Hidden Markov Models for Grapheme to Phoneme Conversion

Paul Taylor

Machine Intelligence Laboratory
Engineering Department, University of Cambridge
`pat40@cam.ac.uk`

Abstract

We propose a method for determining the canonical phonemic transcription of a word from its orthography using hidden Markov models. In the model, phonemes are the hidden states and graphemes the observations. Apart from one pre-processing step, the model is fully automatic. The paper describes the basic HMM framework and enhancements which use preprocessing, context dependent models and a syllable level stress model. In all cases the power of the framework lies in that training of the models (which includes alignment of graphemes and phonemes, training of transitions and training observation probabilities) is performed in a single step.

1. Introduction

This paper tackles the problem of grapheme-to-phoneme (GTP) conversion; that is how to determine the correct or best sequence of phonemes for a word when presented with that words orthography. Before delving into the details we think it is worthwhile examining the general nature of the problem.

Two main justifications are commonly given for the continuing need for a GTP component. Firstly, there will always be genuinely new words ("blairism", "email", "yuppie") created in the course of time. In addition there are many words which may not be new, but were ignored when the system was originally built and have now become common enough to require proper pronunciation (e.g. "bin laden"). In such cases GTP conversion will always be required. Secondly, GTP rules can be used in cases where memory is limited. It almost certainly was the case with early TTS systems that memory was scarce and very large lexicons were not practical. Today however, many TTS systems use hundreds of megabytes of memory in the unit selection system, and by comparison, lexicons containing even millions of words do not make much impact on the overall footprint. Furthermore, it is important to realise that using algorithms to reduce the size of the lexicon comes at an increased cost in processing resource: looking a word up in a lexicon is relatively cheap computationally, whereas most algorithms use considerably more processor resource to produce the phoneme sequence. Hence in a real situation, the decision of what size of lexicon to use is entirely dependent on the footprint/speed requirements of that system, one cannot a priori pick one approach as best.

2. Previous Approaches

Space prevents us from giving a thorough overview, but an excellent review of the several techniques can be found in Damper [1]. Following on from this, we can perhaps identify three main approaches. The first and oldest technique is to use rules which have been written "by hand" (that is, they have been written explicitly by the system developer based on his or her knowledge of the problem). Typically, these rules are in the form of context-sensitive rewrite rules of the form A/X/B → y, meaning that letter X is "rewritten" as phoneme y, when X occurs in the context of letters A and B. A typical rule of this form would be sp/c/e → s, which is read as "letter c rewrites as phoneme /s/ when c occurs at the start of a word and before a letter e".

The second approach we call the "data-driven" approach in which the algorithm is learned automatically from data. The three most successful techniques in this group are the use of decision trees [2], pronunciation by analogy [1] and neural networks (reviewed in [1]). It is interesting to note that the actual operation of many data-driven techniques is no different from the hand-written context sensitive rules; in the case of a decision tree, the questions asked by the tree can often be formulated as context-sensitive rules; and indeed once a decision tree has been created, its operation can be examined. The key difference lies in the way the rules are developed; in one case they are written explicitly; in the other they are learned from data.

The third approach is the statistical approach; this of course is data-driven also, but these techniques not only learn from data, but uses statistical techniques to do so. Chen [4] describes one such technique which makes use of joint n-grams of letter and phoneme sequences in much the same way that one might approach the problem of machine translation between two languages. Jelinek [3] also describe approaches of this type.

3. Hidden Markov Models

Here, we present an approach that uses hidden Markov models (HMMs) as the basis for determining the phoneme sequence from the grapheme sequence. In this HMM formulation, the phonemes are the hidden states, and the transitions between the phonemes describe the probability that one phoneme will follow another. The graphemes are the observations, meaning that they are generated according to the probability distributions attached to the states.

The algorithm works by finding the most probable sequence of phonemes that could generate the input grapheme sequence, according to the probabilities of the model. Using $G = < g_1, g_2, ..., g_N >$ to represent the sequence of graphemes, and $S = < s_1, s_2, ..., s_M >$ to represent the hidden sequence of phonemes, we can formulate the problem in the standard HMM way as:

$$\hat{S} = argmax_S P(G|S)P(S)$$

where $P(S)$ is the prior probability of a sequence of phonemes occurring and $P(G|S)$ is the likelihood of grapheme sequence G given phoneme sequence S. Below we discuss a series of model modifications and experiment which expand on this basic
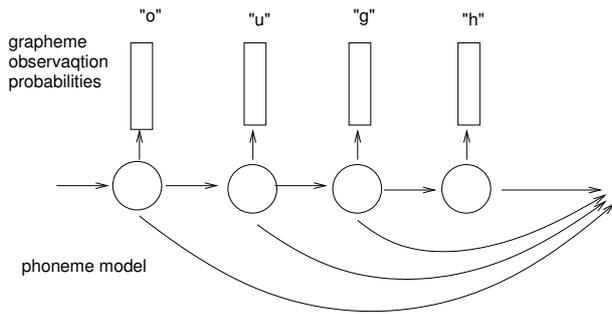
Figure 1: HMM topology and example for phoneme model /ow/

model, but in all cases however the HMM framework is as just described.

Why should this be an appropriate model for GTP conversion? In our view, the phoneme and grapheme sequences have a different nature: the phoneme sequence is seen as the "real", underlying representation of the word, whereas the grapheme sequence is seen as being generated by the phoneme sequence. The relationship between the two is at times quite clear; for example it is clear how the orthographic sequence "hat" can be generated from the phoneme sequence /h a t/. Of course it is not generally the case that one phoneme deterministically generates one grapheme, in words like /ruf/, "rough",; /s t u f/, "stuff" and /g r a f/, "graph", we see that a vowel and /f/ phoneme generate different sequences of graphemes. In the HMM, the graphemes are therefore seen as being generated by the phonemes via a noisy processes, such that given the letter sequence, it is generally non-trivial to determine the phoneme sequence. It is also important to note that it is not necessary for us as model builders to decide which phonemes in /r u f/ generates which graphemes in "rough"; during Baum-Welch training, the HMM aligns all phonemes with all graphemes, weighted by probabilities of that alignment occurring from using the previous set of model parameters. In this way, the model decides for itself whether (say) the "g" of "rough" should be aligned with the /u/ or /f/ phoneme. These alignments may vary from word to word and from training run to training run. Such a treatment is ideal for a language such as English, where spelling is often irregular. We believe the technique is useful for other languages also, in that many new words and names often do not conform to the traditional spelling patterns of that language; furthermore, loan words from English are an increasingly common occurrence.

Each individual HMM represents one phoneme which can generate up to four graphemes. Note that this internal model topology differs from that of a normal speech recognition HMM. There it is usually to have "looping" states which model the time compressibility of the data; in our model we have 4 emitting states, but after emitting an observation from a state the path must move to the next state or the exit state; no looping is allowed. This is intuitive when one considers that each phoneme must be capable of generating up to four graphemes (e.g. the model for /ow/ can generate grapheme sequences "o", "ou", "eau", and "ough", but not for example "ououou"). In nearly all cases the phoneme to grapheme mapping is one to one or one to many. (In only one case - the phonemes /k s/ generating the letter x - is the mapping many-to-one, we address this below). This property of the model is highly significant as it saves us from the so called "null symbol" phenomenon that can be troublesome in other models. This property also makes

it inappropriate to create a model in which the letters are the states and the phonemes the observations; in such a case the hidden states would frequently not generate an observation and this would make the training and decoding of the model difficult.

In the phoneme domain, we have certain constraints and patterns which determine the sequences of possible phonemes. This network of phonemes can be modelled either with or without probabilities. A network without probabilities corresponds to a traditional phonotactic grammar, which states for instance that sequences such as /s k r/ at the starts of words are legal, whereas sequences such as /s g r/ are not. While writing a phonotactic grammar is not entirely trivial (cases such as *vroom* and *sphere*), it is certainly possible to create such a grammar and check that it indeed it does generate all and only those sequences of phones which naturally occur. As far as we are aware, regular languages have the power to describe the phonotactics of all languages, and so using regular languages, which can generate the finite state network is a very natural solution.

The use of a phonotactic network has a significant further advantage; one of the biggest difficulties we encountered when using the decision tree approach was that (despite objectively high accuracy), many of the incorrectly produced phoneme sequences were not only wrong, but were actually phonotactically illegal. This is a severe problem because it means that the actual synthesis algorithm will not be able to generate a waveform from this sequence. There are many sequences though which simply do not occur; they don't occur in any words, the speaker can't physically say such sequences, and so there is no way to have them present in the unit database. The results of ad-hoc correction of bad phonotactic output rarely sound good. With the use of a phonotactic grammar at the heart of the model, this problem is avoided; only legal sequences can ever be followed in the network, which implies that during recognition, only legal phoneme sequences can be produced.

A probabilistic treatment of the state space is of course also possible (and desirable). Here we use the common n-gram formulation, where the probability of being in a state is dependent on the previous N states. While the phonotactic grammar accurately describes the legal sequences, it doesn't say anything about which occur frequently or infrequently. A n-gram by comparison successfully captures the fact that /ax s t/ at the end of a word occurs about 50 times more commonly than other combinations of vowels and /s t/ at word ends.

In addition to these structural or topological properties, HMMs also have the advantage that a powerful training algorithm exists for global optimising the parameters. By use of Baum-Welch training, we can align the graphemes with the phonemes, estimate the transition probabilities in the state space and estimate the observation probabilities which generate the graphemes from each phoneme state, all in a single, globally optimised step. In most other data driven techniques, several distinct (and hence not optimised) steps are required to generate the parameters. In particular, the phonemes and graphemes have to be aligned by a algorithm which is distinct from the rest of the training procedure. Note that the use of HMMs to perform the alignment stage alone has been used by many researchers (see [3]), but most other approaches only use it for this alignment stage. Our model differs in using HMMs for the entire process.

grammar	Words correct (%)	Phoneme accuracy (%)
uniform	0.56	13.23
unigram	4.56	54.03
phonotactic	13.54	55.89
bi-gram	15.56	73.12
tri-gram	32.22	79.41
4-gram	39.13	85.21

Table 1: Results for different grammars

4. Experiments

All our experiments were conducted on the Unisyn English pronunciation lexicon (available from http://www.cstr.ed.ac.uk) which covers approximately 110,000 words of English. The lexicon is free from abbreviations and misspellings, and for the most part defines "normal" English words, rather than names or words of non-English origin. The orthographic sequences are drawn from the standard set of 26 ascii characters, and there are 42 phonemes in the phoneme set.

As is standard in evaluating GTP algorithms, we took the lexicon and randomly divided it into a training set (85% of the entries), a development test set (5%) and an evaluation test set (10%). All reported results are on the evaluation test set. Two measures of performance are used; **phoneme accuracy** is a measure per phoneme, which scores insertion, deletion and substitution errors equally; **words correct** is a measure which counts a word as correct if all the phonemes within it are correct. Note that for this problem, words correct is the vital measure.

4.1. The effect of grammars and n-grams

A separate HMM was created for each of the 42 phonemes in our phoneme set. Training was performed by full "embedded" HMM training; that is; by using the Baum-Welch algorithm without any prior specified alignment of the phonemes to the graphemes. The models are initialised with "flat" likelihoods, that is all phoneme models have identical, uniform observation and transition probabilities.

Table 1 shows results from a series of experiments using a basic HMM, with uniform priors, unigram models, a non-probabilistic phonotactic grammar, bigram, trigram and 4-gram. The n-gram models are trained on phoneme sequences in the training data using Katz-style back-off [3].

The results clearly show how useful the prior information is. As the number of phonemes is small compared to the number of words used in a typical speech recogniser word n-gram, it is possible to robustly estimate the probabilities for high values of n, which leads to the 4-gram being the most successful model. It is interesting to note however that the non-probabilistic phonotactic grammar beats a unigram and is only marginally worse than a bigram. Even bigrams tend to generate illegal sequences, ie the specific problem that the phonotactic grammar solves. Beyond this the use of the phonotactic grammar does not help much as the 4-gram hardly ever generates an illegal sequence (experiments combining the two approaches did not significantly improve on results).

4.2. Pre-processing

One drawback of the HMM model is that it enforces a strictly finite state generative framework, and this can not model some of the subtleties of the problem. From error analysis on the development test set, we identified areas where we could attribute the errors to this aspect of the model. These were nearly all to do with context affects arising from sequence graphemes. While below we show that modelling context between *phonemes* produces significant results, it is important to realise that a standard HMM can not model dependencies between the observations. To counter this, we developed a deterministic preprocessing algorithm that rewrites the graphemes of the input into a slightly rearranged form. By far the biggest problem in terms of numbers of errors arose from problems with "silent e" in words like "hate". The problem is that the "e" itself is not pronounced but it does have an effect on pronunciation, namely that it indicates the vowel should be a long one, as opposed to a short one in words such as "hat". Similarly, "x" was rewritten as "ks". Our pre-processor consists of a small number (< 10) of rewrite rules. The "silent e" rule swaps the order of the "e" and "t", so that "hate" becomes "haet". This is still distinct from "hat" but much more amenable to finite state analysis. With the pre-processing rules, our best system improved to 49.64% words correct and 87.02% phoneme accuracy.

4.3. Context-sensitive modelling

The notion of context-sensitivity is well understood in the grapheme domain; that after all is the basis on which most rule and data-driven models work. As just explained, this can not be modelled directly by our HMM (the HMM can only model dependencies between graphemes and phonemes and between phonemes and phonemes, not between graphemes and graphemes). This however is not a major problem because it appears that most (if not all) context sensitivity which appears in the grapheme domain also appears in the phoneme domain. To take an example, grapheme "c" before "e" or "i" is usually pronounced as an /s/, in other contexts it is pronounced /k/. However, we see a similar effect in the phoneme domain, where sequences of /s/ and certain vowels are likely to produce certain grapheme sequences, which sequences of /k/ and other vowels produce different grapheme sequences. Hence it is possible to model these effects in the phoneme domain. In fact one can go so far as to say that perhaps the "context-dependency" in the grapheme domain is in fact an illusion; the real dependencies lie elsewhere in the model.

We enhanced our system by using context sensitive models everywhere where it was possible to find more that 20 tokens in the training set (as with acoustic triphones, it is generally not possible to arrive at robust estimates for all models).

These models were trained by cloning the context independent models and applying further runs of Baum-Welch. As with before, no forced or fixed alignment was used. When combined with the 4-gram and pre-processing the scores improved to 57.31% word accuracy and 90.98 % phoneme accuracy.

4.4. Stress patterns

A significant number of remaining errors were due to incorrect stress assignment, specifically due to a full vowel being mis-classified as schwa or a schwa being mis-classified as a full vowel. Even a superficial analysis of syllable patterns in words will show that these are by no means arbitrary. Clear patterns show, for instance, with the exception of some function words, all words have one and only one primary stressed syllable, and none have only sequences of schwa. The HMM we have described finds it difficult to model these patterns however as there are too many consonants between each vowel for a 4-gram to adequately model these effects.

We therefore built an entirely new syllable level HMM that

grammar	Words (%)	Phonemes (%)
4-gram	39.13	85.21
+ pre-processing	49.64	94.56
+ context-sensitive models	57.31	90.98
+ stress adjustment	61.08	92.28

Table 2: Final results for words correct and phoneme accuracy

only modelled vowels. In our lexicon each vowel represents exactly one syllable, and as stress is a feature of syllables, building a vowel only model is equivalent to building a syllable model. For this, we deleted all consonants (except y) in the grapheme definitions, deleted all consonants in the phoneme definitions and then replaced each phoneme by the stress number of its vowel/syllable. As we have three levels of stress in our lexicon (primary, secondary and reduced) this left us with just three HMMs. We trained the models as before, but this time the 4-gram was in effect modelling the prior probability of a given sequence of stress patterns occurring, and as expected gave high probabilities to sequences such as *primary*, *reduced* (the most common pattern in two syllable nouns) and low probabilities to patterns such as *reduced secondary primary* (occurring in words such as "appointees"). Sequences that did not occur were taken as impossible, and accordingly no floor or other smoothing of the probability mass was performed.

This simple model correctly predicts the stress pattern of 76% of words. We have not as yet been able to properly incorporate this into the main model, rather we have used a heuristic, whereby if the output of the main model disagrees with the output of the stress model, but a transcription agreeing with the stress model is found in the next N-hypotheses, then the agreeing hypothesis is taken as the answer. From trial and error on the development test set, we found that this was a good approach for values of N up to 8.

5. Conclusions

Table 2 gives a summary of our main results. As demonstrated by the results, the canonical HMM alone is not sufficiently powerful, which led to the motivation for pre-processing, context dependent models and stress adjustments. We believe however that the inherent strengths of the approach outweigh any constraints imposed by the model structure, and further believe that most of these constraints can be bypassed by appropriate data processing and model design.

The pre-processing step obviously has a significant impact on performance, but is a little unattractive in that it uses "knowledge" and is deterministic and not learned (that said, the "knowledge" is no more than that found in the inner page of most English dictionaries). We believe it is possible to learn this step; essentially all that is going on is another finite state process which takes certain sequences and performs minor rewriting to generate other sequences. There are a number of ways to learn this automatically; the most obvious being some technique which rewrites based on a measure of lowering entropy.

While the stress adjustment stage is fully automatic, it is at this stage separate from the main model. Again it is possible to see how this can be improved. Recall that the problem with stress arises because the phoneme n-grams are limited in how much history they can use. While it may be possible to train models longer that 4-grams, many words still have 15 or more phonemes. The problem is therefore that the phoneme n-gram can not model sufficient history to span several syllables and

hence several stress patterns. To counteract this, we effectively eliminated the consonants and built a syllable stress model. This stress model could be combined with the main model in a number of ways; one would be to have a two level HMM, where the first level would be the syllables, marked with stress, which would then emit phonemes, which would then emit graphemes. Alternatively, a more intelligent rescoring technique could be used on the phoneme n-best lists or lattices.

There are a number of errors which we believe lie beyond the abilities of this model. From analysing the test data, we see that by far the biggest source of "regular" errors comes from issues to do with morphology. Within this, the two main issues are compounding and plurals/third person verbs. We get many cases of compounds such as "fasttrack" which is transcribed /f a s t t r a k/. The problem is the double geminate /t/ - this never occurs in non-compounded words (which constitute the majority of the data), and so the probability of /t/ given /t/ in the n-gram is low. In terms of graphemes however, a double "t" is very common of course (eg "betting"), but the model has in general learned to generate two "t" graphemes from one /t/ phoneme. We believe it is very hard for the model to get double consonants in compounds correct, and that the only solution is to do a pre-processing morphology stage where we search for known stems in possibly compounded words.

The second case where morphology can help (in English) is in plurals, genitives and 3rd person singular verbs. In English, there are well governed rules about these suffixes; such that an "s" can be a /s/, /es/ or most commonly /z/ depending on whether there is a unvoiced consonant before the /s/. While this rule can be learned by the model, there are many non-plural words that end in /s/ which don't follow this rule (e.g. "axis", "dialysis"). It is probably not possible for a system trained only on phonemes and grapheme sequences to correctly distinguish this; in reality some level of part of speech tagging and morphology is required.

Finally, we point to a less quantifiable feature of the framework. In systems where lexical resource is not an issue, in real usage it is quite likely that many words being passed to the GTP conversion will not be known to the listener as they will be rare words or names. In such cases it is not essential to get the correct pronunciation; rather one that is acceptable to the user. By making heavy use of the n-gram phoneme model, we believe that even when our system gets the answer wrong, the output is still normally a plausible pronunciation given the orthography; that is, it is similar to the pronunciation that the listener might guess.

6. Acknowledgements

7. References

[1] Damper, R. I., Marchand, Y., Adamson, M. J. and Gustafson, K. (1998) "A comparison of letter-to-sound conversion techniques for English text-to-speech synthesis". Proceedings of the Institute of Acoustics 20(6), 1999

[2] A W. Black, K. Lenzo, V. Pagel, "Issues in Building General Letter to Sound Rules", Third ESCA on Speech Synthesis, 1998.

[3] Frederick Jelinek, "Statistical Methods for Speech Recognition", MIT Press, 1998.

[4] Chen, Stanley F, "Conditional and joint models for Grapheme-to-phoneme conversion", Eurospeech 2003