@ - 0.1096
A	a - 0.0171	‘ - 0.0271	D - 1.13
m	n - 1.937	N - 1.937	1 - 1.937
i	J - 0.09213	I - 0.1024	[- 0.3411
p	b - 1.959	P - 1.974	a - 2
h	S - 0.8784	x - 1.299	G - 1.977
I	J - 0.01285	i - 0.1024	[- 0.4156
f	v - 1.617	w - 1.998	x - 1.999
o	u - 0.01982	O - 0.02772	W - 0.02906

Table 1: Some phonemes from CELEX, with their most confusable phonemes

and at every level of noise was obtained by doing *ten-fold cross-validation* (10 CV): the dataset was partitioned into ten pieces and every part was used as a test set, with the remaining 9 parts serving as training set, after which averages over the ten test sets were computed. For each experiment, generalization accuracy on unseen data was kept both at the grapheme level (number of graphemes correctly predicted), and at the word level (number of words for which all graphemes were correctly predicted).

The results of the experiments are presented in Tables 2 and 3: they show TIMBL's accuracy (the number of correct classifications divided by the total number of classifications) at word level (Table 2) and at grapheme level (Table 3). These results are ordered according to the amount of noise in the data and the TIMBL algorithm and parameters used. The percentages in the Tables are averages of the ten results obtained by doing 10 CV. No remarkable deviations from these averages were detected during the 10 CV runs (standard deviations are presented between parentheses in Tables 2 and 3).

The best performing TIMBL algorithm overall is IB1-IG with $k = 1$, though the difference with IB1-IG with $k = 3$ is not very large. When there is no noise, IGTREE (the more efficient version of IB1-IG) comes close, but it is much more sensitive to noisy input. We see that with reliable input, phoneme-to-grapheme conversion is almost an easy task (99% graphemes and 91% words correct).

The errors in the noise-less case are mainly due to ambiguous phonemes. There are three types of ambiguous phonemes. The most frequent type contains phonemes that have different possible spelling forms, and the spelling form belonging to a particular word is conventional (there are no contextual cues which can decide on the spelling form needed). Some examples of this type are listed in the Table below.

AMBIGUOUS PHONEME	PHONEMIC REPRESENTATION	TIMBL'S PREDICTION	CORRECT CONVERSION
/k/ can be *k* or *c*	/INkledIN/ /vudbAlkOmp@titsi/	incleding voetbalkompetitie	inkleding voetbalcompetitie
/s/ can be *s* or *c*	/bKb@lsitat@/ /sEnzatsiblad@/	bijbelsitaten censatiebladen	bijbelcitaten sensatiebladen
/i/ can be *i* or *y*	/elEktrolitis/ /fil@/	elektrolitisch fyle	elektrolytisch file
/@/ can be *e* or *i*	/zikt@G@drAx/ /orlOxskris@s/	ziektigedrag oorlogscrises	ziektegedrag oorlogscrisis
/M/ can be *au* or *ou*	/zefMna/ /triplEkshMt/	zeefouna triplexhaut	zeefauna triplexhout
/K/ can be *ei* or *ij*	/zenuwlKd@r/ /LtwKt/	zenuwleider uitwijdt	zenuwlijder uitweidt

Words in which assimilation processes are at work may introduce ambiguity in phonemes which are otherwise not ambiguous. The phoneme /m/, for example, usually has to be converted to the grapheme *m*, but when a /b/ follows, it is possible (though not necessary) to convert it to *n*. In some cases, there is not enough contextual evidence for TIMBL to decide on one of the two alternatives. Below are some examples of this second type of ambiguous phonemes.

KIND OF ASSIMILATION	PHONEMIC REPRESENTATION	TIMBL'S PREDICTION	CORRECT CONVERSION
/n/ → /m/ before /b/	/embanswEx/	eembaansweg	eenbaansweg
/k/ → /g/ before /b/	st}gbrek@	stugbreken	stukbreken
/d/ → /t/ at word-end	/rotatsitKt/	rotatietijt	rotatietijd
/b/ → /p/ at word-end	/lOp/	lop	lob
/z/ → /s/ after /x/	/dAxs}st@r/	dagsuster	dagzuster
/v/ → /f/ after /k/	/prAktKkfAk@/	praktijkfakken	praktijkvakken

Noise	Parameter settings of TIMBL			
	IGTREE k = /	IB1-IG k = 1	IB1-IG k = 3	IB1-IG k = 5
0	91.2 (0.21)	**91.4** (0.18)	90.2 (0.22)	89.7 (0.23)
5	79.2 (0.44)	**81.3** (0.33)	79.3 (0.38)	77.7 (0.32)
10	70.9 (0.35)	**73.7** (0.30)	71.5 (0.28)	69.5 (0.24)
15	63.0 (0.48)	**66.3** (0.39)	64.2 (0.37)	61.9 (0.30)
20	56.2 (0.46)	**59.7** (0.41)	58.0 (0.46)	55.7 (0.43)
25	49.7 (0.26)	**53.0** (0.29)	51.8 (0.30)	49.7 (0.42)
30	44.3 (0.18)	**47.7** (0.19)	46.6 (0.29)	44.5 (0.39)
35	39.7 (0.21)	**42.8** (0.34)	41.7 (0.34)	39.7 (0.37)
40	34.9 (0.30)	**37.9** (0.33)	37.1 (0.33)	35.4 (0.34)
45	31.4 (0.32)	**33.9** (0.20)	33.5 (0.28)	31.8 (0.35)
50	28.2 (0.38)	**30.7** (0.25)	30.0 (0.42)	28.5 (0.45)

Table 2: Artificial noise experiments - accuracy at word level (percentages)

Noise	Parameter settings of TIMBL			
	IGTREE k = /	IB1-IG k = 1	IB1-IG k = 3	IB1-IG k = 5
0	99.0 (0.03)	**99.1** (0.02)	98.9 (0.03)	98.9 (0.03)
5	97.4 (0.06)	**97.7** (0.05)	97.5 (0.05)	97.2 (0.04)
10	96.2 (0.05)	**96.6** (0.05)	96.3 (0.06)	96.0 (0.05)
15	94.9 (0.07)	**95.5** (0.05)	95.1 (0.06)	94.7 (0.06)
20	93.7 (0.07)	**94.3** (0.07)	94.0 (0.08)	93.5 (0.09)
25	92.5 (0.05)	**93.1** (0.06)	92.9 (0.05)	92.4 (0.06)
30	91.3 (0.04)	**92.0** (0.05)	91.7 (0.07)	91.2 (0.08)
35	90.2 (0.06)	**90.9** (0.08)	90.6 (0.08)	90.1 (0.09)
40	89.0 (0.06)	**89.7** (0.06)	89.6 (0.07)	89.0 (0.06)
45	88.0 (0.06)	**88.7** (0.09)	88.6 (0.07)	88.0 (0.06)
50	87.1 (0.06)	**87.8** (0.08)	87.7 (0.08)	87.1 (0.10)

Table 3: Artificial noise experiments - accuracy at grapheme level (percentages)

Words (or parts of words) with the same pronunciation but a different spelling, can also result in incorrect predictions. TIMBL predicts *ladikant* as the spelling for the phoneme string /ledikAnt/ (*ledikant*), because CELEX contains a lot of words beginning with *lady−* (*ladyshave*, *ladykiller*, *ladylike*, ...). The same goes for the string /lid@rs/ (*lieders*), which TIMBL converted to *leaders*, because *lieder*(*s*) and *leader*(*s*) are pronounced in the same way (/lid@r(s)/). Errors of this kind are not very frequent, though.

Some ambiguous words or phonemes can never be classified correctly by including only previous and following phonemes in the context: TIMBL needs morphogical or syntactic cues to resolve the ambiguity. A typical example is the Dutch verb *worden* (to become), which is pronounced /wort/ in the first, second and third person singular (present tense) but is spelled differently: *word* in the first person, and *wordt* in the second and third person. Without morphogical or syntactic cues, TIMBL can never find the correct spelling. Examples from TIMBL's output are listed below.

PHONEMIC REPRESENTATION	TIMBL'S PREDICTION	CORRECT CONVERSION
/b@spit/	bespied	bespiedt
/dodblut/	doodbloed	doodbloedt
/Ond@rsxKt/	onderscheidt	onderscheid
/Ond@rhMt/	onderhoudt	onderhoud
/Afrat/	afraat	afraadt

TIMBL's incorrect predictions are not always due to ambiguity: errors also occur in words which are spelled in a way that is not typically Dutch (mainly because these words come from another language and were added to the Dutch vocabulary without adapting it to Dutch spelling conventions), like the following:

PHONEMIC REPRESENTATION	TIMBL'S PREDICTION	CORRECT CONVERSION
/rokAj@/	rokuille	rocaille
/sikorK/	sykcurij	cichorei
/pep@rklIps/	peperclips	paperclips
/tAjkwOndo/	tiekwondo	taekwondo
/kyrasM/	curasau	curacao
/projEktims/	projectiems	projectteams
/fwAje/	foyee	foyer
/matine/	matiner	matinee
/bazuka/	bazoeka	bazooka
/x@krust/	gekroest	gecruist

A typical speech recognizer has an error rate of 25%. Our results indicate that phoneme-to-grapheme conversion with data containing 25% to 30% noise can be carried out with 92% to 93% accuracy at grapheme level and 48% to 53% accuracy at word level. This presupposes of course that the errors of the phoneme recognizer indeed correspond to our method of artificial noise introduction, which is unlikely. Phoneme recognizer errors will be context-dependent, which is not the case for the artificial noise. This context-dependency may make the task easier. On the other hand, the artificial noise only contains substitution errors, where the data from the phoneme recognizer will also contain insertion and deletion errors, making the task harder. In the following section, we will therefore investigate the phoneme-to-grapheme conversion problem with training data from a real phoneme recognizer.

TYPE OF MATCH	COST
The phoneme is an exact match for the grapheme, e.g. /o/ exactly matches the *compound grapheme* O (which represents *oo*).	0
The phoneme is not an exact match for the grapheme, but is one of the "confusable" phonemes (see Table 1) of the exactly matching phoneme, e.g. /u/ may be confused with /o/, and can therefore be aligned with O.	0.5
The phoneme is not an exact match, or one of the *confusable* phonemes, for the grapheme, but the phoneme and grapheme are both vowels (or consonants), e.g. /o/ and a.	1
A deletion (= a grapheme is aligned with '-') or an insertion (= a phoneme is aligned with '-').	2
The grapheme is a vowel (consonant) and the phoneme is a consonant (vowel).	4

Table 4: The costs, as used in our DP algorithm, for a grapheme-phoneme pair in an alignment

3 Experiments with a Phoneme Recognizer

The dataset used for our experiments with real data, consisted of a Dutch text of 129075 words (resulting in 605955 graphemes). This text is a transcription of a recording, part of the Corpus Gesproken Nederlands (*Spoken Dutch Corpus*)[2], of a person reading aloud a story. The ESAT phoneme recognizer used this recording to produce a phonemic representation of the text. Before reporting the results and giving a detailed error analysis, we will describe how we prepared this corpus for our experiments.

3.1 Data Preprocessing

As in the artificial noise experiments, grapheme strings which were shorter than their corresponding phoneme string had to be *aligned* by inserting the null symbol '-', and grapheme strings which were longer had to be shortened by using *compound graphemes*.

However, due to the frequent occurrence of deletions in the phoneme strings in this corpus, we were not able to use the EM algorithm (see section 2.2): the implementation we used, requires that every grapheme string is shorter than (or at most as long as) its phoneme string. Though we shortened the grapheme strings by using *compound phonemes*, there was still a considerable number of longer grapheme strings due to deletions in the phoneme strings. For example, the word *alsof* was transcribed by the ESAT phoneme recognizer as /AsAf/, with the phoneme /l/ deleted. Introducing a *compound grapheme* in this word is not possible - the only way to align the word with its phoneme string is by inserting a null symbol in the phoneme string, resulting in /A-sAf/.

An algorithm that handles deletions effectively, is the *Dynamic Programming* (DP) algorithm (also known as *Dynamic Time Warping*) (Wagner and Fischer, 1974; Kondrak, 1999; Kondrak, 2000). Given two strings to be aligned, this algorithm computes an alignment cost for each pair of symbols in the two strings, and stores this cost in a matrix. When the cost for each pair is computed, the algorithm searches for the least expensive way through the matrix. In our implementation of the algorithm, we used the costs listed in Table 4.

Running our DP algorithm on, for example, the grapheme string *dan een zilveren schijnsel* and its phoneme string /@n z@lov@r@n sxKntso/, results in the following alignment (E, X and % are *compound graphemes*, representing respectively *ee*, *ch* and *ij*):

[2] Sponsored by the Dutch NWO and the Flemish IWT, see `http://lands.let.kun.nl/cgn/ehome.htm`.

E	n		z	i	l	-	v	e	r	e	n		s	X	%	n	-	s	e	l
@	n		z	@	l	o	v	@	r	@	n		s	x	K	n	t	s	o	-

However, phoneme strings with null symbols (like /sxKntso-/ in the example above) are not very useful as training material: we will not know for previously unseen data where there are deletions. For this reason, we decided to run our experiments with a training set from which all phoneme strings with deletions, and their corresponding words, were removed. To check whether deleting these words does not lead to a decrease in performance, another experiment was set up with a training set from which only the graphemes which were aligned with the null symbol '-' were deleted. The phoneme string /sxKntso-/, for example, did not appear in the training set of the first experimental set-up, whereas in the second experimental set-up, this string resulted in training instances for all phonemes except the null symbol.

With these two different training material set-ups, we conducted several experiments, the results of which and a detailed error analysis will be reported in the next section. In all these experiments, the test material was the same (no words with deletions were removed from the test data).

3.2 Experimental Results and Error Analysis

With both deleted words and deleted instances, we did two main experiments (resulting in a total of four set-ups): to find out whether information about the spelling of the previous or following word leads to more accurate predictions, we ran experiments in which we included this information in the instances, and other experiments in which this information was not included. Spelling was included by replacing all but one word boundary symbols ('=') in the instances with the last and/or second last, or first and/or second, grapheme(s) of the previous or next word. The graphemes in the instances were preceded by '$' to set them apart from the phonemes. Including spelling context in, for example, the first two instances of /sxKntso-/, results in the following two instances:

Left context			Focus	Right context			Class
$e	$n	=	s	x	K	n	s
$n	=	s	x	K	n	t	X

In each of these four set-ups, we ran TIMBL with 10 CV on the dataset, with the same parameter settings as in the experiments with the CELEX data (IB1-IG with $k = 1$, 3 and 5 and IGTREE). The performance of TIMBL at word and at grapheme level in each experiment and for each parameter setting, is shown in Table 5 (word level) and Table 6 (grapheme level). The percentages in these Tables are averages of the ten results of one 10 CV experiment. We encountered no striking deviations from these averages (standard deviations are listed between parentheses in Tables 5 and 6).

The best scoring algorithm on real data is not the default TIMBL algorithm: in all experiments, using TIMBL with IB1-IG and $k = 3$ gives the best results at word level,

Experiment	Parameter settings of TIMBL			
	IGTREE k = /	IB1-IG k = 1	IB1-IG k = 3	IB1-IG k = 5
+ spelling & no words with deletions	43.5 (1.8)	43.9 (2.0)	**44.8** (2.0)	44.3 (1.8)
- spelling & no words with deletions	46.8 (2.0)	46.9 (2.0)	**47.0** (2.0)	**47.0** (2.0)
+ spelling & no instances with deletions	42.3 (1.8)	42.5 (1.9)	**43.9** (1.9)	43.4 (1.9)
- spelling & no instances with deletions	46.3 (2.0)	46.4 (2.0)	**46.5** (2.0)	**46.5** (2.0)

Table 5: Real data experiments - accuracy at word level (percentages)

DATASET	Parameter settings of TIMBL			
	IGTREE k = /	IB1-IG k = 1	IB1-IG k = 3	IB1-IG k = 5
+ spelling & no words with deletions	72.5 (1.3)	72.2 (1.3)	74.8 (1.3)	**75.3** (1.2)
- spelling & no words with deletions	73.3 (1.3)	73.1 (1.4)	74.5 (1.3)	**74.8** (1.3)
+ spelling & no instances with deletions	74.9 (1.3)	74.6 (1.3)	76.9 (1.3)	**77.0** (1.3)
- spelling & no instances with deletions	76.4 (1.4)	76.2 (1.4)	77.3 (1.4)	**77.4** (1.4)

Table 6: Real data experiments - accuracy at grapheme level (percentages)

although the difference with IB1-IG with $k = 5$ is not very large. At grapheme level, the best scoring algorithm is IB1-IG with $k = 5$. At word level, the results come close to what we predicted with our artificial noise experiments: our expectation was a score of 48-53%, and our best result is 47%. At grapheme level, however, even the best result lies far below our expectations: our artificial noise experiments predicted a score of 92-93%, but our best result is only 77%.

The results in Tables 5 and 6 indicate that leaving out words with deletions entirely, instead of only deleting the graphemes aligned with deletions, does not lead to a decrease in performance: when the instances do not contain spelling from the previous or next word, the best results at word level are obtained with training material from which words with deletions were removed.

Including the spelling of the previous and next word, however, does not result in better performance: the results of the experiments with training material in which spelling is included, are considerably lower, certainly at word level, than the results of the experiments with a dataset without spelling cues.

Tables 5 and 6 give a general idea of TIMBL's performance: to find out how good (or bad) TIMBL predicts a particular grapheme, we computed precision (how many items found for a class are correct), recall (how many items that should have been found for a class, are actually found) and F_β scores (Equation 1) for each class in the dataset, i.e. graphemes, *compound graphemes* and the null symbol.

$$F_\beta = \frac{2 \times precision \times recall}{precision + recall} \tag{1}$$

Analyzing the output of the experiment with the best results at word level (TIMBL with IB1-IG and $k = 3$ on the dataset without words with deletions and without spelling cues), we found that the average F_β score of TIMBL's prediction is 53.93, with a large standard deviation (SD) of 24.80. Table 7 lists the classes which are (1) below the mean score minus one SD, (2) between the mean score minus one SD and the mean score itself, (3) between the mean score and the mean score plus one SD, or (4) above the mean score plus one SD. For some classes, the F_β score is undefined, which means that either all predictions of that class were wrong, or that the class was never predicted. The classes for which the F_β score is undefined or lower than 29.13, do not have much influence on the overall performance of TIMBL, because they are very infrequent in the dataset (classes with an undefined F_β score occur less than 30 times, and classes with $F_\beta \prec 29.13$ occur less than 300 times).

Table 7 makes clear that the *compound graphemes* are a major source of errors: for nearly all of them, the F_β score is below average. It is, however, not possible to abandon the concept of *compound graphemes* as was explained earlier.

To conclude this section, we give an example of TIMBL's best output in Figure 2: the first column contains TIMBL's output, the second column is the correct version for that part of the dataset. Obviously, not every word in the example in Figure 2 is an OOV: when

F_β = UNDEFINED	q	bp	zz	fv	zs					
$F_\beta \prec 29.13$	pb	rr	th	y						
$29.13 \prec F_\beta \prec 53.93$	au	bb	c	dd	dt	eu	eau	ff	gg	kk
	mm	nn	ou	pp	sz	tt	td	u	x	-
$53.93 \prec F_\beta \prec 78.73$	a	aa	b	d	f	g	h	ee	ei	i
	ie	ij	j	k	l	ll	m	ng	o	oo
	oe	p	ss	t	uu	ui	w			
$78.73 \prec F_\beta$	ch	e	n	r	s	v	z			

Table 7: F_β scores

TIMBL's output	correct version
cafe zag en wild zwaaien	cathy zag hen wild zwaaien
haar vader stak zijn dan omhoog	haar vader stak zijn duim omhoog
fassaf hij wilde zeggen	alsof hij wilde zeggen
het komt we goed jog	het komt wel goed joch
haar maar klefde bijnae tegen ik aultoeraanpiee aan	haar moeder kleefde tegen het autoraampje aan

Figure 2: An example of TIMBL's best output

using a large vocabulary speech recognizer in combination with a phoneme-to-grapheme converter, the only words that are of interest are the OOVs. It is therefore interesting to look at the performance of TIMBL on these OOVs only, which is what the following section will deal with.

3.3 Analyzing the out-of-vocabulary items

In our dataset, 8913 words were marked as OOVs. For each experimental set-up and for each parameter setting, we picked the OOVs out of TIMBL's output, and computed accuracy scores at word (Table 8) and at grapheme level (Table 9).

Here, the best scoring algorithm at word level is IB1-IG with $k = 5$. TIMBL's performance, however, is much lower on the OOVs than on the complete dataset: performance at grapheme level decreases with $\sim$15%, while performance at word level takes an enormous drop of $\sim$40%.

One of the reasons for the poor performance at word-level, are the deletions in the phoneme strings: the collection of OOVs contains 3985 (44.7%) words recognized with deletions. Because of the one-to-one correspondence between phoneme and grapheme strings in our dataset, TIMBL can never convert a phoneme string with deletions to a completely correct word: even if TIMBL were able to convert all recognized phonemes

Experiment	Parameter settings of TIMBL			
	IGTREE k = /	IB1-IG k = 1	IB-IG k = 3	IB-IG k = 5
+ spelling & no words with deletions	5.4	5.1	6.3	**6.5**
- spelling & no words with deletions	6.1	6.2	6.7	**6.9**
+ spelling & no instances with deletions	5.1	5.0	6.2	**6.4**
- spelling & no instances with deletions	6.1	6.2	6.7	**6.9**

Table 8: Real data experiments - accuracy at word level for OOVs (percentages)

DATASET	Parameter settings of TIMBL			
	IGTREE k = /	IB1-IG k = 1	IB-IG k = 3	IB-IG k = 5
+ spelling & no words with deletions	59.1	58.1	62.3	**63.0**
- spelling & no words with deletions	59.7	58.9	62.0	**62.7**
+ spelling & no instances with deletions	60.1	58.9	63.1	**63.7**
- spelling & no instances with deletions	60.1	59.9	62.8	**63.3**

Table 9: Real data experiments - accuracy at grapheme level for OOVs (percentages)

correctly, the performance at word-level could still not be higher than ∼55%.

The low accuracy at word level does not mean that TIMBL's output is useless, however. The underlined words in the phrases in Figure 3 are examples of OOVs converted by TIMBL, using IB1-IG with $k = 5$. Though only one of these five words is transcribed correctly, it would not take much effort for readers to figure out what the correct version of the misspelled words should be. Obviously, this does not count for the misspelled word *belleen* in Figure 3, which hardly resembles its correct version, *bmw*.

The errors made by TIMBL are not equally distributed over the OOVs: Figure 4 compares the expected number of errors in a word of length n (= wordlength × TIMBL's average percentage of errors at grapheme level, i.e. 38.3% (100 - 61.7%)) with the observed average number of errors in the words of that length. The bars in Figure 4 depict the frequency of words with length n. We see that the two curves are not totally equal: short OOVs contain more errors than expected, while long OOVs have fewer errors. Only in words with length 10 to 15, the observed number of errors is more or less equal to the expected number. However, as the bars in Figure 4 illustrate, the shorter words, with a higher than expected number of errors, are more frequent than the longer words.

A possibly useful post-processing step would be to integrate spelling correction in the phoneme-to-grapheme converter. However, a priori it is not very likely that spelling correction will lead to substantial increase of accuracy at word-level: incorrectly converted OOVs containing many errors, and not just one or two, make it difficult for a spelling converter to find good alternatives. Figure 5 shows that such OOVs are quite numerous: in the output files from each experiment, we counted how many OOVs had a certain amount of errors per word, and put the averages in the graph in Figure 5. The most numerous are OOVs with 3 incorrectly predicted graphemes per word (1759 OOVs), and OOVs with 4, 5 and even 6 errors per word are frequent (1510 OOVs). Still, if we can get most of the OOVs with only 1 or 2 mistakes per word (2619 OOVs) right using spelling correction, this would lead to an increase of ∼25% in word level accuracy.

TIMBL'S OUTPUT	CORRECT VERSION
het komt wel goed <u>jog</u>	het komt wel goed <u>joch</u>
bijna tegen het <u>aultoeraampiee</u> aan	bijna tegen het <u>autoraampje</u> aan
haar neus werd <u>pladvedrukt</u>	haar neus werd <u>platgedrukt</u>
cathy zag de <u>belleen</u> langzaam verdwijnen	cathy zag de <u>bmw</u> langzaam verdwijnen
cathy staarde hem <u>bevreemd</u> aan	cathy staarde hem <u>bevreemd</u> aan

Figure 3: An example of TIMBL's best output - only OOVs

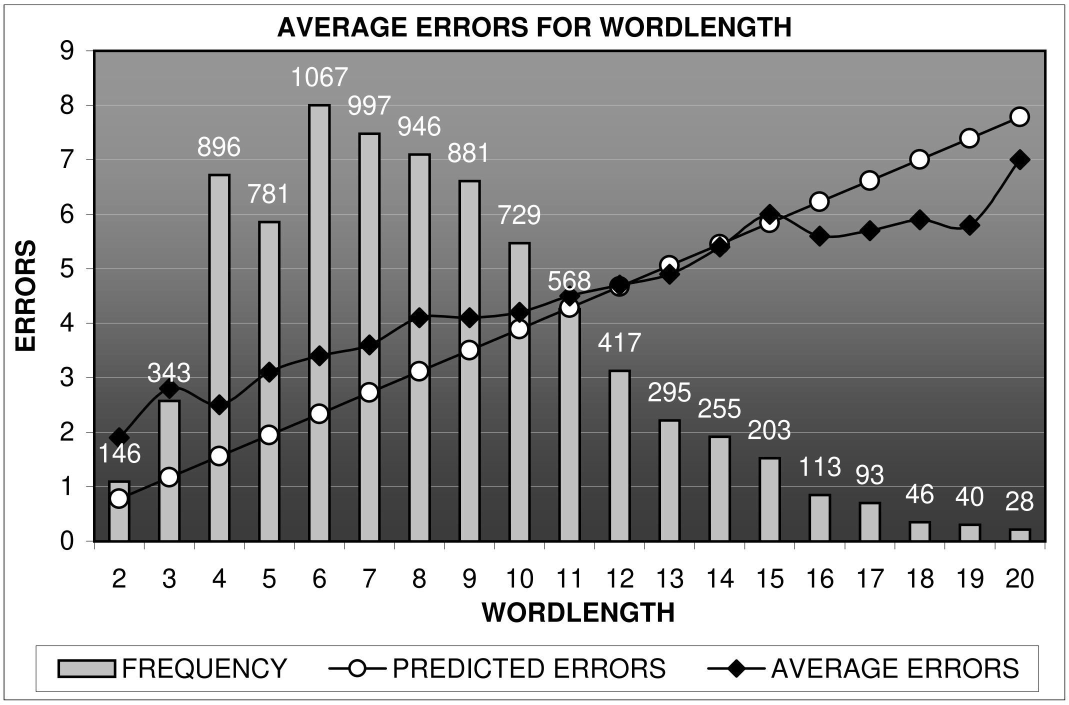

Figure 4: Real data experiments - average errors per word for wordlength

4 Conclusion

In this report we investigated the feasibility of phoneme-to-grapheme conversion for out-of-vocabulary items in speech recognition, and the ability of machine learning methods for this task to adapt to the errors produced by the phoneme recognizer. We have discussed the results of experiments in which TIMBL, a memory-based learner was used for this task. We saw that it can carry out this task almost perfectly with a clean dataset, i.e. presupposing perfect phoneme recognition. In that case, 91% correctly transcribed words is feasible, with errors mainly related to the conventional lexical aspects of Dutch spelling. Adding 30% artificial noise to that dataset by substituting phonemes with similar phonemes, results in a drop to 48% accuracy at word level.

Using a dataset with phoneme strings generated by a phoneme recognizer that also contains more or less 30% noise, we achieved a similar result at word-level on the entire dataset. However, performance on the OOVs in this dataset, in which we are especially interested, is only 7% at word level (about 60% of the graphemes correctly predicted). Although at a very low level, this accuracy may still be useful because many of the orthographically transcribed words can be recognized easily. An important problem is that the real phoneme recognizer introduces a lot of deletions of phonemes, which is a situation impossible to handle in the current architecture of the system.

Our future work on phoneme-to-grapheme conversion for OOVs will concentrate on: (1) the optimalisation of TIMBL on the task (we will exhaustively optimise parameter settings and information sources for the task), (2) trying to solve the problem with deletions in the phoneme strings (possibilities here are to tune the recognizer towards fewer deletions, or to adapt the learning regime toward taking into account possible deletions), (3) using spelling correction with a large vocabulary as a post-processing module, and (4) evaluating the phoneme-to-grapheme converter in combination with *confidence measures*

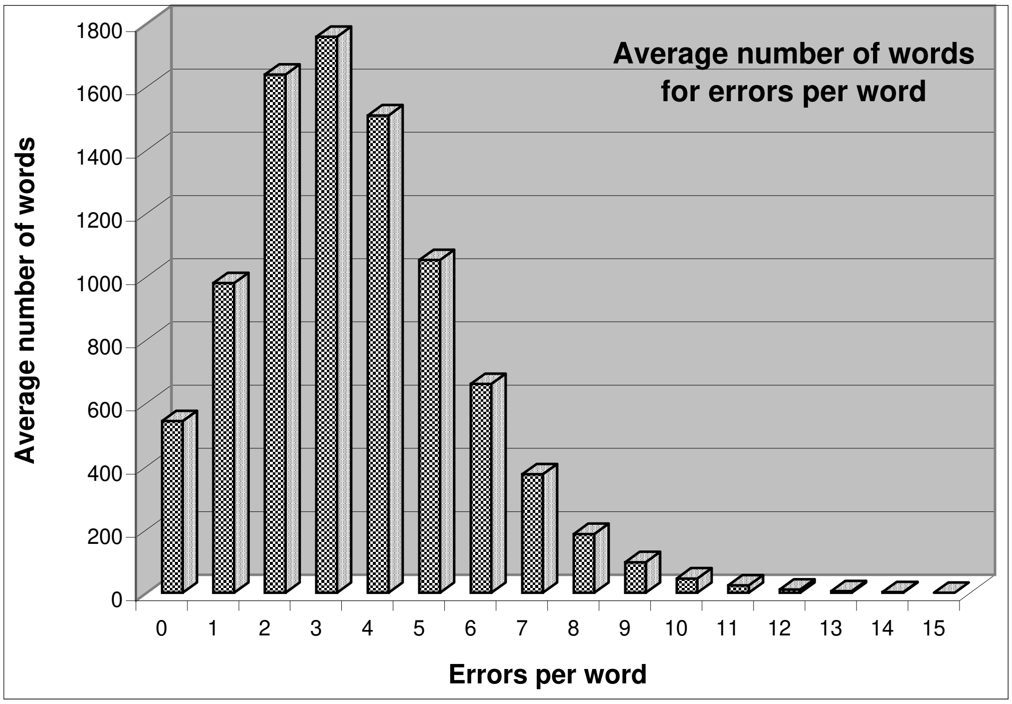

Figure 5: Real data experiments - number of words for errors per word

for OOVs developed by ESAT. At the level of error analysis we will also try to see in how far the learning method actually succeeds in adapting to the systematicities of the errors of the phoneme recognizer.

5 References

Baayen, R. H., R. Piepenbrock, and H. van Rijn. 1993. *The* CELEX *lexical data base on* CD-ROM. Linguistic Data Consortium, Philadelphia, PA.

Busser, B., W. Daelemans, and A. van den Bosch. 1999. Machine learning of word pronunciation: the case against abstraction. In *Proceedings of the Sixth European Conference on Speech Communication and Technology, Eurospeech99*, pages 2123–2126, Budapest, Hungary, September.

Cost, S. and S. Salzberg. 1993. A weighted nearest neighbour algorithm for learning with symbolic features. *Machine Learning*, 10:57–78.

Daelemans, W. and A. van den Bosch. 1992. A neural network for hyphenation. In I. Aleksander and J. Taylor, editors, *Artificial Neural Networks 2: proceedings of the International Conference on Artificial Neural Networks*, pages 1647–1650, Amsterdam. Elsevier.

Daelemans, W. and A. van den Bosch. 1996. Language-independent data-oriented grapheme-to-phoneme conversion. In J. P. H. Van Santen, R. W. Sproat, J. P. Olive, and J. Hirschberg, editors, *Progress in Speech Processing*. Springer-Verlag, Berlin, pages 77–89.

Daelemans, W., A. van den Bosch, and A. Weijters. 1997. IGTree: using trees for compression and classification in lazy learning algorithms. *Artificial Intelligence Review*, 11:407–423.

Daelemans, W., A. van den Bosch, and J. Zavrel. 1999. Forgetting exceptions is harmful in language learning. *Machine Learning, Special issue on Natural Language Learning*, 34:11–41.

Daelemans, W., J. Zavrel, K. van der Sloot, and A. van den Bosch. 2000. TiMBL: Tilburg memory based learner, version 3.0, reference guide. ILK Technical Report 00-01, Tilburg University. Available from `http://ilk.kub.nl`.

Demuynck, K. 2001. *Extracting, modelling and combining information in speech recognition*. Ph.D. thesis, K.U.Leuven, ESAT, February.

Demuynck, K., J. Duchateau, D. Van Compernolle, and P. Wambacq. 1998. Fast and accurate acoustic modelling with semi-continuous HMM. *Speech Communication*, 24(1):5–17, April.

Demuynck, K., J. Duchateau, D. Van Compernolle, and P. Wambacq. 2000. An efficient search space representation for large vocabulary continuous speech recognition. *Speech Communication*, 30(1):37–53, January.

Duchateau, J. 1998. *HMM based acoustic modelling in large vocabulary speech recognition*. Ph.D. thesis, K.U.Leuven, ESAT, November.

Hoste, V., W. Daelemans, E. Tjong Kim Sang, and S. Gillis. 2000. Meta-learning for phonemic annotation of corpora. In A. van den Bosch and H. Wiegaard, editors, *Proceedings of the twelfth Belgium-Netherlands artificial intelligence conference (BNAIC'00)*, pages 331–332. Extended abstract of ICML 2000 paper.

Hoste, V., S. Gillis, and W. Daelemans. 2000. Machine learning for modeling dutch pronunciation variation. In P. Monachesi, editor, *Computational Linguistics in the Netherlands 1999. Selected papers from the tenth CLIN meeting*, pages 73–83.

Kondrak, G. 1999. Alignment of phonetic sequences. Technical Report CRSG-402, University of Toronto. Available from `http://www.cs.toronto.edu/~kondrak`.

Kondrak, G. 2000. A new algorithm for the alignment of phonetic sequences. In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 288–295, Seattle. NAACL.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.

van den Bosch, A. and W. Daelemans. 1993. Data-oriented methods for grapheme-to-phoneme conversion. In *Proceedings of the 6th Conference of the EACL*, pages 45–53.

van den Bosch, A. and W. Daelemans. 1994, revised in 1999. ILKALIGN. Software tool for automatic alignment, developed at the University of Tilburg.

van den Bosch, A. and W. Daelemans. 1998. Do not forget: Full memory in memory-based learning of word pronunciation. In D.M.W. Powers, editor, *Proceedings of NeMLaP3/CoNLL98*, pages 195–204, Sydney, Australia.

Wagner, R. A. and M. J. Fischer. 1974. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173.