Creating Small Speech Recognizers Quickly

Björn Bringert

January, 2005

Abstract

Creating a speech recognizer from scratch can be quite daunting for the beginner. We present a simple and economical method for creating speech recognizers. Our goal is a favorable cost/benefit ratio rather than the best possible recognition score. We are mainly interested in recognizers with restricted domains, for example for use as input to a dialog system. We have created a number of tools which make the process easier, and use a running example to illustrate how it is done. Our starting point is a Grammatical Framework grammar, and we use HTK and ATK to create the speech recognizer.

1 Introduction

The aim of this paper is to lower the barrier to getting started with creating speech recognizers. While it can hardly be called an introduction to the area, the hope is that the reader can use this material to overcome the first hurdles.

The paper is written in a tutorial style, and each section is accompanied by the exact commands used for that step, to make it easy for the interested reader to repeat the experiments. The reader is assumed to be familiar with some basic speech technology terminology and simple Unix shell scripting. After reading this paper, it might be appropriate to continue with the HTK book [16] or the SphinxTrain manual [12].

In order to use speech recognition in an application, a number of components are needed. Typically, this includes:

- An acoustic model.
- A pronunciation model.
- A recognition grammar or statistical language model.

The following sections outline simple methods for creating the different components, and introduce some tools which we have created to simplify the steps needed.

Once we have a speech recognizer capable of producing text from speech, the application needs to interpret it. In this paper, we assume that this is done by parsing using a grammar. Other possible methods include phrase spotting and statistical parsing. Furthermore, we assume that the parsing grammar already exists and is written in Grammatical Framework, though the general approach taken here should be applicable to other grammar formalisms as well.

This paper contains a running example: the creation of a speech recognizer for a grammar for simple Swedish sentences using a limited vocabulary. The example in the paper uses a Grammatical Framework (GF) [10] grammar to parse the input, and the Hidden Markov Model Toolkit (HTK) [16] and the Application Toolkit for HTK (ATK) [15] are used to build the speech recognizer. Because of resource constraints, we create a speaker-dependent recognizer, using recordings of only one subject, the humble author.

The HTK-specific parts of this paper are largely based on the tutorial chapter of the HTK book [16]. The tools and the collected data for the example are available from: `http://www.cs.chalmers.se/~bringert/darcs/atkswe/`.

2 A Quick Introduction to HTK and ATK

The Hidden Markov Model Toolkit (HTK) [16] is a general toolkit for Hidden Markov Models (HMMs). It is mainly geared towards speech recognition, but can be used for other tasks as well. HTK includes a large number of tools for training and manipulating HMMs, working with pronunciation dictionaries, n-gram and finite-state language models, recording and transcribing speech, etc.

We found a small bug in the HTK `HDMan` program. In order to generate a list of triphones from the dictionary, you have to apply the patch which is available in the software distribution accompanying this paper.

The Application Toolkit for HTK (ATK) is a C++ library for creating experimental speech recognition systems using HTK. We use HTK for building and evaluating the recognizer, and ATK to use the finished recognizer with other systems.

3 The Example Grammar

As an example, we will build a speech recognizer for the *Swedish Stoneage grammar*, an example grammar included with the GF system. The grammar essentially consists of a Swadesh list [13] (a list of basic words, used in glottochronology to determine the distance between languages), together with some simple function words and basic syntactic constructs. There are Stoneage grammars available for 7 languages in the GF distribution, all sharing a common abstract syntax.

In order to have a single grammar file to work with, we package the grammar and all its dependencies in `stoneage-swe.gfcm`, a single file in UTF-8 encoded GFCM format:

```
echo 'pm -utf8 | wf stoneage-swe.gfcm' | gf StoneageSwe.gf
```

4 Pronunciation Model

The pronunciation model maps speech sounds to words in the language, or vice versa, depending on how it is used. How difficult it is to produce such a model depends on the relationship between the language's orthography and pronunciation.

Pronunciation dictionaries for some languages are already available, for example for British English [11]. If there is no existing dictionary for the language, we can either write

it by hand, generate it using pronunciation rules for the language, or use a combination of the two approaches.

The first step is to decide on a set of phone transcription symbols. For our example we use a version of the Speech Assessment Methods Phonetic Alphabet (SAMPA) system for Swedish [14], slightly modified in order to use symbols more compatible with HTK.

For generating a Swedish pronunciation dictionary, we have modified Markus Forsberg's Swedish transcription software [5], which produces IPA transcriptions of Swedish words. This, together with a simple tool which extracts the list of word forms used in a GF grammar, allows the user to automatically create a pronunciation dictionary containing every word form in the grammar:

```
mkwordlist.pl stoneage-swe.gfcm | gendict_swe > stoneage-swe.dict
```

Here is an except from the generated `stoneage-swe.dict` file:

```
blommornas          b l U m U rn a s
blommors            b l U m U rs
blåser              b l o: s e r
bred                b r e: d
```

Since the pronunciation rules for Swedish are quite complex and the transcription program is still in an early stage, the transcriptions are less than perfect. This should be corrected by improving the transcription software. The pronunciations for words unique to the application in question, such as names, can be hand-edited if needed.

5 Acoustic Model

The acoustic model is used to recognize speech sounds given sound data. It is produced by training it on recorded and transcribed speech. Creating the acoustic model appears to be the most time and data consuming of the steps outlined here. Fortunately, a well-trained acoustic model is to a large extent domain-independent.

The steps required for producing an acoustic model include:

- Selecting and recording the training utterances, or collecting training utterances and transcribing them.
- Setting up data parametrization and creating an initial model.
- Training the model using the data.

5.1 Producing Transcribed Speech

There are several ways to produce the transcribed data set use to train the model. The data can be recorded, for example from telephone conversations, and then manually transcribed. This has the benefit that the speech is spontaneous and unrestricted. There are existing such data sets for some common languages, for example the DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus [1]. If such a corpus is available, especially if it is already transcribed, this method should be the most economical. However,

few such corpora are available free of charge, and few are available for languages other than English.

Another way to produce the data is to select or generate a number of utterances, and record subjects reading them. Since we do not have access to any existing corpus of Swedish speech, we will use this method.

Transcribing the data, for both of the methods described above, can be done on the phone or word level. If it is done on the word level, the pronunciation dictionary can be used to translate this to phone level transcriptions. This makes the transcription task easier, and, in the case where the utterances are know beforehand, even eliminates it. If there are several possible pronunciations of a word in the pronunciation dictionary, some method is needed for deciding which one is used in a given recording. The phone level transcriptions might also, depending on how the data is used for training the acoustic model, need to be aligned with the sound data.

5.1.1 Generating Training Utterances from a Grammar

To select some sentences for recording, we could manually construct sentences which cover the phonetic features of the language, to get as broad a coverage as possible over the language. On the other hand, if we have a limited domain, it might be more beneficial to select sentences within that domain to ensure good coverage of the relevant sound combinations. Since we already have a grammar for parsing the speech input, we can use that grammar to generate training utterances.

We have created a tool which uses GF's built-in random generation facility to generate training utterances. Duplicate utterances are eliminated, as these are expected to add little new information.

```
genutts.pl stoneage-swe.gfcm 200 > train_utts.txt
```

This writes 200 numbered utterances to the file `train_utts.txt`, and prints some statistics about the grammar and generated utterances to the terminal.

Here is an excerpt from the generated file:

```
000020: han slår djuren
000021: tre näsor känner lukten av de smutsiga våta naglarna
000022: frun flyter
```

5.1.2 Recording the Utterances

Recording the utterances is simply a matter of giving the subject one utterance at a time and recording the subject reading it. We have created a small tool which takes a file such as the one created in the previous step, and prompts the user to record each one using the `HSLab` tool from the HTK distribution. Each recording is saved to a file with the same name as the label of the utterance.

```
mkdir train_data
cd train_data
HSLab noname &
prompt ../train_utts.txt
```

After the recording is completed, the `train_data` directory will contain one sound file for each utterance.

5.2 Building the Model

Following the HTK tutorial for building an acoustic model once we have recorded the data and produced the pronunciation dictionary requires about 50 steps of various degrees of automation.

We have created a script and some helper programs which automates all of these steps, and creates the model given a GF grammar, a pronunciation dictionary, a list of the recorded utterances, and the recorded speech files.

To build the monophone and triphone models from our grammar, dictionary and training data, we run:

```
mkrec.pl stoneage-swe.gfcm stoneage-swe.dict \
   train_utts.txt train_data
```

The script performs the following steps (quite simplified, see the comments in `mkrec.pl` for more details):

1. Create phone level transcriptions of the training utterances by using the dictionary. The transcriptions use the first available pronunciation for each word. This problem will will be taken care of in a later step.

2. Parametrize the recorded data. We currently use the settings from the HTK tutorial: Mel Frequency Cepstral Coefficients (MFCCs), 10 ms frames, a 26 channel filter bank, and the output is 12 MFCC coefficients.

3. Create simple prototype models, and re-estimate them using the data.

4. Use the current models to for each word select the closest pronunciation from the dictionary, and then re-estimate the models using the new data.

5. *We now have a set of monophone models which can be used for recognition. We will now proceed to build triphone models.*

6. Create triphone models by copying the monophone models.

7. Re-estimate the triphone models, with the transcriptions converted to use triphones.

8. *We now have a set of triphone models which can be used for recognition.*

9. Here the HTK tutorial goes on to create tied-state triphone models, in order to make more robust models. However, this would seem to require manually deciding which triphones to cluster together and has been skipped to keep the process simple.

6 Language model

In order to guide the recognizer, a grammar or statistical language model is used. Speech recognition grammars are normally regular or simple (for example non-left-recursive) context-free grammars, for performance reasons. These grammars are only used for recognition; no parse trees are produced. Speech recognition grammars are sometimes weighted to allow selecting the most likely interpretation of the speech input. An alternative or complement to grammars is statistical language models, such as triword models.

We have added functionality to the GF system for generating finite-state networks in HTK's Standard Lattice Format (SLF). We can generate such a network from our grammar with the following command:

```
echo 'pg -printer=slf | wf stoneage-swe.net' | gf stoneage-swe.gfcm
```

This writes an SLF network to the file `stoneage-swe.net`. The finite automaton is generated in the following steps:

1. A context-free grammar is generated from the GF grammar, as described by Peter Ljunglöf [6].
2. The context-free grammar is approximated by a regular grammar using an algorithm due to Mohri and Nederhof [7].
3. The regular grammar is compiled into a non-deterministic finite-automaton with an algorithm described by Nederhof [8].
4. The non-deterministic finite automaton is minimized by forward and reverse determinization, producing a deterministic finite automaton. This method is due to Brzozowski [3].

7 Running the Recognizer

Once the recognizer has been built, we need to be able to run it and use it from our application. ATK or HTK can be used directly, but in order to provide a simpler interface appropriate for the most common situations we have created a library, *libatkrec*, for running ATK recognizers from C or Haskell. The library is available from `http://www.cs.chalmers.se/~bringert/darcs/atkrec/`.

We have then used the Haskell interface to add a speech input command to the GF system. The command generates a finite state network for the currently active grammar and starts the speech recognizer with it. This is an example GF session using English and Swedish grammars for numerals:

```
> si -lang=english -tr | p -lang=english -tr | l -lang=swedish
three hundred and seventy - four
num (pot2as3 (pot2plus (pot0 n3) (pot1plus n7 (pot0 n4))))
tre hundra sjuttio fyra
```

Grammar	L	TS	TW	CS	Acc	CW	D	S	I
Numerals	1	100	74	0.00	66.2	70.2	4.05	25.7	4.05
Numerals	3	100	74	35.0	73.0	79.7	4.05	16.2	6.76
Stoneage	1	200	92	15.0	59.8	77.2	0.00	22.8	17.4
Stoneage	3	200	92	35.0	70.8	79.4	0.00	20.7	8.70

Table 1: Evaluation results

The `si` command uses an ATK speech recognizer to get one utterance from the user. The flag `-flag=english` makes it generate the recognition network from the loaded English grammar. The `-tr` option, which can be used with most commands, causes the command to print its output to the terminal as well as passing it to the next command in the pipeline. The `p` command parses the output from the speech recognizer, again using the English grammar. Finally, the `l` command linearizes the parse tree using the Swedish grammar. The three lines printed by the commands are the speech recognizer output, the parse tree, and the result of the linearization.

8 Evaluation

In order to evaluate the recognizer, we record some more utterances as described in section 5.1.2. We have created a small tool which uses runs the recognizer on the test utterances, and uses tools from HTK to compare the output to the original utterance text:

```
eval_rec.pl stoneage-swe.dct stoneage-swe.net test_utts.txt \
  test_data hmm_tri/hmmlist hmm_tri/macros hmm_tri/hmmdefs
```

Table 1 lists the evaluation data for four different recognizers. They were all evaluated with 20 utterances from the same grammar from which the training utterances were taken. All utterances were spoken by the same Swedish male. Note that no significance tests have been made, and the number of digits shown in the table is not related to the number of significant digits in the values. The table column headings mean:

Grammar The grammar from which the training and test utterances were taken. "Stoneage" refers to the grammar used in the running example. It contains 837 lexical items. "Numerals" is a grammar for Swedish numerals from 1 to 999999 which contains 40 lexical items. Both grammars are available in the GF distribution.

L The phone string lengths used, 1 for monophones and 3 for triphones.

TS Number of training utterances.

TW Total number of words in the 20 test utterances.

CS Percentage of whole test utterances which were recognized correctly.

Acc The accuracy, calculated as $CW - I$.

CW Percentage of the test words which were recognized correctly.

D Number of deletions as percentage of the number of test words.

S Number of substitutions as percentage of the number of test words.

I Number of insertions as percentage of the number of test words.

The recognition accuracy is quite poor for all of the recognizers, with the best one getting more than one in four words wrong. However, considering the minimal effort involved in building each recognizer (less than 30 minutes, with data collection taking up most of the time once we had all the infrastructure in place), this seems like a good start. We need to investigate the effect of increasing the size of the training sets, and experiment with the settings to see how much this can be improved. It should also be noted that the "Stoneage" results have been achieved using an unedited auto-generated pronunciation dictionary with many errors.

9 Related Work

With most existing speech recognition systems, for example Nuance Recognizer [9], it is easy to get an application-specific speech recognizer for an already supported language by supplying a recognition grammar or network (though this task can be further simplified by generating recognition grammars from grammars in higher-level formalisms as described here and elsewhere [4, 2]).

However, if there is no available acoustic model and pronunciation dictionary these will have to be created. The manuals for HTK [16] and SphinxTrain [12], two of the freely available speech recognition toolkits, list large numbers of steps that need to be taken to create even the most rudimentary acoustic model. We hope that our work can be a small step towards simplifying this process.

10 Future Work

We have made no attempt to tweak the numerous parameters used when training and running the recognizer. Doing so is quite likely to yield better results. It would be interesting to create a tool to automate the build-evaluate-tweak cycle.

The Swedish transcription system which we used needs more work in order to generate high quality transcriptions. From inspection of its current output it seems that vowel quantity is one particular area which needs more work. Since we use the recognizer to select the best pronunciation for each word in the data, the transcription tool could be more liberal in outputting more than one pronunciation for each word.

Since it appears to require manual intervention, we skipped the creation of tied-state triphones in the acoustic model creation. As we are training on very little data, clustering the triphones should make the triphone models more robust. Some (semi-)automatic method for this should be investigated.

It would be interesting to try this method on a larger data set, to see how good a system can be made with this simple method.

No effort is currently made to make the test sentences cover the phonetic variation in the grammar. Thus it is likely that certain sound combinations are underrepresented or

missing in the recorded data. The utterance generator could take phonetic variation into account.

We could use GF's support for weighted grammars to generated weighted finite-state recognition networks. Carrying the weight data though the transformation from a GF grammar to a finite-state network while preserving the meaning of the weights seems to be a non-trivial problem.

The data recording tool is somewhat primitive, and does for example not allow the user to go back and re-record utterances where something went wrong. This tool would have to be improved before any larger scale data collection could be done.

11 Conclusions

We were able to successfully build a speech recognizer in well under thirty minutes once all the tools were created. This could help making speech recognizer prototyping more accessible to users outside the immediate field of speech technology.

The quality of the produced recognizer is still low, but there are several avenues to explore for achieving better results while maintaining the relative ease with which the recognizer can be built.

The following tools and libraries have been created to support this work:

- A tool for generating training utterances from a GF grammar.
- A tool for generating a pronunciation dictionary for a Swedish GF grammar, using Markus Forsberg's automatic Swedish transcription system.
- A GF command for generating finite-state recognition networks from a GF grammars.
- A simple tool for facilitating the recording of prepared utterances.
- A tool for easily creating and training an acoustic model for a simple speech recognizer, given a GF grammar, a pronunciation dictionary, and recorded training utterances with transcriptions.
- A simple tool for evaluating speech recognizers.
- C and Haskell libraries for running simple ATK speech recognizers.
- A GF command for getting input in the current grammar from an ATK speech recognizer.

References

[1] DARPA TIMIT acoustic-phonetic continuous speech corpus. `http://www.ldc.upenn.edu/Catalog/LDC93S1.html`.

[2] BRINGERT, B. Embedded grammars. Master's thesis, Chalmers University of Technology, Gothenburg, Sweden, February 2005.

[3] BRZOZOWSKI, J. A. Canonical regular expressions and minimal state graphs for definite events. In *Mathematical theory of Automata*. Polytechnic Press, Polytechnic Institute of Brooklyn, N.Y., 1962, pp. 529–561. Volume 12 of MRI Symposia Series.

[4] DOWDING, J., HOCKEY, B. A., GAWRON, J. M., AND CULY, C. Practical issues in compiling typed unification grammars for speech recognition. In *Meeting of the Association for Computational Linguistics* (2001), pp. 164–171.

[5] FORSBERG, M. Automatic transcription of swedish. http://www.cs.chalmers.se/~markus/transcription/gtp.cgi.

[6] LJUNGLÖF, P. *Expressivity and Complexity of the Grammatical Framework*. PhD thesis, Göteborg University, Gothenburg, Sweden, November 2004. http://www.cs.chalmers.se/~peb/pubs/p04-PhD-thesis.pdf.

[7] MOHRI, M., AND NEDERHOF, M.-J. Regular approximation of context-free grammars through transformation. In *Robustness in Language and Speech Technology*, J.-C. Junqua and G. van Noord, Eds. Kluwer Academic Publishers, Dordrecht, 2001, pp. 153–163. http://www.coli.uni-sb.de/publikationen/softcopies/Mohri:2001:RAC.pdf.

[8] NEDERHOF, M.-J. Regular approximation of cfls: A grammatical view. In *Advances in Probabilistic and other Parsing Technologies*, H. Bunt and A. Nijholt, Eds. Kluwer Academic Publishers, 2000, pp. 221–241. http://www.coli.uni-sb.de/publikationen/softcopies/Nederhof:2000:RACa.pdf.

[9] NUANCE COMMUNICATIONS, INC. *Nuance Speech Recognition System 8.5: Introduction to the Nuance System*. Menlo Park, CA, USA, December 2003.

[10] RANTA, A. Grammatical Framework, a type-theoretical grammar formalism. *The Journal of Functional Programming 14*, 2 (2004), 145–189. http://www.cs.chalmers.se/~aarne/articles/gf-jfp.ps.gz.

[11] ROBINSON, T. British english example pronunciations dictionary, 1997. http://www.speech.cs.cmu.edu/comp.speech/Section1/Lexical/beep.html.

[12] SINGH, R. SphinxTrain documentation. http://fife.speech.cs.cmu.edu/sphinxman/scriptman1.html.

[13] SWADESH, M. Lexico-statistic dating of prehistoric ethnic contacts. *Proceedings of the American Philosophical Society 96* (1952), 452–463.

[14] WELLS, J. SAMPA for Swedish. http://www.phon.ucl.ac.uk/home/sampa/swedish.htm.

[15] YOUNG, S. *ATK - An Application Toolkit for HTK*, 1.4.1 ed. Machine Intelligence Laboratory, Cambridge University Engineering Dept, Trumpington Street, Cambridge, CB2 1PZ, United Kingdom, July 2004. http://mi.eng.cam.ac.uk/~sjy/ATK_Manual.pdf.

[16] YOUNG, S., EVERMANN, G., GALES, M., HAIN, T., KERSHAW, D., MOORE, G., ODELL, J., OLLASON, D., POVEY, D., VALTCHEV, V., AND WOODLAND, P. *The HTK Book (for HTK Version 3.3)*. Cambridge University Engineering Department, April 2005. http://htk.eng.cam.ac.uk/prot-docs/htkbook.pdf.