Language Modeling and Grammar Construction for a Hidden Markov Model Continuous Speech Recognition System

Owen LaGarde

Department of Electrical and Computer Engineering
Mississippi State University
Mississippi State, Mississippi 39762
lagarde@isip.msstate.edu

Abstract

An implementation of Language Model (LM) objects for the construction of regular stochastic grammars is presented based on Bigrams and Ngrams for both words in a training text and for phones in phonetic series equivalents for those words. An externally produced word-to-phone dictionary compliant with the Worldbet symbol set is recommended as a supporting resource. A method of deriving Ngrams as the joint product of weighted Bigrams is defined and investigated.

The implementation of grammar construction and polling objects targets a Continuous Speech Recognition (CSR) system's search engine as the principle user; the objects are to provide query response services for probabilities of current and proposed states in the search engine's domain as well as sequences of hypothesized next-states. The model serves primarily to assist in formation of sentences from hypothesized word sequences through the implementation of a token-based grammar and Unigram/Bigram/Ngram generation scheme. Although the current CSR organization does not require phonetic-level support, such support is easily be provided in the token-based model organization. An inherent secondary goal of the project is the creation of a set of robust LM objects for use in CSR experimentation. As such, other models such as Case Based Grammars will be investigated, but since this project will be conducted in parallel with projects developing other aspects of the target CSR system, priority for model implementation is defined exclusively by the needs of the remaining project teams.

The LM is tested using input perplexity as a benchmark and both alternate Ngram generation methods and independently produced models providing similar services as measures of performance. Deliverables include object interface and implementation specifications, results of Ngram generation methodology experimentation, and descriptions of a set of C++ coded language model construction and request servicing objects.

35. Introduction

Type and quality of the Language Model (LM) can be crucial to performance of the speech recognition system through a variety of factors. Representation of a priori knowledge in the input domain, required to estimate the existential probability of a given input sequence, is imbedded in the LM alone. Constraints of the model (vocabulary, processing speed, etc.) define the search space volume relative to that domain and, to a fair degree, the functionality of the resulting recognition system. Accuracy of the model in calculation of probabilities defines the maximum rate of convergence toward a desirable solution in that search space [1,2].

This design assumes an intended use with a Hidden Markov Model (HMM) Continuous Speech Recognition (CSR) system and is therefore most directly applicable to such an application; the primary goal is to implement as C++ objects a series of utility modules which in turn will provide LM services for a specific HMM CSR implementation. Of equal importance, however, is the creation of robust LM objects that can be used in speech recognition development and experimentation with other recognition packages[3,4]. Topics for investigation center on stochastic LM designs which have been well known for some time [1]. In particular, the plan of attack centers on Bigram, Trigram, Ngram, and Co-occurrence probability measures, though difficulties with higher length Ngrams for large vocabularies[2,5] limit feasible approaches to Trigram implementations. The utilities consume a text corpus, generate an N-gram model of that text, and use that model to provide probabilistic measures for hypothesized next states given a sequence of proceeding tokens. Since the recognition system's queries to the LM could eventually require both English text and phonetic sequence formats, addition of a word-phone dictionary for the given input text is also supported. Inclusion of phonetic sequences also prompts the definition of delimiters. Word-grammars model a collection of sentences as a sequence of tokens -- including words and sentence break symbols -- such that possible token series may extend across sentences as discrete instances of token series bounded by explicit sentence breaks. Token series may therefore be

represented in crossing sentence boundaries while allowing queries to consider those same sequences as not cross sentence boundaries as required by the global system. Phone-grammars consider words as discrete phone sequences only; phonetic equivalents to words are modeled in isolation, i.e., with context independent phone symbols.

36. Problem Overview

The basic purpose of the language model is the prediction of patterns in series of input tokens -- for the CSR, spoken English -- much in the same way as the acoustic models employed by the CSR's HMM predict features in the input signal waveform.

36.1. Global Architecture

In general, the target architecture consists of three primary task-defined areas: the front-end, the search engine, and the language model. The front end module transforms the input signal of digitized spoken English audio into a stream of feature vector packets. The search engine finds the hypothesized utterance sequence most likely to match a given sequence in the feature vector stream by evaluating hypotheses for utterances with acoustic models for those utterances. The language model predicts the likelihood of a given hypothesis of utterance given the likelihood of that utterance in a text body representative of the domain of the system's spoken input, thereby decreasing the number of hypotheses the search engine must evaluate and increasing the probability that the hypothesis corresponding to the current feature vector alignment is evaluated before other hypotheses (which in turn increases the speed of the global system).

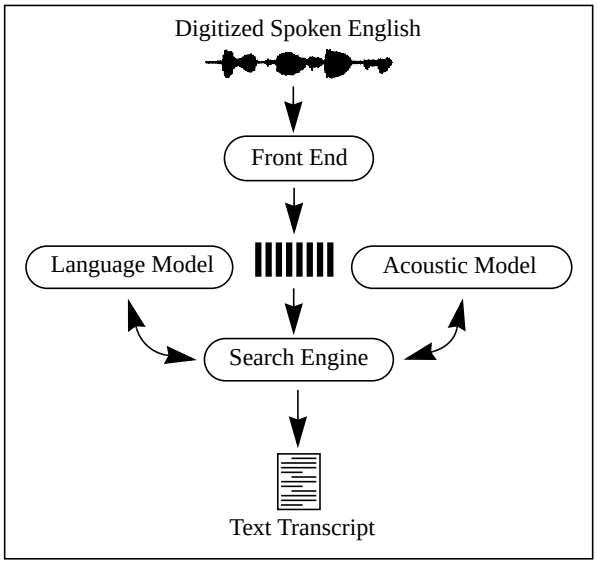

Figure 1: Global Architecture of the Target System

36.2. Functional Requirements

In general, the goal is to supply a HMM search engine with probabilities of correctness for a specific next state (word) given a specific series of previous states; the search engine can then maximize the probability that a hypothesized text sequence is the correct equivalent for a speech input sequence. This, in turn, maximizes performance of the CSR in terms of correct decoding of a speech input into text output. The LM must therefore provide *a priori* contextual knowledge for the linguistic domain of the speech input through access to a body of text representative of the domain of expected speech[6].

The search engine, in locating the best path, creates several additional requirements for the LM functionality; since the recognition system is continuous, the search engine must incrementally decode speech to text across all input for a given execution. This incremental best-path search will require measures of fitness incremental across the input speech sequence, which will require the LM to supply measures of fitness for possible next-states given relatively short sequences of previous states. The search engine can also be expected to justify the pruning of lower quality paths by testing the fitness of relatively long sequences, which will require the LM to supply measures given longer previous-state sequences. Last, the search engine will, as a rule, deal with both words as text and words as phones; the LM services can be expected to operate with both formats. These four requirements -- context, short sequence fitness, long sequence fitness, and word/phone support -- form the core of functional requirements for LM implementation.

In terms of tasks performed for the CSR, the functional requirements of the LM can be expressed as the generation of both linguistic and phonetic grammars and the calculation, on a single token and token series basis, of existential probabilities; stochastic grammars will assist in the hypothesis of probable words while linguistic grammars will assist in the hypothesis of probable sentences. In terms of services rendered to the HMM search engine, functional requirements are limited to the ability to predict near-future events in the input given a current known state. This translates to the generation of existential probabilities for a given sequence, and as an extension of the same, the generation of a list of n-best possible next events given a series of previous events.

It is important to note that the purpose of providing measures of probability is to guide the HMM search engine in selection of next-states given current and past states; this does not necessarily require probabilities, but a means of *consistent ranking relative to probability.* The CSR system definition operates in real time; in general, the faster we can process tasks -- without

appreciable loss in global performance -- the better our resulting system.

37. The Deterministic Approach

The formal definition of a deterministic model includes a *vocabulary* **V**, or set of unique tokens from which the language is constructed, a set of *production rules* **P** governing the manner in which tokens are combined into series, and a specialized token **S** denoting the start of a valid token sequence. The resulting model is known as the *grammar* **G** of the language:

$$G = (V_n, V_t, P, S)$$

.

The vocabulary is divided into *terminals* and *non-terminals*; terminal tokens are symbols that comprise the language, such as English words or phones, while non-terminals are symbols that are expanded via the production rules into series containing at least one terminal. Repeated selection and application of production rules to the starting symbol S will then produce a series of terminals, or a valid sequence of the language for which the grammar G has been defined.

Since the bulk of the model's ability to represent a language is incorporated in the production rules, it becomes fairly straightforward to classify the grammar; there are four widely accepted types have been defined by Chomskey [9]: type 0, *unrestricted*; type 1, *context sensitive*; type 2, *context free*; type 3, *regular.* Type 0 or unrestricted grammars have essentially only one production rule; any series A can be transformed into any other series B: $A \Rightarrow B$. Type 1 or context sensitive grammars bring co-occurrence into focus; sequence A can be transformed into sequence B only in the presence of specific neighboring sequences w_1 and w_2 (which may also be of zero length), A being a subset of the non-terminals and B of the combined terminals and non-terminals: $w1Aw2 \Rightarrow w1Bw2$. Type 2 or context free grammars retain the requirements of type 1 grammars in that A must be a subset of non-terminals and B a subset of the complete vocabulary, but do not impose the constraint of context: $A \Rightarrow B$. Type 3 or regular grammars require the inclusion of a terminal a or b in the result of transforming non-terminals A and B:

$$A \Rightarrow b \qquad or \qquad A \Rightarrow aB$$

38. The Stochastic Approach

In contrast to the deterministic formal grammar, which models language as set of *transformations*, the stochastic grammar models *transitions* between tokens and defines probabilities of the existence of those transitions in the language domain. The inclusion of a measure of probability for the transition of a given production rule set is termed a *stochastic grammar* and is the focus of this project for the functional requirements outlined, particularly in the ability to predict the certainty with which a given sequence will occur given a previous sequence since the probability of a transition can indicate an existential likelihood for the produced sequence.

38.1. Basic Units and Methods

Generation of a stochastic grammar for a given text consists of construction of a Lexicon, i.e., an extended Word Frequency List (WFL) combining the symbols, existential probabilities, and phonetic equivalents of tokens encountered in the training text, and a Matrix of existential probabilities for two-token sequences, or Bigrams. The combination of the Lexicon, Matrix, word-phone dictionary -- and external resource from which phone equivalents to encountered words may be collected -- and creation/management objects and methods therefore comprises the bulk of the language model. The Lexicon's representation schema is token based; words and sentence terminator symbols comprise the token alphabet for linguistic grammars while phones comprise the phonetic grammar alphabet. Although the CSR search engine employes the Viterbi Beam Search algorithm [8] with word-based acoustic models and is not presently expected to require linguistic model coverage of token sequences across sentences or phonetic model coverage across words, this LM project will implicitly supply the capability by representing sentence breaks as special-case **<s>** tokens; run-time configuration for or against such support can easily be added to the project definition if the need arises.

The generation of Unigrams -- measures of probability for the existence of single tokens -- is arguably the simplest of tasks. Given a training text, the probability of existence for a given token can be expressed statistically as the number of instances, or frequency, of that token over the number of instances of all tokens *n* in the text:

$$P(t_i) = \frac{F(t_i)}{\sum_n F(t_n)}$$

The task is further simplified by the existence of the WFL, which is used to collect frequencies of occurrence for each unique token in the training text during the process of constructing the Lexicon. The counting

function then becomes a division of the sum of all WFL entry frequencies by the frequency of the target entry:

$$P(t_i) = \frac{WFL[i, F]}{\sum_n WFL[n, F]}$$

Unigram queries can be construed to fall into the Ngram query category since an Ngram query can be defined to support token sequences of one. Queries for single tokens, two-token series, three-token series, and n-token series are, in theory, separate entities, queries to the LM will exist only in the form of a query for a $P(t_{i,j})$ for a specific token series *i,j*.

38.2. Bigrams, Trigrams, and Ngrams

The generation of Bigrams and Trigrams -- existential probabilities for a two-token series -- is similar to that of Unigram generation in that the basic task still reduces to a counting problem, though in this case the counting problem begins to look familiar to anyone acquainted with the fundamental equation of speech recognition and Bayes rule[5,7]. The measure can be statistically expressed as the frequency of the series in the training text -- similar to the calculation of Unigrams, though we cannot use the WFL to reduce the task further reduce the counting task -- by considering the proposed Bigram itself as a token.

To conceptualize the construction of the Bigram list, consider it's information content. Since the Bigram list is relatively small (compared to a trigrams list or the training text) and scanning the training text prior to grammar generation is already required to generate the WFL and Unigram lists, pointers can be considered as additional tags on entries in the WFL indicating the following token and weighted with the frequency of that [current token, following token] pair found in the training text. The probability for a given Bigram or the measure of the existence of token *b* given a proceeding token *a* can then be statistically expressed as the frequency of that token pair β_i over the sum of frequencies of all token pairs *n*:

$$P(\beta_i) = \frac{F(\beta_i)}{\sum_n F(\beta_n)}, \qquad \beta_i = \{a, b\}$$

The Bigrams list can easily be generated by first producing an extension of the WFL in the form of an *n* by *n* matrix for *n* tokens, a word frequency matrix (WFM) with each position containing the frequency of the x-y indexes' sequence in the text. The probability for any given token pair, given a strictly applied order (P(x,y), for instance) is then the indicated frequency over the sum for the matrix:

$$F_{xy} = WFM[x,y]$$

Although simple to generate, however, this method is certain to introduce zeros into the matrix for xy pairs non-existent in the training text, for which steps must be taken to insulate the search engine from obtaining such an absolute measurement (see Smoothing Functions).

Two factors prohibit the calculation of Trigrams and Ngrams in similar fashion -- processing time and storage space. Such calculations require further scanning of the training text, which adds disk I/O time to our operational speed, or requires us to store the entire training text in memory, usually impossible given hardware constraints. Furthermore, calculation for an *n*-order series of tokens would require frequency measures for all *n*-ary combinations of members of the vocabulary occurring in the training text; the processing and storage requirements to generate the list of possible series alone approaches the prohibitive, though for texts of lower perplexity, generation of a Trigrams list is often possible[4].

Rather than dedicate to the explicit calculation of Trigrams, however, we can note that Ngrams may be calculated as the joint probability of intersecting Ngrams of lower order existing in the target sequence; we can therefore recursively define all Ngrams of order greater than two (Trigrams included) as a sequence of overlapping Bigrams. This method does require the existence of a transitive property across probabilities for lower order token series that does not necessarily exist, but current sources of stochastic LM development[1,3] indicate satisfactory results:

$$P(a,d) = P(a, b) \cdot P(b, c) \cdot P(c, d) \cdot K$$

Experimentation shows that weighting is not actually required, though the inclusion of a scaling constant does allow for a greater range of flexibility and control in the general function, which allows us, in turn, the ability to tune our Ngram generator for longer and shorter sequences, in particular to emphasize sequences based on length since longer sequences produce probability measures so small as to be prohibitive in terms of data structure value bounds. In addition, there are other methods for countering such problems, as are discussed below in the Implementation and Evaluation sections of this document.

39. Word-Phone Productions

The unigram and bigram construction scheme together with the structural representation satisfy the bulk of functional requirements as outlined thus far. One service that remains, however, is the additional (though currently unused) support for phonetic equivalencies of model tokens, or words. Addition of such support is defined through the use of an external, off-line phonetic dictionary and the inclusion of phonetic sequences from that dictionary in the Lexicon where Lexicon symbol and dictionary symbol match. This organization, in addition to being eradicable to a single instance of off-line linear search, introduces the ability to change the phonetic symbol set employed by the global system merely by changing the external file containing the phonetic dictionary.

40. Tokens, Objects, and Methods

Given the conceptually fixed set of resources, we must now turn our attention to implementation, which dictates a further subdivision of functional requirements into two groups. Prediction of future language domain events is the primary goal, encompassing the representative of state space for a specific language and usage domain and the evaluation and ordering of possible solutions based on probability of correctness. Operation in conjunction with a real-time system is the secondary goal, including the requirement of efficient data structures extensible for alternate or variant models and a resource management schema tunable for size and speed considerations.

40.1. External Resources -- File Structures

Goals regarding continuous real-time performance require the stringent use of processing time; to that end, model construction methods can be employed off-line of the global system. In addition, the bulk of construction can be performed with scripts and landmark UNIX utilities, “tr”, “sort”, “uniq”, and “awk” in particular. The lexicon, for example, requires an intermittent WFL which can be constructed as follows: first, the training text is piped through “tr” to transform or remove unwanted characters or character sequences, and to isolate the resulting words, one word per line of text. Second, the resulting list of isolated words is piped through “sort” to arrange the words in alphabetic order and, as a result, group identical words on adjacent lines. Third, the sorted grouped list is piped through “uniq -c” which removes duplicates and adds an integer indicating the number of instances of each unique word. Fourth and last, “awk” formats the list of unique words and their frequencies into an output format of <word> <frequency>. As an optional additional operation, the frequencies can be normalized to Bayesian probabilities by dividing each frequency by the sum of all frequencies with a simplistic two-pass binary executable coded to sum on the first pass and divide on the second as the following pseudo-code illustrates:

```
begin
    sum = 0
    seek file record 0
    while not end-of-file
        sum = sum + frequency
        seek next record
    seek file record 0
    while not end-of-file
        frequency = frequency / sum
        seek next record
end
```

Figure 2: Pseudo-Code for Unigram Generation

In like manner, the produced list is extended to include optional phonetic equivalencies to the contained words by traversing the resulting list and, for each entry, querying an external phonetic dictionary file for the appropriate sequence. In this way, the list (a WFL) becomes the Lexicon, and construction performance is bounded by two traversals of the vocabulary of *n* words plus one traversal of *m* words in the training text. Since the number of unique words in a text must be equal to or less than the total number of words, the performance bound for creation of the Lexicon is on the order of the text length, or O(*m*), assuming all text in the training file is consumed for this purpose. The resulting file is of text format, and retains this format for simplicity of testing and verification. For final implementation, however, the Lexicon is converted to binary format with fixed-length strings used to create a constant sized data structure; this representation is wasteful of on-disk space but in trade is much faster to access and supportive of random access implicitly indexed by the entry count minus one (as with ‘C’ arrays).

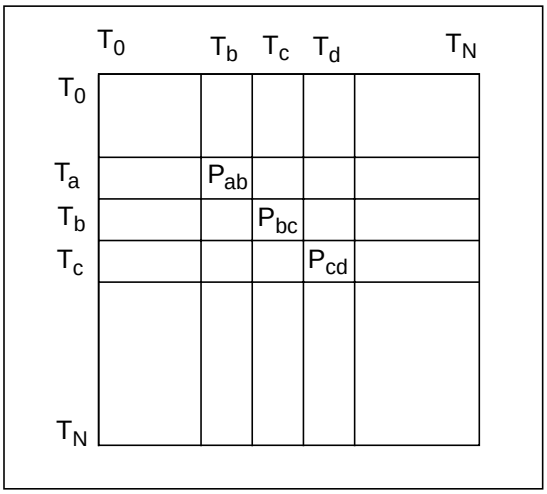

Figure 3: Bigram Matrix showing elements of P(a,b,c,d)

Construction of the Matrix of bigrams is conducted by first counting the number of elements *nxn* in the Lexicon and creating a template binary Matrix file of *nxn* zero values. The training text is then traversed on a word-by-word basis; at each step, the current and following word are considered as a pair and the corresponding Lexicon entries located. Using the Lexicon indexes, the Matrix is accessed at position (current-word index x *n*) + next word index, and the value increased by one; the frequencies of all word pairs in the text are thereby counted. As before, these values can optionally be normalized through division by the frequencies' sum.

The resulting Matrix can then be accessed in similar fashion to the Lexicon, as a two-dimensional zero-biased array, using indexes from Lexicon entries corresponding to the desired first and second symbols in a two-token bigram pair. Furthermore, the Matrix can also be used to produce Ngrams -- at the cost of extended disk access speed -- by sliding a two-token window along the sequence of the Ngram's token series, moving the window one token at a time until the end token - 1 is reached.

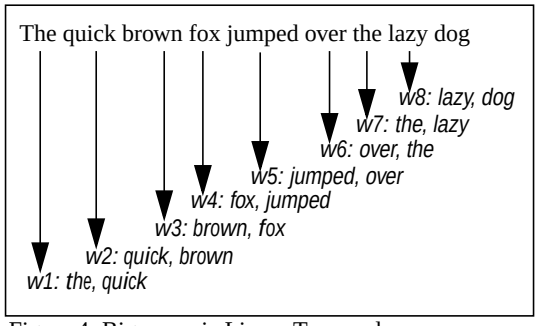

Figure 4: Bigrams via Linear Traversal

40.2. Internal Resources -- Data Structures

Internal representation of the model consists of a network of objects, one object per token, linked via weighted directed edges. The network -- perhaps more correctly visualized as a weighted, directed graph -- is constructed by first creating a *root* node to contain the symbol, unigram, and phones of the Lexicon's single most fit (highest unigram) token. The Matrix will then be traversed, starting with the root's token index, to load all tokens into nodes for all bigrams starting with the root. The loading sequence will continue, using each node in the previous load sequence as root, until a pre-defined load level is traversed in the Matrix. The net effect is to generate a weighted directed graph with nodes for tokens and edges for transition [probabilities for all tokens with a pre-defined neighborhood of the original root. This representation allows a degree of control over the volume of the matrix loaded at any one time, relative to the branching factor of the Matrix, or the number of tokens in the Lexicon, *n*. To query the model for the probability measure for a given token sequence, the controlling model object locates the first token of the sequence, locates the second token in the current token's list of neighbors, and traverses the indicated edge to the new node. The traversal continues as the controlling object collects transition probability measures from the edges traversed; if a node in the sequence is not found in the network, a smoothed "zero" is substituted or, if traversal has led to the edge of the load level's neighborhood for the primary root, the traversal state is saved while the network is replaced, updating with the current node as the new root.

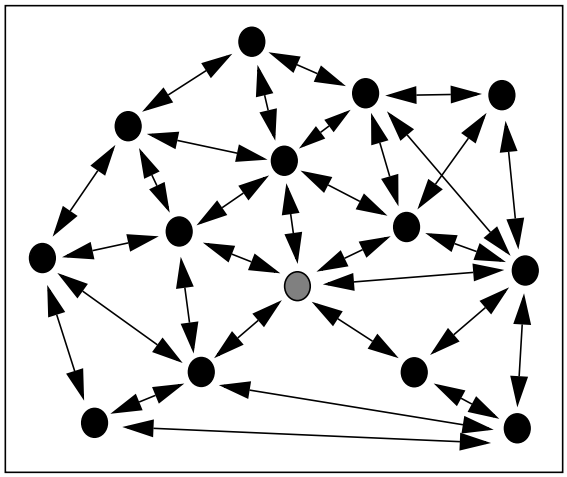
Figure 5: The internal model network, load level 2, primary root shaded

This method introduces several interesting capabilities: first, given a sufficient series of queries the average query length and thus the average number of serviced queries can be predicted between required re-loading of the network; second, given a load level of *l*, the model can guarantee a minimum of *l* total traversals through the network before a reload will be required. Together, these two capabilities supply all information needed for future methods in the controlling object dedicated to optimizing use of memory given current available memory and system performance.

41. Smoothing Functions

Three general cases are defined from which zeros arise in a stochastic language mode. First, the model can contain a zero for an unknown token; if a symbol does not exist in the training data, it will also be absent from the vocabulary, and the existential probability (without smoothing) is then zero. Second, the model can contain a zero for a contextual sequence that did not exist in the training data, such as with bigram pairs composed of a known token (i.e., found in the training data and thus located in the net vocabulary) and an unknown token; a single zero (unknown token) in the calculation of an Ngram probability will zero the function. Third, the

model can contain a zero due to constraints imposed by the search engine and acoustic model; if the search engine requires the model to be discontinuous across sentence breaks, for example, the language model will contain zeros for the transitions that exist exclusively across sentence boundaries.

For Unigrams, probabilities are generated only for known tokens (sequences of one). Bigrams, however, introduce the side effect of including two-token sequences that do not exist in the training text and therefore carry a measure of zero. While accurate in representation of the training text, the purpose of the model is not to re-create the text but to predict events in bodies for which the text is representative; a zero probability will force the HMM search engine to ignore any possible longer sequences incorporating the Bigram in question; zeros must therefore be replaced with a proxy value greater than zero. Values greater than the WFM minimum would be equally misrepresentative, aligning a Bigram non-existent in the training text with those that are. Given the extremely small probabilities resulting from frequency counts of small token sequences and large bodies of text, this project employs a smoothing function of a constant replacement of all zeros with the minimum value possible for the implantation data type, thus ensuring that the value, while greater than zero, will be less than the minimum Bigram measure with a positive frequency (or equal, if the data type is insufficient for the range of actual probabilities).

42. Grouping and Class Grammars

Class grammars, though not incorporated into this project, are examined as a grouping tool with interesting capabilities given the stochastic network representation chosen. Since the model is token-based and gains it's language tokens from a pre-processed training text body the symbol set can be exchanged for the language as easily as for the phonetic equivalents of the language (see Word-Phone Productions). The nodes of the stochastic network literally embody the probability of transition between tokens, but can also be interpreted as representing the probability of transition between _groups_ of tokens (at the moment, groups of one token each. Given a network constructed from a training text in which words were replaced with the used part of speech, the token-based model would represent the transitional probability between word classes, class being defined by English usage rules. Such a model would not, however, constitute a class grammar, because no such transition would insure the production of a state with at least one terminal symbol. Rather, the class model would predict the likelihood of transition between parts of speech. To imbed such transformations the class model could, however, be loaded in parallel with the stochastic model and a link layer defined connecting the two such that stochastic nodes for a given symbol were linked to all class nodes of parts of speech representable by the symbol. The link layer would then constitute a hybrid between stochastic and deterministic grammars and could access the network layer of least confusion for the current state, i.e., least variance between transition probabilities to immediate neighbors. This ability is especially interesting as a topic for future research since the implication is toward a model encompassing multiple symbol sets, from which one is selected for each given transition based on the level of confusion at the current state.

43. Implementation and Experiments

The language model is implemented in two distinct modules: the first, dedicated to off-line services performed before the instantiating of the running CSR system, is composed of two binary executables "lexicon.exe"and "matrix.exe"; the second, dedicated to on-line services performed during training and normal operation once the CSR system begins processing digitized audio, is composed of object and method definitions encapsulating the routines required to load, update, and respond to queries on the model network.

43.1. Off-Line Services

The two executables "lexicon.exe" and "matrix.exe" are utilities that, respectively, build the Lexicon and Matrix files from the training text. For purposes of testing and experimentation output was directed to text files, but for incorporation into a global system the file formats are changed to fixed binary typed. Given the computational simplistic tasks, three separate approaches were examined: scripts that call legacy UNIX utilities; single files with complex structure; multiple files with simple structure.

To approach the task with scripts and legacy utilities is by and far the most portable and modifiable as the scripts make piped calls to such utilities as "tr", "sort", "uniq", and "awk". The following sequence, for example, can be used to create a vocabulary complete with frequency counts from an arbitrary text:

```
cat text | tr -s '[a-zA-Z0-9]' '[\n*]' | sort -y |
    unique -c > vocab
```

The labels "text" and "vocab" refer to, respectively, the training text body and the vocabulary file. The

```
tr -s '[a-zA-Z0-9]' '[\n*]' | sort -y
```

command accepts the streamed training text from "cat" and removes all characters not alphanumeric, isolating the remaining words each as a single word on a line.

This action results in multiple instances of a word being translated into multiple instances of that word alone on a line. The sort utility accepts the data stream from “tr” and outputs the stream sorted by alphanumeric, resulting in multiple instances of a word being grouped together. The “uniq -c” then removes duplicates of any symbols found, leaving a count of the number of instances initially located for each symbol. The net result is a alphanumeric-sorted list of unique words in the training text along with a frequency count for each.

It should be noted that although the scripted approach is highly portable and makes excellent use of the UNIX environment to employ multiple parallel pipes, the actual scripting ability is limited to tasks which reduce entirely to counting functions. A binary executable is still required to move from the Lexicon to the Matrix.

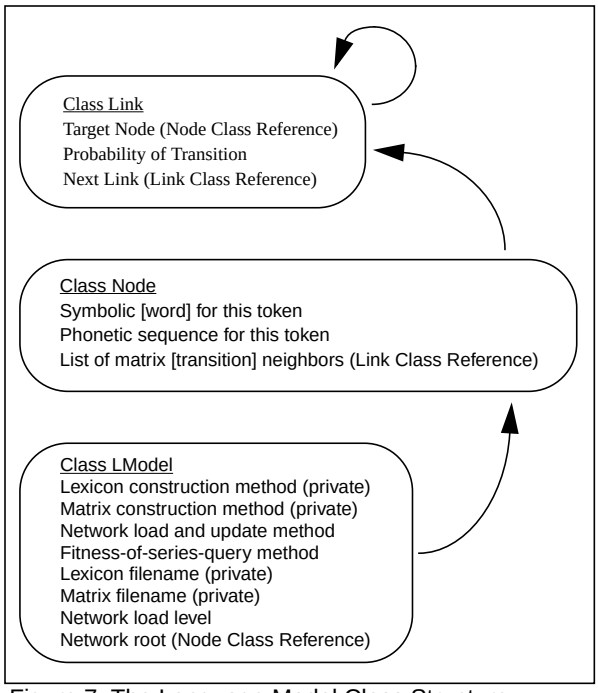

Figure 7: The Language Model Class Structure

The remaining two approaches involve coding binary executables to parse and count from the training text to the respective off-line resource files; these approaches differ only in attitude toward the role of complex or simplistic object definition. The more simple object design is preferred for the ease with which it can be incorporated into other LM code libraries at later dates. The core of the resource file generation programs is a simple node object that serves to collect frequency counts, or “hits”, for a specific symbol encountered in the training text and to maintain the location of the “next” node, thus internally maintaining a linked list structure. The detection of bigrams vs. unigrams is decided by the symbol pattern stored as the symbol for each node’s token; once the entire training text is consumed, the list of nodes is written to file in the order dictated by the particular resource file format.

43.2. On-Line Services

The class definitions for network load, update, and query response are fairly simplistic; little computational energy is required save for the task of loading the root’s neighborhood of bigram data from the off-line matrix. Even in that aspect, the principle concern is the tracking of nodes currently loaded such that a given node’s bigram is not duplicated, destroying the integrity of the network. To that end, the update method is used both to initially load and on request reload.

Since the update method must therefore already track currently loaded nodes, an extension of the network update method is proposed: since the load level is known before and after a network update, and since the list used to protect against multiple loads of a bigram must employ knowledge of the level of the node relative to the current root, the same list and information can be used to minimize the number of additional nodes which must be unloaded or loaded during an update.

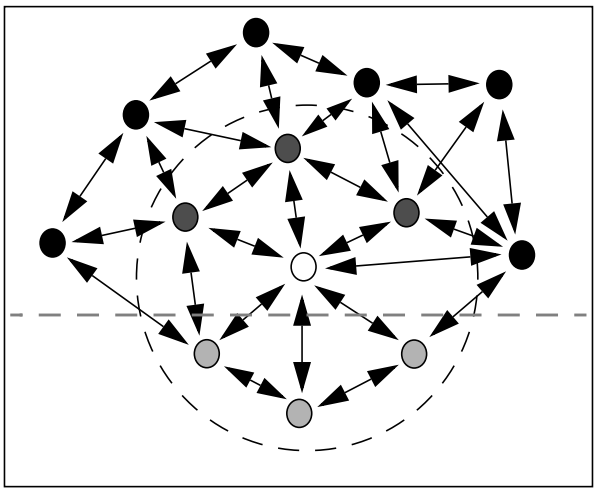

Figure 8: A minimal-change update to the model network

Consider the above figure 8; the entirety of the network lies above the horizontal dashed line and a query calls for a traversal through the white node and across the network boundary. If current-level information is employed at each node and assuming a load level of one, the update method need only unload the black nodes, load the light grey nodes, and leave the dark grey nodes untouched.

44. Evaluation

Unfortunately, little can be done to benchmark external performance of the LM independent of the CSR client; parallel implementations [if possible] of the CSR -- one

with this project's LM and one with another LM -- can provide some means of performance measure relative to the two models, but the bulk of meaningful performance benchmarking must occur internally.

$$Q(\underline{w}) = 2^{\hat{H}(\underline{w})} \approx \frac{1}{\sqrt[N]{\hat{P}\left(w_1^N\right)}}$$

$$\hat{H}(\underline{w}) = -\lim_{N \to \infty} \log \hat{P}\left(\underline{w}_1^N = \underline{w}_1^N\right)$$

Figure 6: Perplexity (top) and Entropy (bottom)

Input to the LM is a large body text, therefore the natural benchmark is perplexity of the model produced from prospective text corpora; however, perplexity of a model represented by a complete directed weighted graph is less meaningful than isolated factors of entropy at and across individual nodes in the network, and neither yields a good measure of the efficiency of off-line resource production. Testing is therefore performed in three stages: first, Ngram generation methodology is verified comparing generated Ngram measures; second, model bigram and unigram measures are compared with that of an independent LM; third, production routines will be compared across series of input characteristics to detail the order of performance.

Ngram measure generated by multiplying series of bigram measures exactly matched those gained by searching the training text -- though one should note that Ngrams (in specific, five-token series) were chosen that were known to exist in relatively high numbers. Although the CMU Toolkit scripts used to produce Ngrams for comparison employed a measurement range and scale several orders of magnitude above our own (the CMU scripts do not normalize frequency counts but use them directly), the order and content of an n-best token list given a random current token state is practically identical between both models. Deviation exists only where the project's normalized probability measurements approach minimum values for the chosen data type, a restriction handled automatically for the CMU scripts by the operating environment and command shell. Comparison of produced Lexicons for a test body of text -- Friedrich Wiesler's *Nature of Value*, a 100+ word online book on financial and socio-economic value standards -- produced deviation between CMU and project models identical to that found in Ngram comparison; the obvious conclusion is that the project's data type containing the probability measures is poorly matched with the range, if not domain, of actual probability measures. This discrepancy can be overcome directly through one of three basic approaches: one -- increase the bit-size of the containing data type; two -- modify the scaling constant in the Ngram probability function to produce a range of values shifted towards greater magnitudes (the problem areas all lie in close proximity to long token sequences and extremely small competing probabilities); three -- modify or abandon the normalization of unigrams' and bigrams' probability measures to the same effect.

45. Conclusions

The Language Model (LM) is shown to play a key role in the performance of the Hidden Markov Model (HMM) Continuous Speech Recognition (CSR) system; quality and type of the LM is expressed as a limiting factor of maximum performance of the CSR. Standard approaches and methods in LM design and construction are examined, and a stochastic model chosen. Methodology is introduced to streamline production and incorporation of the language model and allow for a relatively wide freedom movement in modifying the model mechanics and performance.

References

149. **Deller, J. R., J. G. Proakis, and J. H. Hansen. 1993.** *Discrete-Time Processing of Speech Signals*. Macmillan Publishing, New York, pp. 677-804

150. **Rosenfeld, Ronald. 1995.** *CMU Statistical Language Modeling Toolkit: Manual*. Carnegie Mellon University Internet WWW Service, online URL "*http://www.cs.cmu.edu/afs/cs//user/roni/WWW/toolkit-SLT95-revised.ps"*

151. **Rosenfeld, Ronald. 1995.** *Optimizing Lexical and N-gram Coverage Via Judicious Use of Linguistic Data*. Carnegie Mellon University Internet WWW Service, online URL "*http://www.cs.cmu.edu/afs/cs//user/roni/WWW/vocov-eurospeach95-proc.ps"*

152. **Rosenfeld, Ronald. 1995.** Adaptive Statistical Language Modeling: A Maximum Entropy Approach. Ph.D. Thesis, Carnegie Mellon University, Technical Report CMU-CS-94-138, Carnegie Mellon University Internet WWW Service, online URL "*http://www.cs.cmu.edu/afs/cs//user/roni/WWW/thesis.ps"*

153. **Picone, Joseph. 1990.** *Continuous Speech Recognition Using Hidden Markov Models*. IEEE ACASSP Magazine, July 1990:26-40, IEEE Society Publications 1990

154. **Picone, Joseph. 1994.** *Context-Sensitive Statistical Signal Processing: Toward User Configurable Speech Recognition.* Systems and Information Science Laboratory, Texas Instruments, presentation March 22 1994

155. **Chomskey. 1959.** *On certain formal properties of grammars*. Information and Control 2:137-16, Dellar

Associates

156. **Ries, Klaus, Finn Dag Buo, and Ye-Yi Wang. 1995.** *Improvised Language Modeling by Unsupervised Acquisition of Structure*. Interactive Systems Laboratories, Carnegie Mellon University, ICASSP Lecture Notes May 1995, Carnegie Mellon University Internet WWW Service, online URL "*http://www.cs.cmu.edu/afs/cs//project/cmt-38/ries/www/icassp_95.html*"

157. **Rudnicky, A., F. Lee, and A. G. Hauptmann. 1994.** *Survey of Current Speech Technology.* Communications of the ACM 37(3):52-57, Carnegie Mellon University Internet WWW Service, online URL "*http://www.speech.cs.cmu.edu/rspeach-1/air/papers/cacm94.ps*"

158. **Koppleman, J. 1995.** *A Statistical Approach to Language Modelling in the ATIS Domain*. MIT Department of Electrical Engineering and Computer Science, January