# Language models: where are the bottlenecks?

András Kornai
IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120

## Abstract

*Statistical, parsing, database, and other methods of bringing contextual information to bear on the recognition task are described in a uniform framework in which the central data structure mediating between the recognition and the contextual components is a segment lattice, a directed graph that contains the alternative segments and their confidence/probability ranking. Explicit measures of the value of such segment lattices and the correctness of language models are proposed, and the dominant technologies are critically evaluated.*

## 0 Introduction

Since stand-alone recognition is unlikely to provide high quality output in the foreseeable future, the overall performance of recognition systems will continue to be critically impacted by the language model. Section 1 of this paper provides a critical overview of the traditional system architecture in which a language model component acts as a postprocessor correcting the output of the recognizer, and proposes explicit measures of the contribution of the components to the reduction of error rate. Section 2 discusses the more current Hidden Markov Model (HMM) architecture in which recognition and language modeling are performed in parallel. The main bottlenecks identified in this survey appear in **boldface**.

## 1 Language modeling as error correction

Laboratory systems for the recognition of written or spoken input were first demonstrated in the fifties, and as soon as the size of the permitted input exceeded a few dozen fixed templates, some sort of contextual analysis component was added (Bledsoe and Browning 1959). From early on, speech researchers preferred stochastic models, while in optical character recognition (OCR) the dominant model was, and to a significant extent continues to be, lexicon-based. The typical architecture of such models can be depicted as follows:

*(preprocessed) signal*

$\boxed{\text{Recognizer}}$

*segment grid*

$\boxed{\text{Fixup}}$

*segment lattice*

$\boxed{\text{Language Model}}$

*output strings*

This model has its origins in recognition tasks such as OCR of programmers' data entry sheets (Munson 1968, Duda and Hart 1968) where segmentation is not much of a problem either at the character level (because evenly spaced ticks are provided on the form) or at the word level (because whitespaces are trivial to detect). *Fixup* was added only in later systems when it became clear that a simple grid (whith a column of hypotheses for each input "segment") can not accomodate segmentation hypotheses of different length. Over the years, the Fixup component also tended to assume the task of modeling the error characteristics of the recognizer: what started with adding an $u$ or an $m$ to the lattice where the recognizer had $ii$ or $iii$ was soon followed by adding $q$ for $g$ and $c$ for $o$ if that was what the recognizer systematically missed.

There are two basic problems with this architecture but only one of them, **combinatorial explosion** has been addressed from the beginning. The language model can be a simple list of words, a more complex algorithm dealing with capitalization, affixation, and other problems of dictionary lookup, or it can be a very sophisticated parser or database module. But whatever it is, calls to this component are expensive, and recognition is feasible only if the number of paths through the segment lattice is kept small. On the other hand, the recognizer (even after fixup) is not perfect, and if the correct hypothesis is not output among the alternatives, a local error (failure to suggest the correct segment) becomes a global error (failure to find the correct string). Therefore the number of alternatives must be kept high, which creates exponential growth in the number of paths through the segment lattice. For the larger part, the history of error correction and language modeling is the history of fighting against this combinatorial explosion.

But there is another, more subtle, problem with the model depicted above, which became clear only in light of the experience of speech researchers, and this is the **lack of joint optimization**. Broadly speaking, there is a tunable parameter of each component: the number of hypotheses $N$ provided by the recognizer, the number of error patters $E$ considered for substitution by the fixup, and the aggressiveness of the pruning $A$ in the search, and in principle we can use something like gradient descent to find a global optimum in the space described by $N$, $E$, and $A$. But from a more technical perspective the parameters are not so easy to set: the confidences of low probability hypotheses are unreliable, the error patterns change with the setting of $N$, and controlling $A$ is a complex undertaking which often involves software engineering decisions that cannot be implemented as a change to a single numeric parameter. As a result, instead of systematic optimization by techniques guaranteed to converge, in practice we find lots of hand-tuning which is both labor-intensive and liable to miss the real optimum.

To resolve this problem we need to identify and measure the contribution of each component to the overall recognition rate. As a first step, let us define a *Language* over a finite alphabet of phonemes or graphemes (including pause or whitespace) as a function $f$ that assigns a non-negative probability $f(s)$ to every string over the alphabet in such a manner that the sum of these is bounded (can be normalized to 1). To forestall confusion it should be emphasized that probability is not meant as a numerical scale of degrees of grammaticality: syntactically well-formed strings can have zero or low probability and syntactically ill-formed strings can have relatively large probability. The definition of Language (with capital L) embodies the simplifying assumption that string probabilities are fixed once and for all, though in estimation tasks it has been often noted that for low values the static probabilities are outweighted by context effects. To fix ideas, a Language is best thought of as the set of strings that will be encountered by the recognizer, weighted by the frequency of such encounters. Though hard to measure or even estimate, this is a static population, fixed once and for all by the intended application domain.

A *Language Model* is defined as any combination of table lookup and other algorithmic procedures that will assign a non-negative number to each string over the alphabet. We do not require the sum of these numbers to converge because we do not wish to exclude those language models which approximate probabilities by a zero-one decision but permit an infinite number of valid strings. Given an alphabet $T$, a Language $f : T^* \rightarrow R^+$, a Language Model $g : T^* \rightarrow R^+$, and a precision $\epsilon > 0$, we define the *underestimation error* $U(\epsilon)$ of $g$ with respect to $f$ by

$$U(\epsilon) = \sum_{s \in T^*, g(s) < f(s) - \epsilon} f(s) - g(s)$$

and the *overestimation error* by

$$V(\epsilon) = \sum_{s \in T^*, g(s) > f(s) + \epsilon} g(s) - f(s)$$

.

The impact of such errors on the overall performance of the system will of course depend on the manner in which the Language Model is utilized. For example, if recognition/fixup always returns two strings, the correct one and a random one, and the LM simply picks the one with higher probability, a perfect LM ($f = g$) will make the right choice with probability one, since the probability of a random string is zero. However, if the LM is a parser that accepts (returns the value 1) for every string of even length, the model will induce the incorrect choice for all cases where the true candidate has odd length while the other hypothesis has even length. Let us suppose this parser only makes 5% error because $p = \sum_{|s|=2n} f(s) = .95$. Running the recognizer/fixup without the Language Model means we have to randomly pick one of two candidates i.e. an overall recognition rate of 50%. Running with this particular LM (which will force a random selection only if both candidates are odd or both are even) improves the overall recognition rate to 72.5%, which is considerably better, but somehow less than what we would expect from a language model that is, after all, 95% correct. Including the LM remains profitable as long as it expresses a genuine regularity about the Language (for all Languages with $p > .5$) but even for the ideal case where our "grammar" is perfect ($p = 1$) the error rate will be 25%.

The above example is somewhat unrealistic: most recognizers will rank their output or the strings returned by fixup are endowed with some *ranking, confidence* or *probability* measure, either on a character by character basis or as a global measure normalized for string length. In the absence of a language model all we can do is pick the highest ranked choice, and I will define the *value* of a segment lattice $\alpha$ output by the recognizer for input $s$ as

$$v(\alpha) = \frac{P(s \mid \alpha)}{\sum_{t \in \alpha} P(t \mid \alpha)}$$

i.e. the probability of picking the string $s$ divided by the sums of probabilities of picking any of the strings $t$. Clearly, if the recognizer returns not only two strings as above (value .5) but also a ranking that is .95 for even and .05 for odd strings, the value of the segment lattice is greatly enhanced (actually to the point that the LM can no longer improve it). Since the segment grid is a special kind of segment lattice, its value is defined by the same formula as above. Fixup works well to the extent it improves the value of the hypotheses for frequent inputs more than it breaks on infrequent ones. If the segment lattices before and after fixup are $\alpha_s$ and $\beta_s$, fixup brings $\sum_{s \in T^*} f(s)(v(\beta_s) - v(\alpha_s)) \sim \sum_{s \in T^*} g(s)(v(\beta_s) - v(\alpha_s))$ increase in value. Note that good Language Models can improve things even when the value of the recognition/fixup contribution is zero – this will happen any time the LM picks a correct solution which was not present in the segment lattice.

Let us now turn to cases where the grammar is more complex. In most applications the dominant problem is lack of coverage (underestimation) which can be caused by several factors. First, **inappropriate setting of the alphabet** (in OCR, lack of training for certain symbols) is not at all uncommon. What we call the Latin alphabet started out as a set of two dozen symbols, and the addition of lowercase only doubled this number. But over the centuries a considerable number of special symbols were added for punctuation, mathematical symbols, accented, crossed, and other modifications of the existing letters, monetary symbols and other ideograms like ©, so in a comprehensive inventory like Unicode we find several hundred relevant symbols.

Second, typical language models do not express regularities directly in terms of the underlying alphabet but rather through the interaction of several strata. It is assumed that one symbol of the underlying alphabet, namely whitespace, can be used to parse the string into *words* over which regularities can be stated more clearly. Given a base alphabet $T$, we now have a *dictionary* $V \subset T^*$ such that strings of the Language over $T$ are expressed as strings over $V$. Further (phrasal, sentential) strata are often added in the same manner. There are two problematic assumptions here. First, the **assumption of parsability** – typically, dictionaries ignore (mis)hyphenated and runon forms, and if further strata are added the same assumption is repeated at the phrasal, sentential, etc. levels. Second, the **closed world assumption** that drives the never-ending search for larger and larger dictionaries, ignoring the simple fact that in most domains $V$ can not be expressed as a finite list. It is not uncommon for hyphenation/runon problems to cause as much as 5%, and lexical coverage problems to cause over 30% of the underestimation errors of the language model.

To take a Language of medium complexity, US mail *address blocks* are ideally composed of NAME, NUMBER, STREET, CITY, STATE, and ZIP fields, in this order, and a simple unigram Language Model per field is not hard to build. The overall model then takes the probability of an address block to be the product of the probabilities of the entries within a field, ignoring the relationship between city, state, and zip. But if we take names to be single entries, Turing-Good estimates on coverage will show that over 40% of names will be missed even with lists comparable in size to the biggest dictionaries in use (several hundred thousand entries). Thus we are forced to parse names into several fields such as TITLE, FIRST-NAME, MIDDLE-NAME, and LAST-NAME, and some of these fields can now be empty. The dictionary coverage problem is less acute within these fields (though it is still hard to get better than 95% coverage on first names and better than 90% on last names) but there is a price to pay both in terms of overestimation (Latino first names with Vietnamese last names are predicted with higher probability than what the Language actually has) and in terms of **pattern complexity** to which we turn now.

While the regular English pattern (TITLE) (FIRST-NAME) (MIDDLE-NAME) LAST-NAME is indeed the dominant one in the Language of personal names with over 60% of the cases, it is far from the only one. There are GENERATION markers such as *Jr., 3rd,* and so on, which can be added to the above pattern without breaking the linear order. But there are other patterns, such as *Smith, John* (often without explicit comma) or *John cardinal O'Connor,* which cannot be absorbed in the dominant pattern and will require their own separate rule. Also, dominant/frequent patterns of other languages will appear with lesser, but not negligible frequency: *C. de la Vallée-Poussin, B. van der Waerden,* and so on. Finally, subfields can often be iterated *(Herr Prof. Dr.)* or coordinated *(Commodore and Mrs. Smith, H. Rodham Clinton).* Anybody who has ever written a parser for sublanguages will also know that some of the Language is just plain wrong: misspelled, scrambled, semantically erroneous. It is an important design decision whether we incorporate such **ungrammatical entries** in our Language Model and if not, whether we prune such entries aggressively even if the recognizer/fixup module returned high confidence results.

## 2 Language modeling as constraining the search space

Most of the key problems with the standard methods of parsing natural language input are already visible on the limited domain of address blocks: weak dictionary coverage, failures of parsing into higher strata elements, endemic pattern complexity, and ungrammatical input all exact their toll on the Language Model. In addition, the dominant technology of parsing, using (augmented) context-free grammars, leads to overestimation problems by the very nature of context-free rules. If we introduce very narrow dictionary classes like

LATINO-FIRST-NAME and VIETNAMESE-LAST-NAME we can avoid the overestimation problem discussed above, but **obtaining reliable estimates** of unigram probabilities will require several orders of magnitude more data. We can have a good idea about the relative frequency of *Jack* or *Bill* among FIRST-NAMEs but it will require more data to find out their frequencies among CHINESE-FIRST-NAMEs or OTHER-FIRST-NAMEs.

These problems, coupled with the somewhat diffuse but often quite palpable desire to "fire a linguist and increase productivity" led, in the past decade, to the widespread adaptation of a simpler (finite state as opposed to context free) class of Language Models. Originally developed by Markov (1913) for the purpose of studying word frequency distributions, Markov chains came to be viewed as a realistic alternative to more complex models for several reasons. First, such models were easy to integrate into the Hidden Markov Model (HMM) paradigm that dominated speech research since the eighties. Second, automatic training algorithms that created a Language Model with little or no human intervention were available, replacing the labor-intensive process of writing a parser. Third, the Markovian assumption leads to models that can be directly used for left-to-right pruning.

As we noted in Section 1, calls to a parser are expensive, and in a typical context-free parser the number of steps grows with the square of input length. Multistratal analysis and the presence of optional elements means that the easiest strategy is submitting the whole string (often spanning several hundred symbols from the base alphabet) for parsing, so the fundamental nonlinearity of the algorithm becomes a real issue. Although island parsing techniques (see Carroll 1983) can be used to exploit the fact that the gaps between well-recognized segments are often much shorter, the worst-case scenario is found too often for this to be a generally effective strategy. **Exploiting the fact that strings in the segment lattice are similar** is a nontrivial task in any case, and here the matter is complicated by the requirement of carrying along and **updating the probability/confidence ranking**. Finally, Language Models tend to make heavy use of string distance calculations (which provide the only means of recovering from errors of the recognizer if fixup fails) and the **integration of string distance calculations with context-free parsing** techniques remains problematic.

This state of affairs is to be contrasted to the situation in which left-to-right pruning of the segment lattice is possible. In general, a Language Model as defined in Section 1 is not suitable for left-to-right pruning, because in order to discard some initial segment $s$ we must know not only that $f(s)$ is sufficiently low but also that for every $t, f(st)$ will also be sufficiently low. However, with a Markovian assumption we can estimate how much the overall probability of a string can improve by adding further segments to it, so hypotheses can be discarded in a left-to-right, limited lookahead fashion (typically implemented as dynamic programming). The maintenance/update of probabilities is done at a low level, and the incorporation of string distance calculations is considerably simplified.

Finally it should be emphasized that a conceptually simple method of measuring the relative contributions of the components, based on considerations of mutual information and entropy (Jelinek 1990) is available. In the general case discussed in Section 1, no such model is available, and a second look at our original example will show why. In that example, the recognizer/fixup has exactly one bit of *information deficiency* since getting a single bit of extra information from some oracle would enable us to choose the good candidate string. The output of the LM, however, can have arbitrarily large information deficiency: in the $p = 1$ case it will be correct 75% of the time and will require $O(\log r)$ bits in the remaining 25% of the time, where $r$ is the cardinality of the alphabet $T$. The mutual information between our simple LM and the Language will also depend on the cardinality of the alphabet. **Generalizing the information-theoretic model to the context-free case** remains a challenge.

Now that we have discussed the considerable advantages of Markov models over the parsing/database conception of Language Models we can turn to a discussion of their limitations. The first objection, raised originally in Chomsky (1957), is that the structure of Language is more complex than what can be expressed by a Markovian model. Though this became the accepted wisdom in theoretical linguistics, in the probabilistic framework of Languages the force of the argument is limited inasmuch as the overall frequency of deeply center-embedded constructions is very small compared to underestimation errors coming from other sources. The more practical arguments about **training set size** raised in Miller and Chomsky (1963) retain a great deal of validity, especially when applied to 4th and higher order Markov models, but for most applications third order models, by now well within the capabilities of mid-range workstations and high-end PCs, are sufficient.

A more relevant objection is that **finite-state models do not return any structure of significance**. For information retrieval and other tasks with a considerable semantic component, such intermediary structures as parse trees, dependency graphs, or attribute-value matrices are essential, and if recognition can only return (tagged) strings a more sophisticated parser will have to be run in a subsequent module, often duplicating work, such as dictionary lookup, across the components. However, given that calls to the parser are both expensive and unpredictable (since the same text can be fed into very different parsers depending on the nature of the application) this is not necessarily a drawback. Some data structure mediating between the recognizer and the semantics is necessary anyway, and (tagged) strings provide a natural interface between the two, especially as data in this format, namely (rich) ascii text, is widely available independent of the availability of recognizers.

A closely related problem is the **lack of semantic checking in statistical systems** which, by their nature, tend to be "soft fail" algorithms that will return non-zero probabilities even for ill-formed entries. For purposes like routing mail semantic checking is indeed desirable, but there are other applications (such as documents with legal implications) where scrupulous adherence to the actual input, independent of its semantic coherence, is of great importance. For example, low-level correction of tax returns with errors in the arithmetic would be highly undesirable – what is desirable is letter-perfect recognition with some post-processor that flags such errors. It should be added here that probability values of 1 cause no special problems for the statistical model, so deterministic choices can still be made wherever a closed world assumption is realistic.

To use again our example of address blocks, let us suppose we have a database of (CITY STATE ZIP) triples. We can compile a large network that will contain those and only those strings which make up valid triples, and in fact a great number of database search engines use exactly such networks. What this technology misses is the ability to speed up searches by hashing and other non-local techniques, but it has the advantage of closure under substitution: networks of networks are networks. This means that we can build a network for each city (comprising the alternative spellings of, say, *Winston-Salem*) a network for each state (comprising the alternative spellings *NC, North Carolina, N.C.,* and so on) and a network for each 5-digit number which is a valid zip code, and then link copies of these together in a single large network. **Unparsed querying,** i.e. the ability to integrate queries based on overlapping substrings but directed at different databases, is an unresolved problem outside the finite state domain. To the extent that the basic **segmentation** problem can be kept under control, fast search techniques like hashing offer the promise of outperforming Markovian networks, but the combinatorial explosion discussed at the outset reappears at higher strata as soon as we need to parse the input before we can query the Language Model.

Let me conclude with a point seldom addressed in an engineering context, **language generality**. Most engineers like to maintain that working on English is not only necessary (since that is where the market is) but also sufficient, because "all the problems that occur in natural language also occur, though perhaps in a more limited form, in English". Nothing could be farther from the truth. Much as an anatomist working only with the male of our species would have no clue what nipples are for and could not even guess the existence of an organ like the womb, engineers narrowly focusing on English fail to see the complexity of important problems. I have argued elsewhere (Kornai 1992) that the complexity of the machine translation task is bound from below by the complexity of the morphological analysis task. For English, morphological analysis (removal of *-ing, ed, s*) can be performed in a few dozen lines of code and a few small exception tables. For many other languages inside and outside the Indoeuropean family, this is a task of tremendous complexity from which the issue of dictionary lookup cannot at all be separated. For this reason alone, database technology is unlikely to provide the right tools for Language Modeling.

## 3 References

W.W.Bledsoe and J. Browning: Pattern recognition and reading by machine. Proc EJCC December 1959, pp 225-232.

J.A. Carroll: An island parsing interpreter for the full augmented transition network formalism, Proc ACL First European Conference 1983, pp 101-105.

N. Chomsky: *Syntactic Structures.* Mouton, 1957

R.O. Duda and P. Hart: Experiments in the recognition of handprinted text -II. *Proc FJCC* 1969, pp 1139-1151

F. Jelinek: Self-organized language modeling for speech recognition. In: A. Waibel and K-F Lee (eds) *Readings in speech recognition* Morgan-Kaufmann Publ. 1990, pp 450-506

A. Kornai: Frequency in morphology. In: I. Kenesei (ed) *Approaches to Hungarian* IV, 1992, pp 246-268

A. Markov: Essai d'une recherce statistique sur le texte du roman "Eugène Onéguine". Bulletin de l'Académie Impériale des Sciences de Saint-Petersbrurg VII, 1913

G.A. Miller and N. Chomsky: Finitary models of language users. In: R.D. Luce, R.R. Bush, and E, Galanter (eds) *Handbook of mathematical psychology* Wiley, 1963 pp 419-491

J.H. Munson: Experiments in the recognition of handprinted text -I. *Proc FJCC* 1968, pp 1125-1138