Université de Montréal

Text Prediction For Translators

par

George Foster

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de
Philosophiae Doctor (Ph.D.)
en informatique

mai, 2002

Université de Montréal

Faculté des études supérieures

Cette thèse intitulée:

Text Prediction For Translators

présentée par

George Foster

a été évaluée par un jury composé des personnes suivantes:

Bengio, Yoshua
président-rapporteur

Lapalme, Guy
directeur de recherche

Isabelle, Pierre
codirecteur

Wolfe, Stefan
membre du jury

Vers, Herman
examinateur externe

Da Sylva, Lyne
représentant du doyen

Thèse acceptée le 15 mai 2002

To my loved ones, big and small

RÉSUMÉ

La demande pour les services des traducteurs ne cessant de croître, il y a donc une forte demande pour des outils améliorant leur productivité. Cette thèse propose une approche novatrice leur fournissant un accès interactif à la technologie la plus puissante dans le domaine de la traduction: un système de traduction automatique. Cette nouvelle approche utilise le texte cible de la traduction comme médium d'interaction avec l'ordinateur. Elle est plus naturelle et flexible que les approches actuelles car elle laisse au traducteur l'initiative du processus de traduction, tout en permettant au système de contribuer de façon utile au moment jugé opportun.

Une version simplifiée de cette idée est un système qui tente de prédire le texte cible en temps réel pendant la frappe du traducteur. Ceci peut l'aider en accélérant l'entrée du texte et en suggérant des idées pour la suite. Toutefois, cela peut également le gêner s'il y a trop de suggestions, comme l'ont montré certaines études dans notre laboratoire. Je présente une nouvelle méthode pour la prédiction du texte maximisant explicitement la productivité d'un traducteur en fonction d'un modèle des caractéristiques d'un utilisateur. Mes simulations indiquent une amélioration possible de la productivité d'un traducteur d'environ 10%.

Notre méthode de prédiction est basée sur un modèle statistique estimant la probabilité du texte à venir. Tout en étant précis, il doit être assez efficace pour permettre la recherche en temps réel. Je décris de nouveaux modèles, basés sur le principe de l'entropie maximale, conçus pour équilibrer précision et efficacité de la prédiction. Ils ont amélioré de 50% des modèles de base utilisés lors de travaux précédents.

mots clés: traduction assistée par ordinateur, prédiction du texte, modèles statistiques de traduction

ABSTRACT

Demand for the services of translators is on the increase, and consequently so is the demand for tools to help them improve their productivity. This thesis proposes a novel tool intended to give a translator interactive access to the most powerful translation technology available: a machine translation system. The main new idea is to use the target text being produced as the medium of interaction with the computer. In contrast to previous approaches, this is natural and flexible, placing the translator in full control of the translation process, but giving the tool scope to contribute when it can usefully do so.

A simple version of this idea is a system that tries to predict target text in real time as a translator types. This can aid by speeding typing and suggesting ideas, but it can also hinder by distracting the translator, as previous studies have demonstrated. I present a new method for text prediction that aims explicitly at maximizing a translator's productivity according to a model of user characteristics. Simulations show that this approach has the potential to improve the productivity of an average translator by over 10%.

The core of the text prediction method presented here is the statistical model used to estimate the probability of upcoming text. This must be as accurate as possible, but also efficient enough to support real-time searching. I describe new models based on the technique of maximum entropy that are specifically designed to balance accuracy and efficiency for the prediction application. These outperform equivalent baseline models used in prior work by about 50% according to an empirical measure of predictive accuracy, with no sacrifice in efficiency.

keywords: machine-assisted translation, text prediction, statistical translation models

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Thanks to Marc Dymetman for supplying the roots of many of these ideas long ago, Pierre Isabelle for creating the wonderfully utopic research atmosphere in which they flourished, Elliott Macklovitch for many insights and inspirations, and Guy Lapalme for the incredible energy, perseverance, and dedication with which he stuck to the task of keeping this thesis on the rails.

Thanks also to Philippe Langlais for being a fantastic collaborator on the TransType project, and to all other members of the RALI team for their support.

Finally, thanks to sundry giants. If I have not seen as far as some, it is not because I have not occasionally managed to make it onto the shoulders of the odd giant to enjoy the view for a few moments before losing my balance and tumbling down again.

Chapter 1

INTRODUCTION

> Translation is a fine and exacting art, but there is much about it that is mechanical and routine and, if this were given over to a machine, the productivity of the translator would not only be magnified but his work would become more rewarding, more exciting, more human.
>
> —*Martin Kay*

This thesis is about using text prediction as a tool to assist skilled translators. The idea for such a tool is motivated by a general philosophy of applying AI technology—and in particular machine translation technology—in modest and practical ways, captured in the quote by Martin Kay above and eloquently articulated in his famous tract *The Proper Place of Men and Machines in Language Translation* [69]. The main hallmarks of this philosophy are a division of labour in which the more mechanical aspects of a problem are the ones that get assigned to the computer (instead of the reverse); and a cooperative approach where solutions result from interaction between the human partner and the machine.

Text prediction has both these characteristics. The most accurate predictions will be ones that involve frequently-repeated words or phrases, or entities like numbers and certain proper nouns that go into the target language with little variation. These cases require relatively little thought on a translator's part, and therefore chiefly involve the mechanical labour of typing. Prediction is inherently interactive because it involves observing the translator's progress and periodically making proposals for upcoming text. The translator can respond to a proposal by either accepting it (possibly editing

it afterward) or rejecting it by just continuing to type. This leverages interactivity in a sensible way, allowing the computer to contribute when it can usefully do so, and minimizing the damage when it cannot. Occasionally—and increasingly often as translation technology improves—it will be able to come up with proposals that are less obvious, and thereby take on more of the true cognitive burden of producing a translation.

Creating a viable predictive tool involves two main problems: generating predictions and making them available to a user. Neither are trivial. Generating predictions requires not only the ability to translate (very quickly), but also the ability to anticipate the intentions of a particular translator. Presenting predictions to a user requires an interface that makes proposals easy to take in when they are wanted and unobtrusive when they are not, as well as an efficient mechanism for incorporating them into an existing partial translation.

The research I present here is exclusively concerned with the problem of generating predictions. However, it was carried out in conjunction with the TransType project at the Université de Montréal, which resulted in a prototype text-prediction tool for English to French translation. I have relied on the results of user trials of this tool to define the exact nature of the predictive task, as well as to obtain more realistic assessments of prediction performance. To put my work in context, I begin with a brief description of the TransType prototype; more details are given in the next chapter.

1.1 TransType

The TransType tool provides a graphical interface with basic editing functions, a split-screen view of the current source and target texts, and a pop-up menu containing a short list of word-completion suggestions which is displayed just ahead of the current cursor position. Figure 1.1 shows a screen dump of the interface.

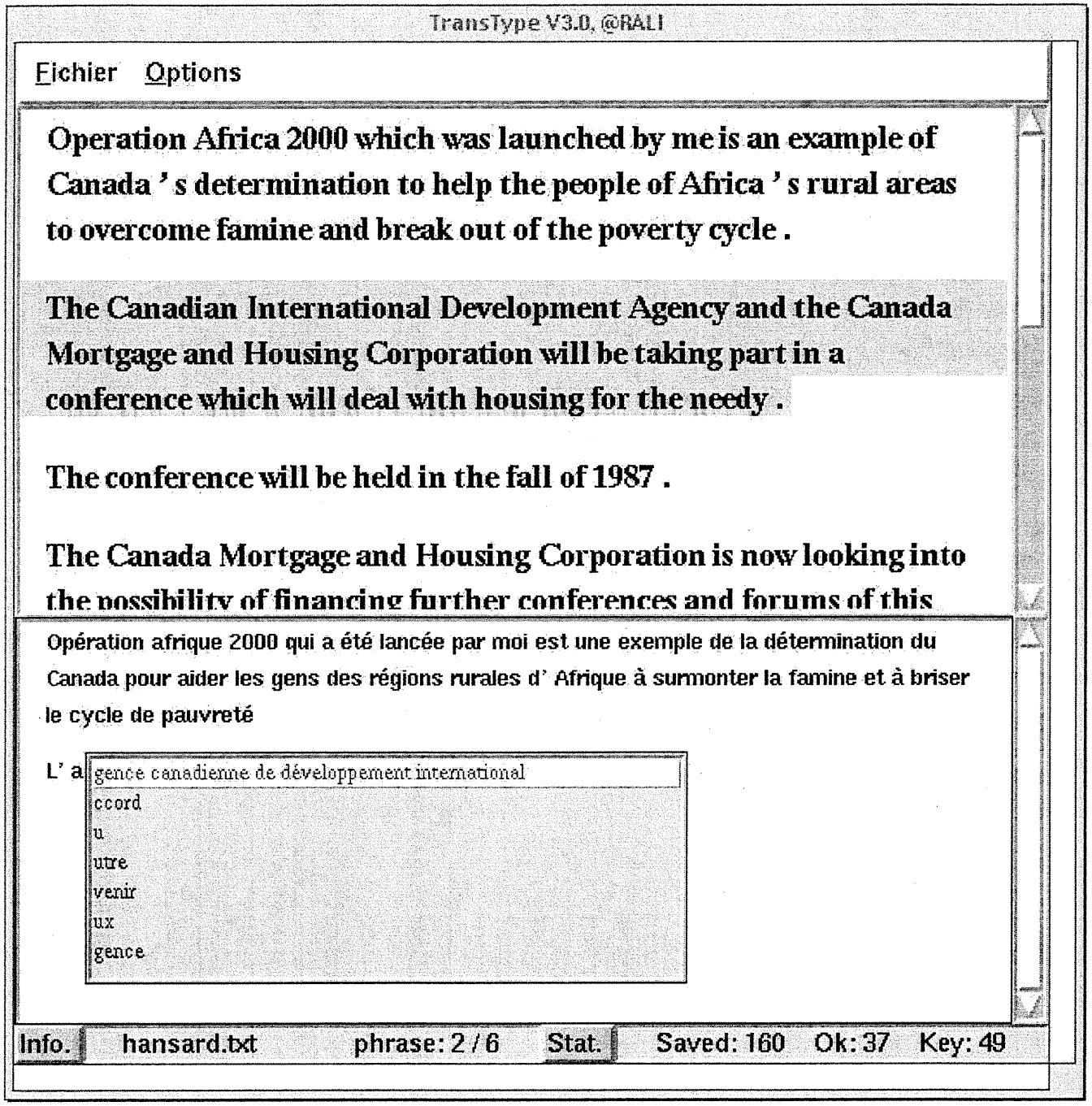

Figure 1.1. Screen dump of the TransType prototype. The source text is shown in the top half of the screen, and the target text is typed in the bottom half, with suggestions given by the menu at the cursor position.

To use the prototype, a translator selects an English source text and begins typing its translation. The current source sentence under translation is highlighted and may be selected using the mouse; it is normally updated sequentially to follow the translator's progress. After each character is typed, the completion menu is moved and its contents are updated. The user may accept the completion at the top of the list by pressing the tab key; other completions may be selected either by clicking on them with the mouse or by scrolling the list until they appear at the top—in either case, the accepted completion is inserted in the translation and the cursor placed at the end of it. Any other keys are interpreted normally, and signal rejection of the current proposal if they cause the current target text to change.

Predictions are generated using statistical models to estimate the probability of the next word in the target text. The probabilities of all words in the vocabulary that match the current prefix (eg, *a* in the example in figure 1.1) are compared, and the top n candidates are proposed to the user, where n is typically 7. To increase the predictive power of the tool, the vocabulary is augmented with multi-word units that are deemed to be translations of segments of the current source sentence.

1.2 The Prediction Task

To facilitate development in parallel with the evolving TransType prototype, I defined and worked on a self-contained prediction task that makes certain basic assumptions about the nature of the user interface, but is otherwise independent of it. The aim was to improve on the prototype in several ways: increasing the accuracy of the statistical models used to generate predictions; extending predictions beyond a single word or lexicalized unit; and defining a realistic model of user behaviour in order to optimize predictions and get an accurate automatic assessment of their value to a translator.

1.2.1 Definition

The prediction task is easy to define, at least superficially: given a source text and some prefix of its translation, make a prediction about what will follow the prefix. As in TransType, I assume that both the current source sentence and some prefix of its translation are known.[1] This allows the predictor to use standard sentence-based MT techniques, and improves its chances of being correct. It supposes that the interface is capable of tracking the translator's progress in order to determine the current source sentence and its (possibly partial) translation. Experience with the TransType prototype suggests that this is feasible, and some commercial translator's workbenches offer this capability as well.

Other similarities with TransType include the fact that predictions are assumed to be made synchronously with respect to the user interface, and that they are made solely on the basis of the available source and target texts. The former is mainly a time restriction, limiting the period available for generating and displaying a proposal to roughly the interval between successive user keystrokes. The latter restriction excludes sources of information, such as a record of which previous proposals have been accepted or rejected, or of response times in various circumstances, that could potentially be useful in adapting the behaviour of the predictor more precisely to the translator's needs.

The main difference from TransType is that predictions can be of arbitrary length rather than just completions for the current word. It is up to the predictor to determine the optimum length in each context, knowing that longer predictions are more valuable if they are correct, but are in general less likely to be correct, take longer to generate, and take longer for the translator to absorb. In contexts where it cannot

[1] I refer to these as sentences for convenience, but there is nothing that prevents the use of small groups of sentences in cases where the translator chooses to depart from a strict one-to-one sentence translation scheme.

confidently make any prediction, the predictor has the option of abstaining, in which case no proposal will be displayed to the user. For concreteness, and in keeping with the sentence-by-sentence translation mode described above, I assume that the scope of predictions is limited to the current target sentence. The consequences of this restriction are not likely to be significant, since proposals of sentence length will take an appreciable time to read and react to in any case. Even if the system correctly predicted every single target sentence, the user would still have to verify each one, so the difference in time saved between predicting the entire target text as one block and predicting it sentence by sentence would be negligible.

A final contrast with the TransType prototype is that, in the problem I have studied, the output from the predictor is a single best-guess proposal rather than a set of alternatives. This simplifies automatic evaluation of the results; and it is also a reasonable strategy for reducing the amount of information the user must absorb in cases where predictions are significantly longer than a single word. It would be fairly straightforward to adapt the techniques I describe below to produce multiple alternatives if desired.

To give a more formal definition and establish some notation, the inputs to the predictor are the current source text S and a prefix H of its translation, along with the current source sentence $\mathbf{s} \in S$ and a prefix $\mathbf{h} \in H$ of its translation. H represents all of the target text typed by the translator so far, and $\mathbf{h}$ will normally be at the end of H, but this will not always be the case—for instance, the translator may decide to back up and revise a previously-translated sentence. The output from the predictor is a proposal $\mathbf{x}$ for what should follow $\mathbf{h}$ in the translation of $\mathbf{s}$. Other quantities of interest are T, the eventual complete translation of S; $\mathbf{t}$, the eventual complete translation of $\mathbf{s}$; and $\mathbf{x}^*$, the portion of $\mathbf{t}$ yet to be typed.[2] All these quantities are character strings. Figure 1.2 gives a labelled example of a prediction.

[2] Mnemonic: s for source, t for target, h for history—the target text before the current cursor position, and x for extension.

s: Let us return to serious matters.

t: $\overbrace{\text{On va r}}^{\mathbf{h}}\overbrace{\textit{evenir aux choses sérieuses.}}^{\mathbf{x}^*}$

x: *evenir à*

Figure 1.2. Example of a prediction for English to French translation. s is the source sentence, h is the part of its translation that has already been typed, $\mathbf{x}^*$ is what the translator wants to type, and x is a prediction. The surrounding contexts S and H from which these sentences are drawn are omitted.

1.2.2 Discussion

Text prediction for translators has obvious similarities to two other problems: unilingual text prediction and machine translation. In this section I draw some parallels and make some distinctions that shed light on the nature of the problem.

Unilingual versus Bilingual Prediction

To what extent is it possible to predict text? In a unilingual context, the answer depends solely on how much redundancy it contains. Redundancy is a necessary feature for communication in noisy environments[3] (ie, typical ones), but it imposes a penalty on users of language that, in the case of typing text, equates to more characters and hence more typing time than is strictly necessary. It is precisely this penalty that unilingual text prediction seeks to remove.

[3] For instance, the last three letters of the word *elephant* are redundant because *eleph* is not a word in English. But dropping them would increase the possibility of confusion in spoken language with the word *elf*. In general, language must strike a balance between the competing demands of reliability of transmission, which leads to increased redundancy; and economy of expression, which leads to increased ambiguity. Parallels can be made to analogous concepts in communication theory, but claims that language represents an optimal balance in any formal sense would be

The notion of redundancy in text can be quantified by thinking of text as resulting from a stationary stochastic process that produces a sequence of characters drawn from a fixed alphabet.[4] The entropy rate of a stochastic process is defined as the average per-character entropy in the limit as the number of characters goes to infinity [39]. It gives a lower bound on the average number of bits required to encode each character, and thus on the average number required to unambiguously specify a text. The entropy rate of English text has been estimated to be at most 1.75 bits per character [25]. Assuming that keys on a standard keyboard produce 7 bits each (somewhat unrealistically, because standard keyboards contain fewer than 128 keys), this means that in principle the raw number of keystrokes needed to produce English texts could be reduced by 75%.

In practice, this kind of reduction could only be obtained by someone capable of effortlessly performing arithmetic coding on a desired text and then converting the resulting bit sequence into a sequence of keystrokes for eventual automatic decoding. Text prediction is obviously a more realistic way to speed up typing, but the gains it offers are significantly lower than the theoretical limit, due to the constrained nature of the "compression" process. Moreover, even if the limit could be approached, there is no guarantee that the result would be a useful tool. In general, a given reduction in the number of keystrokes can be achieved either by making many short predictions or fewer longer ones.[5] Although the number of keystrokes may be identical in both cases, the effort expended by the user is much higher in the former case, because of

difficult to substantiate.

[4] Text is almost certainly not a stationary process, but I will ignore this inconvenience, as have most other authors on the subject.

[5] For example, if a predictor makes predictions of fixed length l which the user can accept with a special key or reject by typing any other key, then the relation is given by $F = P(l-1)+1$, where F is the speedup factor (eg, 2 if the number of keystrokes required is reduced by half), and P is the proportion of accepted predictions. So for a fixed F, P and l are inversely proportional.

the need to stop, verify, and decide to accept or reject a larger number of proposals. Unfortunately, long predictions are *much* harder to make accurately than short ones. So for example, while it seems theoretically possible to reduce keystrokes by about 60% in French text [49] using predictions of just over 5 characters on average, it is clearly impossible to do this with a single prediction that anticipates 60% of a large text.

It is probably for this reason that, despite the theoretical gains it offers, unilingual text prediction is not widely used outside of special domains like aids for the disabled [28, 38, 41], or applications like variable or filename completion that are characterized by a high degree of local repetitiveness. Better interfaces and improvements in language modeling techniques [12] may widen its appeal, but it is far from certain that true text prediction will ever be a standard feature of mainstream word-processing environments.

Bilingual text prediction is both quantitatively and qualitatively different from unilingual prediction. In quantitative terms, using techniques similar to those of [25], I estimate the conditional entropy of French given an English source text to be about 0.76 bits per character.[6] Using the same reasoning as above, this gives an upper bound of almost 90% for the potential reduction in typing effort. This cannot be compared directly to the 75% for unilingual prediction, because of the difference in language and domain, as well as my use of a less careful and almost certainly overoptimistic estimation procedure. However, it does make clear that the use of the source language gives a very significant increase in predictive capability. As a simple example of why this should be so, consider a long sequence of digits or a proper noun in $\mathbf{x}^*$ that also appears in $\mathbf{s}$ but nowhere in H—this is virtually impossible to predict in the unilingual case, but is almost trivial given $\mathbf{s}$. Making longer predictions remains a problem for bilingual prediction, but a much less severe

[6] A cross entropy of 4.14 bits per token given by the best model described in chapter 4 divided by an average of 5.47 characters per token.

one than for unilingual prediction.

Bilingual prediction also has a qualitative advantage over unilingual prediction, because knowledge of the source text gives it the capability to act as more than a pure typing accelerator. If its translation component is good enough, it can in principle take on some of the actual work of translating, just like another translator looking over the shoulder of the first. For example, suppose a specialized term appears in $\mathbf{s}$ that the translator is unfamiliar with. If the predictor knows and can suggest a valid translation for this term, the value to the translator is far more than just the number of keystrokes saved by not having to type it manually.

Text Prediction versus MT

Predicting portions of a target text based on a source text has obvious similarities to the classical problem of machine translation (MT), but there are enough differences to make it a distinct research topic in its own right, and one that may offer hope of more success than has been achieved in MT.

First, the task is not to generate an arbitrary valid translation of the source text as in MT, but rather the particular translation intended by the current user, as inferred from the existing partial translation H. Of course, this does not preclude the system's being able to influence the final translation in cases where the translator's intentions are not yet completely formed. Given a typical sentence-by-sentence translation strategy, it is useful to distinguish two ways of exploiting H. Within the current sentence pair, $\mathbf{h}$ is a strong direct constraint—which is increasingly informative as it gets longer—on the possible contents of the desired suffix $\mathbf{x}^*$. Outside the current sentence pair, previously translated sentences provide much less direct information about $\mathbf{x}^*$, but they can be used by the predictor to adapt its behaviour to the translator's preferred ways of translating S.

A second distinction is that, in contrast to MT, there is no requirement to produce a prediction that completes the current target sentence. In fact, there is no

requirement to produce any prediction at all, because silence is preferable to proposing something that is completely wrong. Thus, while the goal of an MT system is to produce the best possible complete translation, the goal of a prediction system is to produce the longest possible prediction that has a good chance of being correct.

Another characteristic of prediction is that proposals must be generated in real time, possibly as fast as the interval between the successive keystrokes of a proficient typist. This is in contrast to standard MT, which for many applications does not have to work even as fast as a human translator.

A final difference which is important from a research standpoint is that the performance of a predictive system can be evaluated automatically by measuring the degree of similarity between the system's proposals and the corresponding segments of target text in a pre-existing complete translation. This is a much less fair test for an MT system, since the aim of MT is simply to find *some* acceptable translation rather than to anticipate the intentions of a particular translator. (In fact, the evaluation of MT systems is a notoriously difficult problem [6].) Of course, similarity measurements are not a perfect metric for predictive systems either, because they take into account only one user, not the expected performance across all potential users. Also, they rely on an idealized scenario about when the system's proposals would be solicited and accepted, abstracting away from the details and performance of the user interface.

1.3 Making Predictions

What is the best prediction to make in a given context? Since the aim of making predictions is to benefit the translator who is using the predictive tool, a commonsense answer to this question is that it is the one that benefits the translator most. Making this explicit allows the prediction task to be formalized as follows:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmax}}\, B(\mathbf{x}, \mathbf{h}, \mathbf{s}), \tag{1.1}$$

where $\hat{\mathbf{x}}$ is the prediction sought, and $B(\mathbf{x}, \mathbf{h}, \mathbf{s})$ gives the expected benefit of $\mathbf{x}$ in the context $\mathbf{h}$,$\mathbf{s}$.[7] The *actual* benefit to the translator obviously depends on whether or not $\mathbf{x}$ is correct, so the expected benefit must be a function of the probability that this is the case. Taking this one step further, since the translator has the option of editing a partially-correct proposal rather than simply accepting or rejecting it as is, the expected benefit should be a function of the probability that $\mathbf{x}$ will require a given edit cost to transform it into the desired text. The exact edit cost will depend on the user interface as well as the translator's skill in manipulating it. To avoid dependence on these unknown quantities, I make the simplifying assumption that the user will edit only by erasing wrong characters from the end of a proposal. Given a TransType-style interface where acceptance places the cursor at the end of a proposal, this is likely to be the most common editing method, and it gives a conservative estimate of the cost attainable by other methods. With this assumption, the key determinant of edit cost is the length of the correct prefix of $\mathbf{x}$, so the expected benefit can be written as:

$$B(\mathbf{x}, \mathbf{h}, \mathbf{s}) = \sum_{k=0}^{l} p(k|\mathbf{x}, \mathbf{h}, \mathbf{s})\, B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k), \tag{1.2}$$

where $p(k|\mathbf{x}, \mathbf{h}, \mathbf{s})$ is the probability that exactly k characters (no more and no less) from the beginning of $\mathbf{x}$ will be correct, l is the length of $\mathbf{x}$, and $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k)$ is the benefit to the user given that the first k characters of $\mathbf{x}$ are correct.

The approach described by equations (1.1) and (1.2) captures almost all the elements of the tradeoff between longer and shorter predictions discussed in the previous section. In general, for fixed l, $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k)$ will be negative for small values of k, reflecting more time spent reacting to and editing the proposal than would have been required to type the same text manually. On average it will increase monotonically with k, becoming positive at some point if l is large enough. Therefore, as l gets

[7] In general, B will also be a function of H and S, but I leave this implicit in order to simplify the notation.

larger, the expected benefit $B(\mathbf{x}, \mathbf{h}, \mathbf{s})$ will include more and more positive terms, associated with large values of k. However, the magnitude of the negative terms for small k will also increase, reflecting higher cost to erase longer erroneous suffixes. Also, the probabilities $p(k|\mathbf{x}, \mathbf{h}, \mathbf{s})$ will tend to decrease as k gets larger, reflecting decreasing confidence in the accuracy of longer predictions. Eventually there will be a balance point at which the increases from adding terms with high k values will no longer outweigh the decreases in terms with low k values. Where this point occurs depends on the context **h**,**s**, as well as the particular prediction under consideration.

One component of the problem that is not addressed by optimizing benefit is the time taken to generate predictions, which will tend to increase with their length. Rather than trying to build a time penalty into the objective function, I simply measured how times increased with prediction length, up to a maximum length which allowed for convenient testing. Calibrated appropriately for computer speed, these measurements could be used to establish a threshold length to ensure that no prediction took so long to generate as to cause a perceptible delay to the user.

Equations (1.1) and (1.2) effectively break prediction down into three smaller problems: modeling the correct-prefix distribution $p(k|\mathbf{x}, \mathbf{h}, \mathbf{s})$; modeling the user benefit function $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k)$; and searching for the $\hat{\mathbf{x}}$ that maximizes the sum over all k of their products. The following sections give an outline of my approach to each of these problems.

1.3.1 Modeling Text

It would be awkward to model the correct-prefix distribution $p(k|\mathbf{x}, \mathbf{h}, \mathbf{s})$ directly, but fortunately it can be easily calculated from the related distribution $p(\mathbf{x}'|\mathbf{h}, \mathbf{s})$, which is more tractable. This gives the probability that an arbitrary character string **x**' will follow **h** in the translation of **s**. Although strings are necessary for the analysis above, they do not provide a good basis for estimating probabilities, because they ignore the most fundamental structure in text, namely its division into tokens (ie,

occurrences of words). To capitalize on this very useful abstraction, I modeled word-based distributions of the form $p(\mathbf{w}|\mathbf{h},\mathbf{s})$, where $\mathbf{w}$, $\mathbf{h}$, and $\mathbf{s}$ are all assumed to be token sequences.[8] Given certain simplifying assumptions, it is fairly straightforward to calculate the required string probabilities from these.

Joint distributions like $p(\mathbf{w}|\mathbf{h},\mathbf{s})$ are difficult to model directly, so a standard step is to apply the chain rule to obtain a family of conditional distributions:

$$p(\mathbf{w}|\mathbf{h},\mathbf{s}) = \prod_{i=1}^{m} p(w_i|\mathbf{h}, w_1, \ldots w_{i-1}, \mathbf{s}),$$

where w_i is the ith word in $\mathbf{w}$. Although it might be interesting to try to exploit the distinction between $\mathbf{h}$ (which has been sanctioned by the translator) and $w_1, \ldots w_{i-1}$ (which has not), I have chosen to ignore it and consider all the factors on the right hand side to be instances of a basic family of distributions $p(w|\mathbf{h},\mathbf{s})$, the probability that a word w will follow the preceding words $\mathbf{h}$ in the translation of $\mathbf{s}$. This is the central object I have concentrated on modeling.

The probabilities $p(w|\mathbf{h},\mathbf{s})$ are independent of the prediction application, and thus can be learned from examples of translated texts. However, the model itself is not independent of the application, because it must satisfy the requirement of supporting very rapid searches for $\hat{\mathbf{x}}$. Another essential feature is that the "probabilities" it returns must really be probabilities, in order to give accurate values for the expected benefit in equation (1.2). This is in contrast to some NLP applications, for instance certain approaches to text classification, where scores that are proportional to or monotonic with true probabilities are often sufficient. Another characteristic is that the distribution being modeled is actually $p(w|\mathbf{h},\mathbf{s},H,S)$, so as noted above the existence of H, S provides an opportunity to adapt $p(w|\mathbf{h},\mathbf{s})$ to the current text.

All these characteristics distinguish the modeling problem from that of statistical MT, to which it is closely related. In SMT the relative importance of accuracy to

[8] The notation for $\mathbf{h}$ and $\mathbf{s}$ is overloaded; whether these denote strings or token sequences will be indicated where it is not clear from the context.

efficiency is much higher, due to the need to produce complete translations and the lack of interactivity. The classical approach to SMT starts with the "fundamental equation" [23]:

$$p(\mathbf{t}|\mathbf{s}) \propto p(\mathbf{s}|\mathbf{t})p(\mathbf{t}),$$

which scales the conditional probabilities of target-sentence hypotheses by a factor of $p(\mathbf{s})^{-1}$. Finally, because standard MT systems work autonomously, they do not have access to a human-sanctioned partial translation H, S of the current text from which to learn.

Because of these differences, I have developed custom models for $p(w|\mathbf{h}, \mathbf{s})$ rather than deriving this distribution from the models commonly used for SMT. My models are simpler and support much faster searches than the best classical IBM models for SMT (described in appendix B), but they are also less accurate: capable of generating the next few words in the target text with good accuracy but rarely of producing coherent complete translations of longer sentences. The themes I have explored in creating these models include: different methods for combining the two main sources of predictive information $\mathbf{h}$ and $\mathbf{s}$, including the use of maximum entropy modeling techniques; learning translation parameters based on $\mathbf{s}$ in the presence of language parameters based on $\mathbf{h}$; feature selection to reduce the size of the models; and developing translation models without explicit latent variables. I have not considered the question of how to exploit information in H, S to adapt $p(w|\mathbf{h}, \mathbf{s})$.

1.3.2 Modeling The User

In theory, the benefit $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k)$ that a user derives from a prediction whose first k characters are correct might include such intangibles as learning a new way to translate some expression in the source sentence. For practical reasons I interpret benefit much more prosaically as the time savings due to the proposal $\mathbf{x}$, expressing this for convenience in units of average keystrokes. Since the benefit due to a proposal

depends on the user's ultimate decision to accept it or not, $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k)$ is really an expected benefit, and can be expanded as follows:

$$B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k) = \sum_{a \in \{0,1\}} p(a|\mathbf{x}, \mathbf{h}, \mathbf{s}, k)\, B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k, a), \tag{1.3}$$

where a is a random variable that takes on the values 0 for rejection and 1 for acceptance.

The two quantities on the right side of (1.3) completely characterize a user for the purposes of the predictor. The decision to accept or reject a proposal is modeled as probabilistic, which reflects the normal non-determinism of human behaviour. The benefit in the case of acceptance, $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k, 1)$, is the difference between the time it would take to type the first k characters of $\mathbf{x}$ manually and the time taken to move the cursor to the position immediately after its kth character by reading, accepting, and if necessary editing the proposal. The "benefit" of a rejected proposal is the negative of the time it takes for the user to make a decision (conscious or not) to ignore it and keep typing.

These quantities can be expected to vary widely among translators, reflecting differences in typing ability, reading speed, attentiveness, etc. Ideally, models of them would reflect the characteristics and preferences of the current user. Since these are unavailable, I created a single universal model that is intended to characterize a "typical translator". To fit the parameters of this model, I relied on data obtained from user evaluations of the TransType prototype [78], which was instrumented to record and timestamp all user actions. The aim of this exercise was not to build a psycholinguistically precise model (for which there would likely not have been sufficient data in any case), just to come up with a plausible profile to guide the predictor in its choice of proposals. One way to make the system more sensitive to differences among users would be to start with this universal model and adapt it based on observations of the current user's behaviour. As mentioned above, this is outside the scope of my work.

1.3.3 Searching for the Best Prediction

Searching directly through all character strings $\mathbf{x}$ in order to find the one that maximizes the expected benefit to the user according to equation (1.2) would be an expensive proposition. The fact that $B(\mathbf{x}, \mathbf{h}, \mathbf{s})$ is non-monotonic in the length of $\mathbf{x}$ makes it difficult to organize efficient dynamic-programming search techniques or use heuristics to prune partial hypotheses. Because of this, I adopted a fairly radical search strategy that involves first finding the most likely sequence of words of each length, then calculating the benefit of each of these sequences to determine the best proposal. The algorithm is:

1. For each length $m = 1 \ldots M$, find the best word sequence:

$$\hat{\mathbf{w}}_m = \underset{\mathbf{w}_m}{\operatorname{argmax}}\, p(\mathbf{w}_m|\mathbf{h}, \mathbf{s}),$$

   where $\mathbf{w}_m = w_1, \ldots, w_m$.

2. Convert each $\hat{\mathbf{w}}_m$ to a corresponding character string $\hat{\mathbf{x}}_m$.

3. Output $\hat{\mathbf{x}} = \underset{m}{\operatorname{argmax}}\, B(\hat{\mathbf{x}}_m, \mathbf{h}, \mathbf{s})$, or the empty string if all $B(\hat{\mathbf{x}}_m, \mathbf{h}, \mathbf{s})$ are non-positive.

I experimented with various values of M up to a limit which allowed for convenient testing. Step 1 is carried out using a very efficient Viterbi beam search. This makes use of the model $p(w|\mathbf{h}, \mathbf{s})$ in a role similar to the one traditionally played by models in SMT. Step 2 is a trivial deterministic procedure that mainly involves deciding whether or not to introduce blanks between adjacent words (eg yes in the case of *la + vie*, no in the case of *l' + an*). Step 3 involves a straightforward evaluation of m strings, but can be expensive depending on the method used to calculate the required string probabilities $p(\mathbf{x}|\mathbf{h}, \mathbf{s})$ from word-sequence probabilities $p(\mathbf{w}|\mathbf{h}, \mathbf{s})$—recall that the former are required for *all prefixes* of each $\hat{\mathbf{x}}_m$, not just for each $\hat{\mathbf{x}}_m$ as a whole. This step relies on being able to interpret the output of $p(w|\mathbf{h}, \mathbf{s})$ as true probabilities.

Apart from efficiency, the main effect of this algorithm is to relegate the estimation of user benefits to the role of choosing among most-probable word sequences of various lengths. The negative consequences of this are easy to illustrate. Suppose that, in some context, the model narrowly prefers *le* to *mademoiselle* as a candidate for the next word. A benefit calculation would favour *mademoiselle*, because the benefit of *le* is likely to be negative regardless of its probability, but *mademoiselle* is excluded from consideration by the policy of choosing only the most probable sequence of each length (single word in this example). Another problem is that the requirement to have proposals end on word boundaries will force the predictor to generate suboptimal predictions in some cases. For example, suppose the model assigned similar probabilities to different morphological variants of a single French verb. A sensible strategy in this case would be to propose only the verb stem, but this is precluded by the whole-word approach.

On the positive side, having a two-step strategy introduces constraints on $\hat{\mathbf{x}}$ that may help to address some deficiencies in the user model $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k)$. This perspective shows the flip side of the coin for both examples in the previous paragraph. For *le/mademoiselle*, it is quite possible that there may be some cost associated with systematically making longer but slightly less accurate predictions, in terms of user confidence, that is not captured by the model. Similarly, proposals that systematically end on word boundaries may be easier to react to than those that can finish mid-word. Given the fairly large degree of uncertainty about the match between $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k)$ and the characteristics of any particular user, the constraints may represent a useful averaging tendency. In any case, it seems prudent to await more research on interfaces for prediction and how people use them before optimizing too aggressively according to the current user model.

1.3.4 Evaluation

The division of the prediction problem into the separate components of text modeling, user modeling, and search provides the opportunity to evaluate each of these components separately. This is particularly useful in the case of the text model $p(w|\mathbf{h}, \mathbf{s})$, which can be evaluated by measuring its degree of "fit" to a parallel test corpus without the need for expensive search procedures. I used this approach to develop and compare a variety of models independently of the prediction application. Similarly, the Viterbi search in step 1 of the algorithm in the previous section was tuned by comparing the word-sequences produced to those observed in the existing translation, avoiding the need to calculate benefits.

Automatically evaluating the performance of the predictor as whole requires simulating a translator's responses to predictions. A parallel corpus can be used to establish the translation that a user wants to type, but of course contains no information about the actions of the translator who produced it. My solution to this is to use the user model to simulate a translator's responses, accepting probabilistically according to $p(a|\mathbf{x}, \mathbf{h}, \mathbf{s}, k)$ and determining the associated benefit using $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k, a)$. Here k is obtained by comparing $\mathbf{x}$ to the known $\mathbf{x}^*$ that the translator wishes to type. It may seem somewhat artificial to be measuring performance according to the objective function that the predictor is trying to optimize, but this is a biased measure only to the extent that it misrepresents an actual user's characteristics. There are two cases: the user is either a better candidate—types more slowly, reacts more quickly and rationally—than assumed by the model, or a worse one. The predictor will not be optimized in either case, but the simulation will only overestimate the benefit in the second case. By being very conservative in estimating the parameters of the user model, I hope to minimize the number of actual users who would fall into the second category, and thus arrive at a realistic lower bound for the average benefit across all translators.

1.4 Organization

The remaining chapters in this thesis are organized as follows:

- Chapter 2 explores a novel form of interactive machine translation that is a generalization of text prediction. The main hallmark of this approach is the use of the target text as the medium of interaction with the computer. This idea is discussed in detail and compared to more conventional translators' tools.

- Chapter 3 turns to the problem of creating a model for $p(w|\mathbf{h}, \mathbf{s})$ for use in a text prediction system. It first deals with the question of how to effectively combine evidence from $\mathbf{h}$ and $\mathbf{s}$ within a Markov framework, and describes a new model for doing so based on the maximum entropy/minimum divergence technique. This model achieves 50% lower test corpus perplexity than a directly equivalent linear combination. Another contribution is an algorithm for selecting informative bilingual word pairs for use within this model. This chapter is an edited version of [47].

- Chapter 4 improves on the model described in chapter 3 by giving it some information about the progress of the translation so far, in the form of the position of the current word w. This does not entail any sacrifice of its efficient search properties. Two methods are compared, and the best is shown to give almost 50% lower perplexity than the equivalent linear model used in the TransType prototype. This chapter is an edited version of [46].

- Chapter 5 tackles the main task of making predictions, using the models developed in the previous two chapters. This further develops the approach based on maximizing user benefit outlined above, and compares numerous predictor configurations. The best is shown to reduce the time taken by an average translator by almost 13% when the predictor is allowed to consider word sequences

up to five words long. This chapter is an edited version of [50].

- Chapter 6 concludes, lists the contributions of this thesis, and discusses some possibilities for future work.

- Appendix A gives some samples of the simulations used to evaluate the system.

- Appendix B gives a brief overview of statistical MT, focussing on the classical IBM translation models.

- Appendix C presents separate work on maximum entropy/minimum divergence language modeling that supplies some of the technical details missing from earlier chapters.

Chapter 2

TARGET-TEXT MEDIATED INTERACTIVE MACHINE TRANSLATION

Text prediction can be seen as one instance of a more general style of interactive machine translation that uses the target text as the medium of interaction. In this chapter I discuss the merits of this idea in detail, comparing it to existing translator's tools.

As a prelude to doing so, I argue that pure MT is not likely to match human performance in the near future. Obviously, if fully automatic translation *were* just around the corner, research into machine-assisted translation would be much less interesting. Given this basic assumption, the motivation for the proposed tool can be summarized as follows. Current MAT tools for translators are fairly rudimentary, so an interesting possibility is to somehow incorporate an MT component. Various ways of doing this have been tried in the past, but without much success. Many of the problems encountered by these efforts have been due to the nature of the interaction between the translator and the MT system, and can be avoided by making the target text the medium of interaction. An approach to interactive MT based on incremental updates to the target text has the potential to maximize the contributions of the MT system while allowing the translator to work naturally and retain full control of the translation process.

There are a number of different ways in which this general idea could be implemented. One of the simplest is a tool that tries to complete words as a translator types. This was the aim of the TransType project at the Université de Montréal, within which much of the work on this thesis was carried out. At the end of the chapter I give a brief overview of this project.

2.1 Status and Prospects for Machine Translation

2.1.1 History

Machine translation is one of the oldest applications for computers. Its most frequently cited genesis is a memo by Warren Weaver in 1949 [123] that suggested using cryptanalytic techniques developed during the Second World War. Weaver's memo ushered in an era of intense international activity that lasted until about the mid-1960's, when the famous ALPAC report was published in the U.S. This in essence concluded that MT research had been largely a waste of money, and that the goal of writing a computer program capable of matching the performance of a skilled human translator on an arbitrary text—known as *fully automatic high quality translation* (FAHQT)—was extremely ambitious, if not chimerical.

The post-ALPAC period was marked by more modest and pragmatic goals on the part of many of the researchers who continued to work in the field. In a paper that anticipated the conclusions of the ALPAC report [11], Yehoshua Bar-Hillel, an influential early figure in MT, identified three practical uses for MT technology: producing rough translations for the purpose of rapid text scanning (often known as MT for assimilation); translation within restricted domains amenable to formalization by virtue of their simplicity; and translation involving some kind of human input.[1] All three became the focus of attention after ALPAC. MT for assimilation has been largely the province of commercial systems such as SYSTRAN, which has been used by U.S. intelligence services since the 1960's and by the European Union since 1976. The canonical example of successful restricted-domain MT is the METEO system [30], developed at the Université de Montréal and used to translate Canadian weather reports since 1977. Work on combined human/machine translation systems also began

[1] As Isabelle notes [61], these three options "cover the complete range of compromises that can be made with respect to the three attributes of an ideal MT system (quality, generality, full automation)".

in this period, with a notable example being the MIND system [68], developed at the RAND corporation in the late 1960's and early 70's.[2]

A major development in the 1980's, in MT as in many other areas of computational linguistics, was the revival of empirical techniques. These had fallen out of favour in the years after ALPAC due in part to a general turn away from empiricism in linguistics under the influence of Chomsky. Successes in speech recognition [9], as well as increased computer power and growing amounts of text available in electronic form motivated their reintroduction. The earliest instance was example-based MT, proposed by Nagao in 1984 [97]. Although example-based MT received considerable attention and funding, especially in Japan, the greatest excitement and controversy[3] was generated by the statistical approach pioneered by a group of former speech-recognition researchers at IBM [23] in the late 1980's and early 90's (see appendix B for an overview of this). The IBM group produced a French-English system called *Candide*, which in a 1994 ARPA evaluation [7] performed at about the same level as *Pangloss* [51], a state-of-the-art system with a long lineage in traditional MT research.[4] This encouraging result, along with successes in other applications such as part-of-speech tagging [32], led to the widespread adoption of statistical techniques, with the consequence that most research-oriented MT systems now comprise at least some statistical components, though pure statistical systems are still probably in the minority.

---

[2] See Hutchins [58] for a much more detailed account of this and other work in MT up until the early 1980's.

[3] See, for example, the proceedings of the TMI conference in 1992, which had as its theme *Empiricist vs Rationalist Methods in MT.*

[4] Pangloss was actually a "multi-engine" system, which included an example-based component among others. Another system compared in the ARPA test was *LingStat*, a statistical/rule-based hybrid.

2.1.2 Current Status

The most recent epoch in MT has been shaped by the ever-increasing power of PC's, which are now amply capable of running full-fledged MT programs; and by the advent of the internet, whose international nature has created new demand for translations of web content and email. Commercial systems have proliferated to meet this demand, both in the form of PC-based products and on-line services. For the most part this is translation for assimilation, although some commentators see email translation as a new niche ideally suited to MT (or at least, due to its real-time nature, difficult for human translators to take on), and have labelled it translation for interchange [56] or translation for communication [90]. Related developments also driven at least partly by the internet include work in human-assisted machine translation (HAMT) with the aim of permitting monolinguals having knowledge of only a source language [17, 82], or only a target language [105, 128] to obtain improved semi-automatic translations.[5]

Another active area is speech-to-speech translation, which is the ambitious goal of many recent projects, including EuTrans [4], Verbmobil [118], JANUS, and ATR. Most of these projects aim at producing working prototypes, but only within very restricted domains. This vastly improves their chances of success, but as Kay and Hutchins have pointed out [57, 70], it also runs the risk of repeating the problems of the pre-ALPAC era by creating a public perception that realistic open-domain speech translation is just around the corner.[6] In contrast to the active scene in web-related applications and speech translation, fundamental research on traditional FAHQT is currently at a low ebb, with only a very few groups active worldwide. Some work

[5] The complementary field of machine-assisted human translation (MAHT), has also recently seen considerable activity, but this includes very little work on the use of MT itself as a tool for translators, so I postpone a discussion of this topic to the next section.

[6] Recent events seem to have borne this out: in a state of the union address in January 2000, U.S. president Clinton referred to machines which "translate as fast as you can speak". For details and reaction, see the home page of the American Translators Association at `www.atanet.org`.

continues on restricted-domain MT, mostly involving technical documentation produced by large companies; an example is the KANT system developed for Caterpillar at CMU [96].

2.1.3 Prospects for FAHQT

Most work in combined human/machine translation, including this thesis, is predicated on the assumption that unrestricted FAHQT is, if not impossible, at least a very long way off. Experts are almost unanimous about this [5, 57, 61, 70, 86, 95]. The fundamental difficulty is that translation requires the equivalent of human understanding, the ill-defined and elusive grail of artificial intelligence. Bar-Hillel illustrated this convincingly with his famous example *the box was in the pen.* Translating this phrase correctly requires knowing that boxes are usually larger than writing instruments but smaller than fenced enclosures—as well as, in general, the ability to identify contexts in which this default assumption no longer obtains. There are no known techniques, either in MT or any other area of artificial intelligence, which can reliably solve this kind of problem.

This is not to imply that truly hard problems like Bar-Hillel's example are the only remaining barrier to attaining human performance. It is no accident that much of the work in MT up to now has concentrated on syntax, because even this relatively accessible subject is poorly understood. It is easy to see the consequences in state-of-the-art MT systems. The current web-accessible version of SYSTRAN, for instance, renders *love is all you need* as *l'amour est tout que vous avez besoin*, committing an elementary grammatical error that no skilled human translator ever would. Arnold [5] gives a detailed compendium of this and other linguistic problems for MT.

For practical applications, what matters is not the existence of such difficulties, but how often they occur. If the hard cases are rare enough then the errors they cause may be tolerable, particularly if human performance is itself not "perfect"—ie, if there is some level of disagreement even between experts. An example of a

highly non-trivial AI task on which performance has recently become good enough to enable significant commercial success is speech recognition. Outside of certain restricted domains, however, MT is nowhere near the level of skilled human translation, and its commercial potential for taking over tasks normally assigned to professional translators is low.

It is difficult to quantify current MT performance precisely, and even more difficult to try to gauge its future rate of improvement. The quickest progress can be expected to result from the solution of problems which are purely linguistic in nature, since these are the ones which seem easiest and are also the ones which have been studied most extensively. However, although progress has been made and continues to be made in linguistics and computational linguistics, the pace is slow. The profusion of competing theories and formalisms in these fields attests to their continuing immaturity. Moreover, the effects of what progress there has been on the quality of MT seem almost negligible. To quote Hutchins, [57]:

> ...overall it has to be admitted that at present the actual translations produced do not represent major advances on those made by the MT systems of the 1970s. We still see the same errors: wrong pronouns, wrong prepositions, garbled syntax, incorrect choice of terms, plurals instead of singulars, wrong tenses, etc., etc.

Furthermore, the gains to be had from resolving purely linguistic problems depend on what proportion of problems confronted by an MT system are in fact purely linguistic in nature, as opposed to those like Bar-Hillel's example which require wider knowledge and understanding. To a certain extent, the dichotomy between linguistic and extra-linguistic problems is a fiction, because language and understanding are deeply intertwined. It is therefore very difficult to judge how much margin there really is for improvements resulting from better linguistics.

For extra-linguistic problems, the prospects seem bleaker—there has been less

work on them (within MT, at least), and the domain is vast. Some critics, including Bar-Hillel himself, have contended that current rationalist approaches can *never* succeed, because they ignore the dynamic nature of human thought and language use. Melby [95] presents a detailed argument along these lines, with connections to the broader debate in artificial intelligence and philosophy. Empirical systems perhaps offer a ray of hope in this regard, since they are based on learning, and learning (about new contexts) is seen to be a key element missing from the way language is conceived in the traditional symbolic approach. Work on adaptive statistical modeling, eg [73, 106], may constitute a rudimentary step in this direction. But the potential of empirical approaches should not be overstated, because their successes so far have been mostly practical, and there is no hard evidence that they can solve the difficult problems which have defeated rationalist methods. To sum up, in the words of Alan Melby:

> Fully-automatic high-quality machine translation of unrestricted text will be a truly surprising, unpredictable breakthrough and therefore is not expected in the foreseeable future, even though it may come at any time.

2.2 Machine-Assisted Translation

The field of machine-assisted translation (MAT) encompasses many different combinations of human and machine effort for many different translation-related applications. Two themes of MAT in particular are relevant here: tools intended to assist professional or other highly-skilled translators, and efforts to integrate human translation with full-fledged machine translation.

2.2.1 Tools for Translators

Tools for translators are typically made available within the framework of a what is often called a translator's workbench. This is basically a word processor with a

split screen for simultaneous viewing and manipulation of source and target texts. In addition to the features normally found in a unilingual word processor, commercial workstations such as the Eurolang Optimizer [45], Star Transit [112], IBM Translation Manager [113], and Trados TranslatorsWorkbench [114] include specialized features such as source text format preservation, access to bilingual term banks and dictionaries, and functions to facilitate administrative control of documents. More advanced features available on some of these workbenches or currently the subject of research include:

- automatic identification of term candidates in the source text [31, 72, 114];
- automatic translation of terms [52, 72, 113, 114];
- translation memories which allow access to previously-translated text [44, 45, 72, 99, 102, 112–114];
- the possibility of running an MT system on the text and editing its output [45, 52, 72, 99, 113, 114]; and
- verification of finished translations for correctness or completeness. [87, 92]

Translation Memory

Of these tools, translation memory is the most highly touted. A translation memory is a database containing *aligned* previously-translated texts. Alignment [107] is the process of putting segments in a source text into one-to-one correspondence with segments in a target text. The segments involved are normally no smaller than a sentence. A translation memory can be searched for text in either or both of the two languages it contains, and typically returns results in the form of bilingual segment pairs.

The simplest way to use a translation memory is as a bilingual concordance [84] to allow the translator to search for previous translations of a troublesome word or phrase.

Another use is motivated by the fact that many translation jobs involve texts which are modified versions of others that have been translated in the past. (This is particularly true of commercial documentation, which probably constitutes the bulk of material submitted to professional translators). Much of the previous translation can simply be copied in this case, but it is not always possible to locate the previous text, nor is it easy to identify the portions that are identical. To help with this problem, some translation memories include a component that automatically finds matches for segments of the current source text among previous material. Whenever a match is found, the corresponding target text segment is retrieved and proposed to the translator. Since "matching" segments of text may have slight variations, such as numbers or dates that have changed, fuzzy algorithms are sometimes used to identify matches, eg [72, 114].

The *active* translation memory described in the previous paragraph constitutes a simple form of machine translation. Although very crude, it has several advantages over the alternate approach of running a full-fledged MT system and editing its output: it is more efficient; it is adaptive, because matches can be sought in recently-translated text; and the fluency of the target text is guaranteed because it has been manually translated.

However, unlike MT, translation memories do not have the ability to translate arbitrary passages of text (see [89] for an analysis of this limitation). Apart from the special situation of repeat texts, their usefulness is therefore limited to looking up words or short expressions. Another limitation, which they share with all current translator's tools, is that they make only one take-it-or-leave-it proposal, leaving it up to the translator to bridge whatever gap exists between this and the desired translation.

2.2.2 Integrating Human and Machine Translation

Machine translation, which is the pinnacle of translation technology, has been contemplated for use as a translator's tool since its earliest days. Efforts to integrate human and machine translation can be categorized according to when the human contribution takes place: before, after, or during the MT system's operation; these are known as *pre-editing*, *post-editing*, and *interactive machine translation (IMT)*.

Pre-editing

In a pre-editing system, the user annotates the source text so as to render it less ambiguous for the MT system, indicating for example the correct senses of words which are homographs, the objects to which pronouns refer, the words modified by prepositional phrases, etc. Once this is finished, the user's contribution ends and the MT system takes over. This is a very old approach which does not seem to be the subject of any current research. The reason it has not been more actively pursued seems fairly obvious: there can be few applications apart from research which justify requiring the user to perform the onerous task of making a detailed linguistic analysis of a text and then encoding the result in some machine-interpretable formalism. Just translating the source text would be easier, for anyone with the capability to do so. However, it is worth noting the similarity between pre-editing and controlled-language translation [100], in which source texts are constrained *during their creation* in such a way as to make them easier to translate.

Post-editing

In post-editing, the translator's role is simply to correct the output from an MT system. It is well established that in order for post-editing to be cost-effective the MT output must be close to the desired quality [85, 94]. When high quality output is desired this is rarely the case, so it is usually faster for a translator to ignore the MT

output and begin again from scratch. As a method of assisting a skilled translator, post-editing can be summarized as follows: it is relatively easy to implement and does no harm (because the translator is free to ignore the MT system's contribution), but it is not usually much help either. The simplicity and low risk associated with post-editing help explain why it is the strategy of choice in current commercial workbenches.

Interactive Machine Translation

In interactive translation, the task of the translator is to guide an MT system to an acceptable solution through a series of timely interventions. The premise is that if the interventions can be made at just the right places—just when the MT system is about to make errors—they will keep the system on track and maximize the amount of useful work it can do. If the system is good enough, the effort invested in the interventions can be less than would be required for manual translation.

IMT has a fairly long history, beginning with Kay's MIND system [68] in the early seventies. The MIND system was a general environment for linguistic processing which included a component called the *disambiguator* whose purpose was to remove parsing ambiguities by asking questions of a human user. The MIND system could be configured for MT, with the disambiguator in place to improve its analysis of the source text; once analysis was complete, no further user intervention was solicited, and the target text was generated completely automatically.[7]

Most subsequent IMT systems have adopted the MIND approach, which can be characterized as *interactive pre-editing* [124]. Research has tended to concentrate on refining the question and answer process through techniques such as ordering the

[7] The reason for this source/target asymmetry is perhaps in part because of the difference between analysis and generation in translation. In theory, a system must be capable of analyzing any valid text in the source language, but it does not necessarily have to be capable of generating all valid texts in the target language.

questions so that likely answers to early questions will make later ones unnecessary [18, 27, 91]; presenting multiple-choice responses, with the system's best hypothesis first [27, 91]; finding more natural formulations such as in terms of paraphrases of the source text [19, 110, 127]; and tailoring questions to the user's level of sophistication [18].

From the viewpoint of a skilled translator, the problems with this style of IMT are similar to the problems with pure pre-editing. First, since making an explicit analysis of the source text is not part of a translator's normal routine, the question and answer process represents a burden, no matter how streamlined it becomes. If this burden is too great, manual translation will be more efficient. Second, and more seriously, the MT system has no human guidance when generating the target text. For applications where high-quality translations are required, this means that the target text will have to be manually revised.

For these reasons, interactive pre-editing is usually promoted as *personal* IMT for people with limited knowledge of the target language [55, 111], or for monolingual authors of documents [20], especially in situations when output quality is not of paramount importance or translation into multiple target languages is required. Two exceptions are the *ALPS* and *Weidner* systems (both described in [58]), which are specifically intended for professional translators. Recent published descriptions of these systems are difficult to obtain, but an evaluation of the Weidner system by Elliott Macklovitch in 1985 [85] found that its use actually *decreased* the productivity of professional translators working in the Canadian government's translation bureau. Neither system seems to be the focus of any recent research, nor does either appear to be currently marketed as a tool for translators.[8]

One alternate approach to IMT has recently been proposed by Yamabana et al

[8] ALPS is said to be used in-house in the ALPNET translation service, and the Weidner system is marketed as a complete MT system under the name *Transcend* [88].

[126].[9] Their idea is to translate each source sentence "bottom up", in a series of steps which the user can control. In the first step the system displays a version of the sentence in which content words have been replaced by their translations, and allows the user to select alternate translations for these words from a menu. In subsequent steps, the system translates increasingly larger segments of the sentence into the target language. Before each step, the user can change the system's choice of the next segment to be translated (effectively providing a bottom-up bracketing of the text). After each step, the user can pick an alternate translation from a menu if desired. This system is not contemplated for use by skilled translators, but the authors make a good case for its use by Japanese speakers with some knowledge of English hoping to produce reasonable-quality English text for email messages and the like.

2.3 Target-Text Mediated IMT

IMT is potentially a very powerful tool for translators. In my opinion, its lack of success so far is due to the inappropriateness of interactive pre-editing for skilled translators, rather than any inherent flaw in the idea of IMT itself. To recapitulate and expand slightly on the discussion in the previous section, the problems with interactive pre-editing are:

1. The task of *explicitly* disambiguating elements of the source text is not a normal part of translation, and therefore represents an additional burden on the translator.

2. The translator's input only makes a difference in cases where the system would have made an error on its own. But since there is no reliable way of detecting

[9] This appears to be an independent re-invention of the approach taken by the CULT system [83] in the 1970's, which had not previously generated any similar work.

these cases a priori, the system must err on the side of caution and ask about a large number of potential problems. Therefore the translator does more work than is really necessary.

3. The translator has no direct control over the target text. A post-editing step is therefore necessary if high-quality output is desired.

The third point is the most serious. In theory it could be addressed by extending the scope of the system's questions to include various features of the target text, such as lexical choice. But this would tend to aggravate the first and second points. It is also not obvious how to frame questions about certain aspects of the target text, for instance its grammatical structure, in a way that a translator could work with easily.

2.3.1 Using the Target Text as the Medium of Interaction

A new approach to IMT which avoids many of the problems of interactive pre-editing is to use the target text as the medium of interaction, and have the translator and the system make alternating contributions to it. In this approach, the translator's inputs serve as progressively more informative constraints for the MT component, which normally responds to each of them with a fresh proposal for some part of the target text. This can be seen as a kind of interactive post-editing; following a previous publication [48], I will refer to it as *target-text mediated* (TTM) IMT. In contrast to other interactive methods, where the translator's interventions occur *during* the MT process, in TTM the interventions occur after discrete MT steps in an iterative procedure. Another major difference is that the MT component does not produce a completely free translation, but one which respects the constraints on the target text arising from the translator's previous contributions.

TTM directly addresses all three of the problems listed in the previous section. First, the extra burden on the translator is limited to having to read and evaluate the system's proposals for parts of the target text. Since the target text is the natural

focus of the translator's attention, this is not as onerous as having to answer questions about the source text. Second, the translator does not have to waste time correcting false mistakes, because it is immediately apparent from the system's proposal whether or not it has made an error. Finally, the translator has direct explicit control over the final form of the target text.

There are a number of ways in which this style of IMT can potentially help a translator:

- when the translator has a particular translation in mind, the system can speed the typing or editing process by anticipating parts of it;
- when the translator is searching for an appropriate translation, the system's contributions can provide ideas; and
- when the translator makes a mistake, the system can provide an immediate warning, either implicitly by failing to generate a proposal at the point where the error occurs, or explicitly by attaching an annotation to the text.

The main potential drawback to this approach is that the system's contributions might prove distracting to the translator, especially if they are often wrong or if they are presented in a clumsy way. A more subtle danger is that the use of TTM could degrade the quality of the resulting translation if the translator occasionally accepts flawed suggestions from the MT component. Clearly, the MT component must be reasonably accurate and the user interface must be carefully designed in order to avoid pitfalls like this. As long as this is the case, the risks associated with TTM are low, because, as in passive post-editing, the translator is always free to ignore the system's contributions.

2.3.2 Prediction Versus Revision

It is useful to distinguish between two basic modes of operation in TTM: prediction, where the user is typing a new target text and the system's contributions are guesses about what will be typed next; and revision, where the user is making corrections to an existing piece of text and the system suggests appropriate modifications.

Prediction

Text prediction for translators was first suggested by Church and Hovy [33] as one of many possible applications for "crummy" machine translation technology. It is a relatively straightforward problem to define: the target text to the left of the cursor is considered fixed, and the system's task is to propose an extension to it.

The system's contributions can vary in how much text is predicted and how it is presented to the user. The ideal length for a prediction will depend on several factors, including how fast the MT component works, how quickly predictive accuracy degrades as a function of distance from the known prefix, and the preferences of the current user. For ergonomic reasons, it may be better to have predictions end on word boundaries. Also, as mentioned in the previous chapter, making them longer than a typical sentence would probably be unrealistic, both in terms of the amount of text the MT component can reliably predict and the amount the user can quickly digest.

Options for displaying a proposal to the translator include putting it directly in front of the cursor, putting it in a separate window of its own, and listing several alternate possibilities in a menu. To reduce the burden on the user, menus should probably be kept short, both in terms of the number of alternatives and the length of each alternative. Another important decision is when to display proposals; possibilities include: after every user keystroke, when the user pauses for a certain length of time, when the system judges they are likely to be right, or only on demand.

After a proposal has been displayed, the translator has the option of either accepting it (for instance by using a special keystroke or mouse command), or ignoring it and continuing to type. In both cases, the system would normally calculate a new proposal immediately after the user's action. Of course, it would also be possible for the translator to accept only part of a proposal or move the cursor back into previously-typed text, in which case the system could switch to revision mode.

Revision

Revision is somewhat more widely applicable than prediction. It could be used in conjunction with prediction in the normal course of producing a translated document, but it could also be used to edit a draft text from a different source—for instance another translator, an MT system, a translator's memory, or a translator's dictation system [21]. In revision, the system's task is to propose a replacement for an existing segment of target text, while considering the text on either side of the segment (if any) to be fixed.

The user must have some way of designating the scope of the desired change. One possibility would be to give the system only the current cursor position, and have it first modify the word under the cursor, then automatically expand the scope outward as necessary around the word—perhaps only in one selected direction—until the results were consistent with the surrounding text. For example, a change to the grammatical number of a noun under the cursor might necessitate a corresponding change to a nearby verb. A less sophisticated but probably more workable alternative would be to have the translator explicitly designate the entire scope of the change, for instance by highlighting a certain region of text.

In the absence of any information about the nature of the desired change, the system's task would be to propose what it deemed a plausible alternative to the selected text. Its performance could be improved by letting the translator give it hints about the problem that needed to be fixed. These might include the suppression

or inclusion of certain words within the segment to be changed, or changes to global properties such as tense, voice, degree of verboseness, etc. Such capabilities would obviously require considerable sophistication on the part of both the MT component and the user interface for the TTM tool.

2.4 TransType

The first working prototype of a TTM system was developed as part of the TransType project, carried out by the RALI group at the Université de Montréal from 1997 to 2000. The TransType prototype tries to complete words and other short lexical units as the translator types. It accepts arbitrary English texts as input, and is capable of correctly anticipating about 70% of the characters in a freely-typed French translation. It consists of two major components: a graphical user interface and a prediction engine for generating completions. The remainder of this section gives a brief overview of the prototype and its performance, as well as the main research carried out during the project. For clarity, the contributions of this thesis are excluded from this description.

2.4.1 Graphical User Interface

TransType's user interface is implemented in TCL/TK. It provides basic editing functions, a split-screen view of the current source and target texts, and a pop-up menu containing a short list of word-completion suggestions which is displayed just ahead of the current cursor position. Figure 1.1 shows a screen dump of the interface.

To use the prototype, a translator selects an English source text and begins typing its translation. The current source sentence under translation is highlighted and may be selected using the mouse; it is normally updated sequentially to follow the translator's progress. After each character is typed, the completion menu is moved and its contents are updated. The user may accept the completion at the top of the

list by pressing the tab key; other completions may be selected either by clicking on them with the mouse or by scrolling the list until they appear at the top.

The interface has many options which can be set dynamically, for instance the number of suggestions which appear in the completion menu, and the minimal length for suggestions. It also writes extensive tracking information to log files, for the purpose of assessing user behaviour, timing, and completion performance. For details, see [79].

2.4.2 Prediction Engine

The prediction engine is implemented in C and C++. It takes as input a source text and a target-text prefix, and outputs a ranked list of completions for the last partial word in the prefix. For efficiency, the source and target texts are not specified from scratch, but rather through a series of incremental updates that mirror the translator's progress—for instance, set the source sentence, append a target word, append a target character, etc.

The prediction engine consists of two main parts: a statistical model for assigning probabilities to candidates for the next word, given the current source text and target-text prefix; and a search procedure to find the candidates with the highest probability according to the model.

Statistical Models

The statistical model returns an estimate of the probability that a given word will follow the current source text and target-text prefix. The baseline model for the project, described in [49], was a fixed linear combination of a standard trigram language model and the IBM translation model 2. The trigram language model [67] predicts a word from the two preceding target-text words. The IBM translation model 2 [23] predicts a word on the basis of its one-to-one translation associations with each of the words

in the source text, as well as its position relative to each of these words. The version of model 2 used in TransType is slightly modified to account for words like proper nouns that translate mostly unchanged from English into French.

The models were trained on the Hansard corpus of Canadian parliamentary proceedings, which are published in English and French versions. Bilingual sentence pairs were identified automatically using the method described in [107]. Approximately 50M words in each language were used for parameter estimation.

Research in the project focussed mainly on improving the baseline model in a variety of ways. A major weakness of this model is that its two components, the language and translation models, are combined linearly with fixed weights. Intuitively, the weight on each component should be allowed to vary with context. For example, the word *président* almost invariably follows the frequent bigram *monsieur le* in the Hansard corpus. When *monsieur le* has just been observed, therefore, the trigram should be given a high weight to reflect higher confidence in its prediction. A number of ways for identifying such contexts and assigning appropriate weights were investigated in [75], but none was found to significantly improve the accuracy of the baseline model.

Another major weakness of the baseline model is that its translation component (model 2) directly captures only word-for-word translation associations, and its knowledge of grammatical relations is limited to a notion that words in similar positions within the source and target sentences tend to be mutual translations. Among other things, this means that the model has difficulty recognizing translation relations between groups of words, especially those in which translation is non-compositional, such as *red tape / paperasserie*. One way to remedy this deficiency would be to use more sophisticated translation models such as the IBM models 3–5. A simpler way is to add selected sequences of words (or *units*) to the lexicons in both languages. For TransType, this has the advantage of allowing multi-word extensions to be proposed to the user at little extra computational cost compared to pure word completion.

Experiments involving translation units demonstrated that they yield a modest improvement over the baseline model [76, 77]. Another valuable way of using units is to be able to rapidly incorporate a term lexicon for some specialized domain into the completion engine.

Search

Conceptually, the search procedure iterates through a vocabulary, submits each word that matches the current word prefix (ie, the partial word typed by the user, which may be empty) to the model, and outputs the set of words with highest probabilities. Since the French vocabulary is very large, however (on the order of 500k inflected forms), the search is initially limited to words which exceed some threshold probability of appearing in a translation of the current source sentence. This "active vocabulary" is generated by the IBM translation model 1. Only if the active vocabulary contains no words that match the current word prefix does the search "break through" into the main vocabulary. The dictionary is stored as a trie to make the process of matching word prefixes more efficient. Further details are given in [49].

2.4.3 Evaluation

The TransType prototype was evaluated in two separate trials using skilled translators as test subjects [78, 80]. The results indicate that theoretical estimates of benefit based on maximum possible characters saved are very optimistic and do not necessarily reflect gains in productivity. Although the prototype was theoretically capable of eliminating about 70% of keystrokes on the test texts used (according to the model of user interaction described in [49]), the actual keystroke savings were only about 45% on average for the first trial and 30% for the second. The discrepancy was due to the fact that the translators did not always accept correct completions as early as possible, as was assumed by the theoretical calculation. Even worse, productivity

(measured as the rate of text production) declined as a result of using the prototype tool for all but one subject; the average loss was 35% in the first trial and 17% in the second. The main reason for this appears to be that it takes time for users to stop typing and read proposals, so if too many proposals are wrong or too short, the net effect will be to slow the translator down. On the positive side, the vast majority of translators (9/10 on the first trial and 9/9 on the second) liked TransType, and most felt that it had helped rather than hindered them.

There are several reasons to believe that the large drops in productivity observed during these trials may not reflect the true potential of the tool. First, the learning curve appears to be quite steep and it is not clear that the time allotted for the experiments was enough to let translators reach their maximum productivity. This is supported by the fact that average productivity in the second trial is higher (with a loss of only 10%) if the results from the first ten minutes are not counted.[10] Second, the latency time for reading and reacting to a menu of completion suggestions was approximately the time it took to type 2–3 characters, indicating that suggestions of three or fewer characters are likely to simply slow the translator down. Despite this, many such completions were proposed and accepted—so simply eliminating these suggestions might improve productivity. Related changes to the interface might help as well, for instance using shorter menus, or displaying proposals only on demand.

[10] This average includes a final phase of the experiment which involved a non-Hansard text and some handcrafted units added to the lexicon. If this phase is excluded, the productivity drop is only 3%. Unfortunately, however, this is still not high enough to exclude the possibility that the translators were simply learning to ignore the completions altogether!

Chapter 3

AN EFFICIENT TRANSLATION MODEL

This chapter deals with the problem of creating a model for $p(w|\mathbf{h}, \mathbf{s})$ that is efficient enough to support real-time text prediction, yet reasonably powerful. An initial challenge in doing so is finding a way to combine the two sources of conditioning information in this distribution. Two methods are compared: a straightforward linear combination, and a novel maximum entropy/minimum divergence (MEMD) combination. The MEMD combination is found to improve test corpus perplexity by 50%.

Another way of speeding up a translation model is to reduce the number of word-pair features it contains. Three methods for doing this are compared, and a simple technique based on the information gain according to the classical IBM model 1 is found to give very good results.

3.1 Introduction

As described in chapter 1, the distribution I have concentrated on modeling for text prediction is $p(w|\mathbf{h}, \mathbf{s})$, the probability that a word w will follow the previous words $\mathbf{h}$ in the translation of a source sentence $\mathbf{s}$. This supports TransType-style word completion trivially, as a search through the target vocabulary to find the most probable next word. To be able to perform very efficient real-time searches for multi-word predictions, as described in chapter 5, I require that the model have the Markov property. This means that the probability of w must depend only on some finite-length suffix of $\mathbf{h}$; in practical terms, this suffix should be at most a few words long.

An immediately apparent difficulty in modeling $p(w|\mathbf{h}, \mathbf{s})$ is that it is conditioned

on two very disparate sources of information. A good model should combine predictions from the source text and the preceding target text in a complementary way, but it is not obvious how this can be achieved. One simple strategy is to use a linear combination of language and translation components, of the form:

$$p(w|\mathbf{h},\mathbf{s}) = \lambda p(w|\mathbf{h}) + (1-\lambda)p(w|\mathbf{s}), \tag{3.1}$$

where $\lambda \in [0,1]$ is a combining weight. However, this is a weak model because it averages over the relative strengths of its components; when $p(w|\mathbf{h})$ is likely to be a more accurate prediction than $p(w|\mathbf{s})$, it is obvious that the model should rely more heavily on $p(w|\mathbf{h})$, and vice versa, rather than using a fixed weight. In theory this could be partially remedied by making λ depend on $\mathbf{h}$ and $\mathbf{s}$, but in practice significant improvements with this technique have proven elusive [75].

An alternate approach would be to use Bayes law to reformulate as follows:

$$p(w|\mathbf{h},\mathbf{s}) = \frac{p(\mathbf{s}|\mathbf{h},w)p(w|\mathbf{h})}{p(\mathbf{s}|\mathbf{h})}. \tag{3.2}$$

This separates out the language component $p(w|\mathbf{h})$ from the translation component $p(\mathbf{s}|\mathbf{h},w)/p(\mathbf{s}|\mathbf{h})$, and combines the two in an optimum way. However, it is not obvious how to model the translation component, particularly in such a way as to preserve the desired Markov property for the overall model. For instance, one idea would be to apply the simple IBM1 translation model described below, with $(\mathbf{h},w)$ in the role of the source text. But then the probabilities of $\mathbf{s}$ would become products of sums over *all* words in $\mathbf{h}$, clearly violating the Markov assumption. The approach captured by (3.2) is analogous to the standard noisy-channel approach in SMT:

$$\begin{aligned} p(\mathbf{t}|\mathbf{s}) &= p(\mathbf{t})p(\mathbf{s}|\mathbf{t}) / \sum_{\mathbf{t}} p(\mathbf{t})p(\mathbf{s}|\mathbf{t}) \\ &\propto p(\mathbf{t})p(\mathbf{s}|\mathbf{t}). \end{aligned}$$

where proportionality holds when searching for the optimum target text $\mathbf{t}$ for a given source text $\mathbf{s}$. This problem is known to be NP-complete [71]—even when $p(\mathbf{t})$ is

modeled by a uniform distribution that makes no distinction between different target sequences of the same length, and when $p(\mathbf{s}|\mathbf{t})$ is modeled using IBM1. The difficulty of finding good suboptimal search techniques is attested to by the large volume of recent research on the topic, eg [53, 54, 98, 101, 108, 120].

A statistical modeling technique that has recently become popular for NLP applications is maximum entropy/minimum divergence (MEMD) modeling [15]. Like linear combination, MEMD makes it easy to incorporate information from different sources, so it is a natural choice for modeling $p(w|\mathbf{h}, \mathbf{s})$. Unlike linear combination, MEMD is a principled approach that does not rely on independence assumptions. For instance, the independence assumptions in the linear model described above are made explicit in the following derivation in terms of a Boolean selector variable L that determines whether language or translation information will be used to predict w:

$$\begin{aligned} p(w|\mathbf{h}, \mathbf{s}) &= p(w, L|\mathbf{h}, \mathbf{s}) + p(w, \bar{L}|\mathbf{h}, \mathbf{s}) \\ &= p(L|\mathbf{h}, \mathbf{s})p(w|L, \mathbf{h}, \mathbf{s}) + p(\bar{L}|\mathbf{h}, \mathbf{s})p(w|\bar{L}, \mathbf{h}, \mathbf{s}) \\ &\approx \lambda p(w|\mathbf{h}) + (1 - \lambda)p(w|\mathbf{s}). \end{aligned}$$

Here the assumptions are that L is independent of both $\mathbf{h}$ and $\mathbf{s}$, and that w is independent of $\mathbf{h}$ (resp $\mathbf{s}$), given L (resp $\bar{L}$).

In this chapter, I describe a highly efficient MEMD model for $p(w|\mathbf{h}, \mathbf{s})$, and compare its performance to that of an equivalent linear model, demonstrating that the MEMD model gives greatly superior results in terms of accuracy. To my knowledge, this is the first application of MEMD to building a large-scale translation model, and one of the few direct comparisons between a MEMD model and an almost exactly equivalent linear model.[1] I also compare several different techniques for MEMD feature selection, including a new algorithm due to Printz [103].

[1] Rosenfeld [106] reports a greater perplexity reduction (23% versus 10%) over a baseline trigram language model due to long-distance word-pair predictions in a maximum entropy (ME) framework

3.2 Models

In this section I describe the two models to be compared.

3.2.1 Linear Model

The linear model I used as a baseline is a combination as in (3.1) of an interpolated trigram model [64] for $p(w|\mathbf{h})$ and the IBM model 1 (IBM1) [23] for $p(w|\mathbf{s})$.

The interpolated trigram is of the form:

$$\begin{aligned} p(w|\mathbf{h}) &= \phi_3(w'', w')\tilde{p}_3(w|w'', w') + \phi_2(w'', w')\tilde{p}_2(w|w') + \\ &= \phi_1(w'', w')\tilde{p}_1(w) + \phi_0(w'', w')p_0(w), \end{aligned}$$

where $\tilde{p}_i()$ is the empirical (relative frequency) distribution over i-grams for $i > 0$; $p_0(w)$ is a uniform distribution over all words in the vocabulary; and ϕ_i is the weight associated with the ith distribution when w'', w' are the last two words in $\mathbf{h}$, under the constraint $\sum_{i=0}^{3} \phi_i(w'', w') = 1$. Following standard practice, I let the weights depend on the frequency of the conditioning bigram, ie $\phi(w'', w') = \phi(\text{freq}(w'', w'))$, so there is one set of weights for each distinct bigram frequency in the training corpus.[2] Values for the weights were estimated using the EM algorithm from a portion of the training corpus distinct from that used to define the empirical i-gram distributions. See appendix C for more details on this process.

IBM1, as originally formulated (see appendix B), models the distribution $p(\mathbf{t}|\mathbf{s})$. However, since target text tokens are modeled as conditionally independent given $\mathbf{s}$, it can also be used for $p(w|\mathbf{s})$. The underlying generative process is as follows: 1)

compared with the same predictions incorporated using linear interpolation. However, since the two models tested apparently differed in other aspects, it is hard to determine how much of this gain can be attributed to the use of ME.

[2] There are better ways of of constructing smoothed ngram models [29], but the interpolated trigram gives good results and is easy to implement.

pick a token s at random in $\mathbf{s}$, independent of the positions of w and s; 2) choose w according to a word-for-word translation probability $p(w|s)$. Summing over all choices for s gives the complete model:

$$p(w|\mathbf{s}) = \sum_{j=0}^{J} p(w|s_j)/(J+1)$$

where J is the length of $\mathbf{s}$, s_j is the jth token in $\mathbf{s}$ for $j > 0$, and s_0 is a special null token prepended to each source sentence to account for target words with no obvious translation in $\mathbf{s}$. The translation parameters $p(w|s)$ can be estimated from a bilingual corpus of aligned sentence pairs using the EM algorithm, as described in [23].[3]

3.2.2 MEMD Model

A MEMD model for $p(w|\mathbf{h}, \mathbf{s})$ has the general form:

$$p(w|\mathbf{h}, \mathbf{s}) = q(w|\mathbf{h}, \mathbf{s}) \exp(\vec{\alpha} \cdot \mathbf{f}(w, \mathbf{h}, \mathbf{s}))/Z(\mathbf{h}, \mathbf{s}),$$

where $q(w|\mathbf{h}, \mathbf{s})$ is a reference distribution, $\mathbf{f}(w, \mathbf{h}, \mathbf{s})$ maps $(w, \mathbf{h}, \mathbf{s})$ into an n-dimensional *feature vector*, $\vec{\alpha}$ is a vector of feature weights (the parameters of the model), and $Z(\mathbf{h}, \mathbf{s}) = \sum_w q(w|\mathbf{h}, \mathbf{s}) \exp(\vec{\alpha} \cdot \mathbf{f}(w, \mathbf{h}))$ is a normalizing factor.

It can be shown [15] that the use of this model with maximum likelihood parameter estimation is justified on information-theoretic grounds when q represents some prior knowledge about the true distribution and when the expected values of $\mathbf{f}$ in the training corpus are identical to their true expected values.[4] The model is the distribution with minimum Kullback-Leibler divergence from the reference distribution (or the one with maximum entropy if the reference distribution is uniform) among all

[3] Note that IBM1 can be seen as a 0th order Hidden Markov Model in which states correspond to tokens in the current source sentence.

[4] Another interpretation, which has been less well publicized in the NLP literature, is that of a single-layer neural net with certain weight constraints and a "softmax" output function [16]. See appendix C for details.

those which give the required expected values for $\mathbf{f}$. It can therefore be seen as the distribution that makes minimal assumptions beyond what is known about the true distribution. There is no requirement that the components of $\mathbf{f}$ represent disjoint or statistically independent events.

Although this result provides motivation for the use of MEMD models, it offers only weak guidance on how to select q or $\mathbf{f}$. In practice, q is usually chosen on the basis of efficiency considerations (when the information it captures would be computationally expensive to represent as components of $\mathbf{f}$), and $\mathbf{f}$ is established using heuristics such as described in the next section. Once q and $\mathbf{f}$ have been chosen, the IIS algorithm [43] can be used to find maximum likelihood parameter values.

In the current context, since the aim was to compare equivalent linear and MEMD models, I used an interpolated trigram as the reference distribution and binary indicator functions over bilingual word pairs as features (ie, components of $\mathbf{f}$). A pair of source,target words (s,t) has a corresponding feature function:

$$f_{st}(w,\mathbf{h},\mathbf{s}) = \begin{cases} 1, & s \in \mathbf{s} \text{ and } t = w \\ 0, & \text{else} \end{cases}$$

Using the notational convention that α_{st} is 0 whenever the corresponding feature f_{st} does not exist in the model, the final MEMD model can be written compactly as:

$$p(w|\mathbf{h},\mathbf{s}) = q(w|\mathbf{h}) \exp(\sum_{s\in\mathbf{s}} \alpha_{sw})/Z(\mathbf{h},\mathbf{s}),$$

where $q(w|\mathbf{h})$ is the interpolated trigram.

Comparing this to the model described in the previous section,

$$p(w|\mathbf{h},\mathbf{s}) = \lambda q(w|\mathbf{h}) + (1-\lambda) \sum_{j=0}^{J} p(w|s_j)/(J+1),$$

we can see that the two models are structurally quite similar, with the MEMD feature weights α_{sw} playing the role of the translation probabilities $p(w|s)$ in IBM1. The main difference between the two is that the contributions from the language and translation

components are added in the linear model and multiplied in the MEMD model. This is highly significant, as will be seen below. One minor difference is that, for the sake of efficiency, the MEMD model sums over contributions from source sentence *words*, whereas IBM1 sums over *tokens* plus the null word. The MEMD model contains $m+n$ free parameters, where m designates the number of free parameters in the trigram and n the number of word-pair features. A linear model defined over the same set of word pairs will contain $m+n+1-|V_s|+|V_t|-1$[5] free parameters, where V_s and V_t are the source and target vocabularies, so if the vocabulary sizes are equal the two models will contain precisely the same number of free parameters.

An important practical difference between the two models is the requirement to calculate the MEMD normalizing factor $Z(\mathbf{h}, \mathbf{s})$ for each context in which this model is used. This makes the MEMD model much more computationally expensive to train than the linear model, so that it is not feasible to have it incorporate all available word-pair features—ie all bilingual pairs of words which cooccur in some aligned sentence pair in the training corpus. Moreover, since the empirical expectations of features are supposed to reflect their true values, having a feature for *every* cooccurring pair in the corpus would be theoretically inadvisable even if it were computationally feasible. Some method of selecting a subset of reliable features is therefore required, as described in the next section.

3.3 Feature Selection

An easy way to select features would be to take high-frequency word pairs, since the empirical counts of these are likely to be close to their true values. The problem with this is that the vast majority of such pairs will be completely uninformative ones like (*the*,*de*) for English/French. These will not hurt the model's accuracy, because they

[5] One free combining weight, one normalization constraint per source word, and $|V_t|-1$ free parameters from $p(w|s_0)$

will be assigned weights close to 0, but they will waste valuable training time that could be spent on other, more informative, pairs. The goal of a good feature selection algorithm should therefore be to not only find reliable word pairs, but also significant ones: those that capture some valid translation relation. I experimented with three methods for selecting bilingual word pairs for inclusion in the models, as described in the following sections. All methods assign scores to individual pairs, so feature subsets of any desired size can be extracted by taking just the highest-ranked pairs.

3.3.1 Mutual Information

The simplest scoring method was mutual information (MI), defined for each word pair (s, t) as:

$$I(s;t) = \sum_{x=s,\bar{s}} \sum_{y=t,\bar{t}} \tilde{p}(x,y) \log \frac{\tilde{p}(x,y)}{\tilde{p}(x)\tilde{p}(y)},$$

where $\tilde{p}(s,t)$ is the probability that a randomly chosen pair of cooccurring source and target tokens in the corpus is (s,t); $\tilde{p}(s,\bar{t})$ is the probability that, in such a pair, the source token is s and the target token is not t; etc; and $\tilde{p}(x)$ and $\tilde{p}(y)$ are the left and right marginals of $\tilde{p}(x,y)$. Mutual information measures the degree to which occurrences of s and t are non-independent, so it is a reasonable choice for scoring pairs.

3.3.2 MEMD Gains

The second scoring method was an approximation of the MEMD gain for feature f_{st}, defined as the log-likelihood difference between a MEMD model which includes this feature and one which does not:

$$G_{st} = \frac{1}{|\mathcal{T}|} \log \frac{p_{st}(\mathcal{T}|\mathcal{S})}{p(\mathcal{T}|\mathcal{S})}$$

where the training corpus $(\mathcal{S}, \mathcal{T})$ consists of a set of (statistically independent) sentence pairs $(\mathbf{s}, \mathbf{t})$, and p_{st} is the model which includes f_{st}. Since MEMD models are

trained by finding the set of feature weights which maximizes the likelihood of the training corpus, it is natural to rate features according to how much they contribute to this likelihood. A powerful strategy for using gains is to build a model iteratively by adding at each step the feature which gives the highest gain with respect to those already added. Berger et al [15] describe an efficient algorithm for accomplishing this in which approximations to $p_{st}(\mathcal{T}|\mathcal{S})$ are computed in parallel for all (new) features f_{st} by holding all weights in the existing model fixed and optimizing only over each α_{st}. However, this method requires many expensive passes over the corpus to optimize the weights for the set of features under consideration at each step, and it adds only one feature per step, so it is not practical for evaluating feature sets containing thousands of features or more.

In a recent paper [103], Printz argues that it is usually sufficient to perform the iteration described in the previous paragraph only once, in other words that features can be ranked simply according to their gain with respect to some initial model. He also gives an algorithm for computing approximate gains which requires only a single pass over the training corpus. The method involves maximizing $g_{st}(\alpha)$, the difference in log likelihood between a model in which $\alpha_{st} = \alpha$, and the initial model. Under the approximation described in the previous paragraph, $\max_\alpha g_{st}(\alpha) \approx G_{st}$. The technique used to maximize $g_{st}(\alpha)$ is to compute the value of its derivative $g'_{st}(\alpha)$ at a predetermined set of sample points $\{\alpha_1, \ldots, \alpha_N\}$ in one pass over the corpus, then construct a numerical approximation $h'_{st}(\alpha)$ from these values. The root of $h'_{st}(\alpha)$ approximates the optimum value of α, and a definite integral which depends on the root can be evaluated numerically to give an approximation to $\max_\alpha g_{st}(\alpha)$ and therefore to G_{st}. Appendix C gives more details on this procedure. I adopted Printz' algorithm to compute gain scores for word pairs, using the reference trigram as the initial model.

3.3.3 IBM1 Gains

The final scoring method ranks word pairs according to the gain of the corresponding translation parameter $p(t|s)$ within IBM1. Instead of taking gains with respect to an initial model as in the previous section, I computed them with respect to a "full" model which was trained using all available word pairs:

$$G_{st} = \frac{1}{|\mathcal{T}|} \log \frac{p(\mathcal{T}|\mathcal{S})}{p_{\bar{st}}(\mathcal{T}|\mathcal{S})},$$

where $p_{\bar{st}}$ denotes the full IBM1 model p with the parameter $p(t|s)$ set to zero and the resulting distribution $p(w|s)$ renormalized. The advantage of this method is that it gives a measure of each parameter's worth in the presence of other parameters. However, as is the previous section, it is an approximation because determining the true gain would require retraining $p_{\bar{st}}$ and not merely renormalizing.

A problem with IBM1 gains is that they are not very robust. This can be seen from the definition:

$$G_{st} = \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{s},\mathbf{t})\in(\mathcal{S},\mathcal{T})} \log \frac{p(\mathbf{t}|\mathbf{s})}{p_{\bar{st}}(\mathbf{t}|\mathbf{s})}.$$

If the corpus contains a sentence pair $(\mathbf{s}, \mathbf{t})$ which consists only of a single word pair (s, t), then G_{st} will contain the term $\frac{1}{|\mathcal{T}|} \log \frac{p(t|s)+p(t|s_0)}{p(t|s_0)}$, so if $p(t|s_0)$ is close to zero (as is frequently the case), G_{st} will be close to infinity, even though (s, t) may occur only once in the training corpus.[6] To remedy this, I computed gains with respect to a linear combination of IBM1 and a smoothing model u, of the form $\lambda p(w|\mathbf{s})+(1-\lambda)u(w|\mathbf{h}, \mathbf{s})$. In the experiments reported below, I used a uniform distribution for u, with $\lambda = .99$.[7]

[6] Recall that s_0 is the null word prepended to all source sentences to account for target words which have no obvious translation in $\mathbf{s}$. In general, maximum likelihood values for $p(t|s_0)$ will tend to be low for those target words t that are infrequent and have "dedicated" translations s, because the corpus likelihood can be increased by assigning more probability mass to other words in the distribution $p(w|s_0)$.

[7] Another interesting choice for u would be the interpolated trigram, which would make the method described here more similar to the MEMD gain ranking described in the previous section.

for all word pairs (s,t): $G_{st} \leftarrow 0$

for each sentence pair $(\mathbf{s},\mathbf{t}) \in (\mathcal{S},\mathcal{T})$:

  for each token t in $\mathbf{t}$:

    $K \leftarrow \sum_{j=0}^{J} p(t|s_j) + \frac{(1-\lambda)(J+1)}{\lambda} u(t|\mathbf{h},\mathbf{s})$

    for each word s in $\mathbf{s}$:

      $G_{st} \leftarrow G_{st} + \log \frac{K}{K - \mathrm{freq}_{\mathbf{s}}(s) p(t|s)}$

      for all $t' \neq t$:

        $G_{st'} \leftarrow G_{st'} + \log \frac{K}{K + \mathrm{freq}_{\mathbf{s}}(s) \frac{p(t|s)p(t'|s)}{1-p(t'|s)}}$ *

for all (s,t): $G_{st} \leftarrow G_{st}/|\mathcal{T}|$

Figure 3.1. Algorithm for IBM1 gains. $\mathrm{freq}_{\mathbf{s}}(s)$ gives the number of times s occurs in $\mathbf{s}$.

Smoothed IBM1 gains can be computed in parallel in a single pass over the training corpus using the algorithm in figure 3.1. The line marked with an asterisk takes into account the increase in $p(t|s)$ due to renormalizing the distribution $p(w|s)$ after setting $p(t'|s)$ to zero, for each word $t' \neq t$ in the vocabulary.[8] Since this must be performed for every bilingual token pair in the corpus, it is extremely expensive. To make the algorithm tractable, I performed this step only for those t' such that $p(t'|s) \geq .01$. This causes the gains for pairs (s,t) such that $p(t|s) < .01$ to be slightly overestimated, but since the gains of such parameters are low in any case this does not seem likely to radically change the ranking of the top-ranked features in the parameter set.

[8] That is, setting $p(t'|s)$ to 0 and renormalizing causes $p(t|s)$ to rise slightly, so $G_{st'}$ has a non-0 contribution from terms of the form $\log \frac{p(t|\mathbf{s})}{p_{s\bar{t}'}(t|\mathbf{s})}$ whenever $s \in \mathbf{s}$. Using the notation of figure 3.1, this contribution is $\log \frac{K}{K + \mathrm{freq}_{\mathbf{s}}(s)[-p(t|s) + \frac{p(t|s)}{1-p(t'|s)}]}$, where $p(t|s)$ in $p(t|\mathbf{s})$ is replaced by the renormalized term $\frac{p(t|s)}{1-p(t'|s)}$ in $p_{s\bar{t}'}(t|\mathbf{s})$. Rearranging gives the expression shown on the starred line. Note that this quantity is negative, so the effect is to lower $G_{st'}$.

segment	file pairs	sentence pairs	English tokens	French tokens
train	922	1,639,250	29,547,936	31,826,112
held-out	30	54,758	978,394	1,082,350
test	30	53,676	984,809	1,103,320

Table 3.1. Corpus segmentation. The *held-out* segment was used to train interpolation coefficients for the trigram and the combining weights for the overall linear model; the *train* segment was used for all other training tasks.

3.4 Experiments

All experiments were run on the Canadian Hansard corpus (transcripts of parliamentary proceedings) for the years 86-94, with English as the source language and French as the target language. After tokenization, mapping to lowercase, and segmentation into sentences, the corpus was aligned using the method described in [107], then, to improve its quality, filtered to retain only 1-1 sentence alignments containing 40 or fewer words per side. It was divided into disjoint training, held-out, and test segments, as shown in table 3.1.

To evaluate performance, I used perplexity: $p(\mathcal{T}|\mathcal{S})^{-1/|\mathcal{T}|}$, where p is the model being evaluated, and $(\mathcal{S}, \mathcal{T})$ is the test corpus. Perplexity is a useful measure because it takes on convenient values (on the order of 100), is independent of corpus size, and has an intuitive interpretation as the size of a uniform distribution (assumed to contain the correct word) which would give the same classification performance as the model being evaluated.[9] It has also been used to evaluate full-fledged SMT systems [2]. One thing to note about this measure is that its value is infinite whenever a model assigns zero probability to a word in the corpus.

[9] Thus it can be interpreted as the average size of a TransType-style menu of alternate proposals that would be required to ensure that the window contained the correct proposal.

To ensure a fair comparison, all models use the same source and target vocabularies consisting only of words which appeared in the training corpus, plus one special unknown word *UNK*. During training, any word with frequency 1 in the corpus was mapped to UNK. During testing, any word not found in the vocabulary was mapped to UNK, and the probability assigned to each UNK target token was divided by the total number of out-of-vocabulary words in the text. Not including UNK, the English source vocabulary contained 51,604 words, and and the French target vocabulary contained 65,961 words.[10]

The first set of tests served to determine the optimum MEMD feature-selection method. I ranked all 35 million bilingual word pairs cooccurring within aligned sentence pairs in the training corpus using the MI and IBM1 gains methods. Because the MEMD gains method was much more expensive, I used it to rank only a short list of approximately 800,000 pairs derived by merging the top 500,000 candidates from each of the other methods. As shown in table 3.2, the three methods give substantially different rankings, even among the top 10 pairs. For each method, I trained MEMD models on a sequence of successively larger feature sets consisting of the top-ranked word pairs for that method. The results are shown in figure 3.2. Due to time constraints, I trained models with more than 10,000 features only for the IBM1 feature sets, which outperformed the other methods by a small margin. It is noteworthy that test perplexity is still dropping at 30,000 features, indicating that it would be worthwhile to train even larger models (as corroborated in table 3.3). [11]

In the next set of tests, I compared the performance of the MEMD models to the linear models. Since the number of features in the MEMD models I used was

[10] Note that the 500k word forms used in the TransType prototype include those from a French dictionary. This was not used for the work described in this thesis.

[11] Since the source vocabulary contains about 65,000 words, one would expect at least this many word-pair parameters in a model. However, an inspection of the list of pairs ranked lower than 50,000 by the IBM1 gains method shows significant amounts of noise.

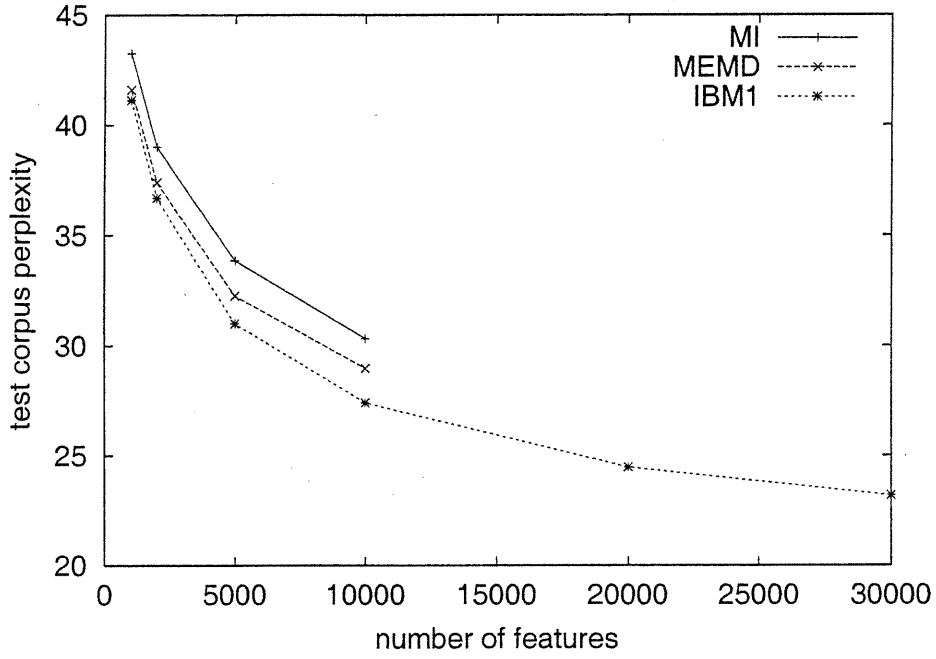

Figure 3.2. MEMD performance versus number of features for various feature-selection methods.

MI		MEMD gains		IBM1 gains	
:	:	mr.	m.	and	et
mr.	m.	i	je	government	gouvernement
we	nous	we	nous	we	nous
i	je	?	?	,	,
?	?	government	gouvernement	:	:
mr.	:	and	et	minister	ministre
minister	ministre	:	m.	i	je
)	)	member	député	?	?
(	)	minister	ministre	.	.
government	gouvernement	but	mais	canada	canada

Table 3.2. Top 10 word pairs for each feature-selection method.

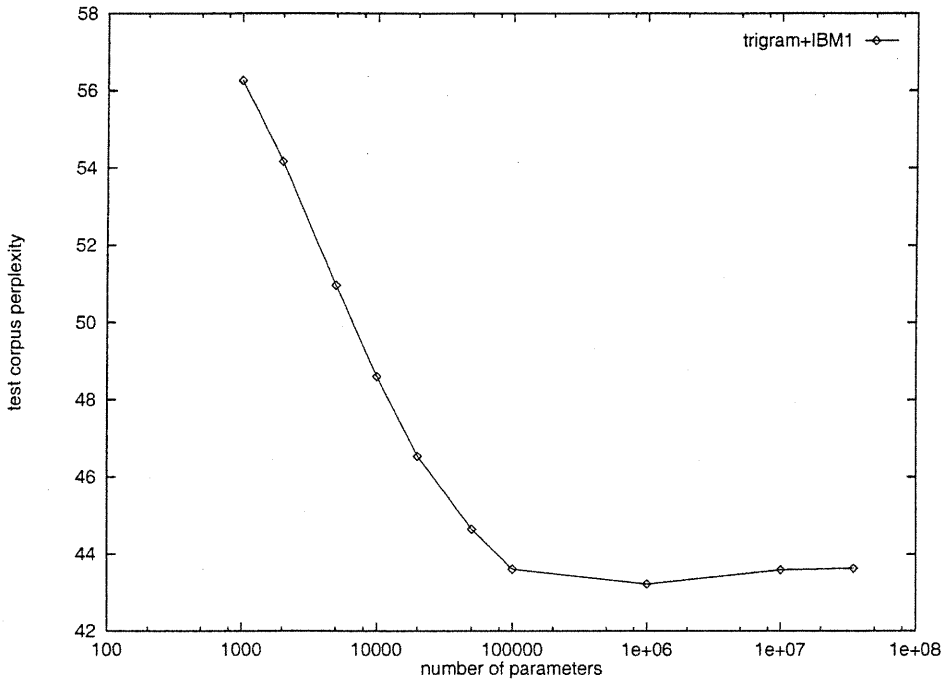

Figure 3.3. Performance of the linear model versus number of IBM1 parameters.

much smaller than the number of parameters in the full IBM1, I wanted to be sure that any performance difference was not due to IBM1 overfitting the training corpus. To eliminate this possibility, I optimized the number of IBM1 parameters by training linear models with various sizes of translation parameter sets obtained from the IBM1 gain ranking.[12] As shown in figure 3.3, the larger linear models do exhibit a very slight overtraining effect, with the optimum parameter set size around 1M, compared to 35M parameters in the full model.

The final results from various selected linear and MEMD models are presented in table 3.3. The MEMD models give a striking improvement over the linear models,

[12] In cases where the parameter set included no translations for some "known" source word s, I removed any tokens of s from the source sentence before calculating $p(w|\mathbf{s})$ using IBM1, so as not to penalize the smaller models by dividing by an overly large normalization constant $J + 1$.

model	translation parameters	perplexity	improvement over baseline
trigram	—	61.0	—
trigram + full IBM1	34,969,331	43.6	0%
trigram + optimum IBM1	1,000,000	43.2	0.9%
MEMD with IBM1 gains	1,000	41.1	5.7%
MEMD with IBM1 gains	30,000	23.2	46.9%
MEMD with IBM1 gains	50,000	21.8	50.0%

Table 3.3. Comparison of model performances

with a 1000-feature MEMD model performing better than the best linear model (despite containing 1000 times fewer word-pair parameters), and the best MEMD model yielding a perplexity reduction of 50% over the baseline linear model.

3.5 Discussion

The main result of this chapter is that the MEMD framework is a much more effective way to combine information from different sources than linear interpolation, at least for the problem studied here. It is fairly easy to see why this should be the case: MEMD essentially multiplies predictive scores arising from different sources rather than averaging them. This gives information sources which assign either very high or very low scores in some context much more influence over the final result. When such scores are based upon reliable evidence, this will lead to better models.

Another advantage enjoyed by the MEMD model in the current context was that its feature weights were trained in the presence of the reference trigram (ie, the same setting in which the weights are destined to be used), whereas the IBM1 parameters were trained in isolation. It would be an interesting experiment to modify the EM algorithm for IBM1 to address this imbalance, but given the fundamental weakness

of linear interpolation it is very doubtful whether this would significantly narrow the large performance gap between the MEMD and linear models. A symmetrical and perhaps more sensible experiment would be to compare the MEMD model with a log-linear combination of a trigram and IBM1, of the form $p(w|\mathbf{h})^{\alpha}\, p(w|\mathbf{s})$. This would give a better idea of the effect of training word-pair parameters in the presence of the trigram, although it would still differ from the MEMD model in the way evidence from different source words was represented and combined. Yet another interesting experiment would be to compare the perplexity given by the Bayes formulation in equation (3.2).

One somewhat surprising result of these experiments was that the IBM1 gains feature selection method resulted in better models than the MEMD gains method. Even more surprising was that models with IBM1 feature sets had lower *training* corpus perplexity than those with MEMD feature sets, despite the fact that the MEMD method is supposed to rank features according to how much they will lower training corpus perplexity. A possible explanation for this is that the gain over the reference trigram is not a good predictor of the gain in the presence of many other features; this is borne out by the fact that, for very small feature sets (on the order of 100 words and less), the MEMD method did outperform the IBM1 method, in both test and training perplexity. This suggests that a staged approach to MEMD selection might be useful, where in each stage the top-ranking features would be added to the model, which would then be retrained prior to calculating a new ranking with Printz' algorithm. Another explanation for the poor performance of the MEMD selection strategy is simply the presence of noise from Printz' algorithm, which has many parameters that require tuning. More testing is needed to determine if this method could be improved by finding better parameter values for Printz' algorithm, but in any case IBM1 gains appear to offer a viable and much less computationally expensive alternative.

3.5.1 Comparison with SMT

The MEMD model with a reference trigram has an interesting similarity to the noisy channel model of SMT in that both models combine their language and translation components by multiplication: $q(w|\mathbf{h})\exp(\vec{\alpha}\cdot\mathbf{f}(w,\mathbf{h},\mathbf{s}))$ in the former case, and $p(\mathbf{t})p(\mathbf{s}|\mathbf{t})$ in the latter. Furthermore, since many SMT systems use ngram language models, and perform left-to-right searches to find the optimum target text, the quantities evaluated during noisy-channel decoding are often of the form $q(w|\mathbf{h})v(w,\mathbf{h},\mathbf{s})$, where $q(w|\mathbf{h})$ is an ngram and $v(w,\mathbf{h},\mathbf{s})$ is some function derived from $p(\mathbf{s}|\mathbf{t})$ for scoring the partial translation hypothesis $(\mathbf{h},w)$. The similarity between this expression and the one for the MEMD model above suggests that the use of a MEMD model for $p(w|\mathbf{h},\mathbf{s})$ to perform SMT within the framework proposed here may not entail any loss of accuracy compared to the noisy channel method.

One possible objection to this is easily disposed of. MEMD models tend to be expensive because the normalization constant $Z(\mathbf{h},\mathbf{s})$ must be calculated for each new context $(\mathbf{h},\mathbf{s})$ as it is encountered, and this requires summing over the whole target vocabulary. However, any search procedure must in principle examine all words in the vocabulary in any case, so calculating $Z(\mathbf{h},\mathbf{s})$ implies very little extra cost. MEMD models *are* very expensive to train, but I have demonstrated that it is feasible to train viable models containing large sets of word pairs, and these constitute the bulk of any statistical translation model.

Another objection to this idea is that it would be too difficult to formulate a realistic translation model for $p(w|\mathbf{h},\mathbf{s})$ (as opposed to $p(\mathbf{s}|\mathbf{t})$), especially within the MEMD framework. This may indeed be the case; the model I have described in this chapter is certainly too weak to constitute counter evidence. It is obvious that the IBM models 2–5, described in appendix B, capture more sophisticated translation phenomena than just the word-for-word relations in this model. However, even IBM1, when used within the noisy-channel framework, is likely to outperform its MEMD

counterpart. To see why, suppose for simplicity that the language model is a uniform distribution, and observe that the expression to be optimized in $\mathbf{t}$ when using a "reversed" IBM1 is:

$$p(\mathbf{s}|\mathbf{t}) = \prod_{j=1}^{J} p(s_j|\mathbf{t}) = \prod_{j=1}^{J} \sum_{i=0}^{I} p(s_j|t_i).$$

This effectively ensures that, to be considered viable, a candidate $\mathbf{t}$ must contain a reasonable translation for *each* source word s_j, since otherwise the factor $p(s_j|\mathbf{t})$ would make the probability associated with $\mathbf{t}$ very low. Finding a concise $\mathbf{t}$ that meets this requirement is the essence of the NP-completeness proof by reduction from the minimum set cover problem in [71].[13] In contrast, the analogous expression for the MEMD model would be:

$$p(\mathbf{t}|\mathbf{s}) = \prod_{i=1}^{I} p(t_i|\mathbf{s}) = \prod_{i=1}^{I} \exp(\sum_{s \in \mathbf{s}} \alpha_{st_i})/Z(\mathbf{s}),$$

which is optimized by picking the single word $\hat{t} = \text{argmax}_t \sum_{s \in \mathbf{s}} \alpha_{st}$ that is most strongly predicted by $\mathbf{s}$, and placing this at every position in $\mathbf{t}$. To make the MEMD model competitive with the noisy channel approach, therefore, a first step would be to add explicit parameters to enforce the coverage constraints that the noisy channel enforces implicitly. These would reduce the probability of target words w whose translations $s \in \mathbf{s}$ already had valid translations in $\mathbf{h}$. Unfortunately, this would have the consequence of introducing a complete dependence on $\mathbf{h}$, and therefore drastically increasing search complexity.

[13] The other, orthogonal, proof given in this paper is by reduction from the Hamilton circuit problem, and has to do with the difficulty of finding the best permutation of a given set of target words according to a bigram language model. These two proofs cover the two main problems of translation by word replacement: word selection and word ordering.

3.6 Conclusion

The main conclusion from this chapter is that a simple MEMD-based translation model for $p(w|\mathbf{h}, \mathbf{s})$ can outperform an equivalent linear combination of a trigram language model and the IBM model 1 by 50% in test corpus perplexity, and do so with far fewer parameters. The small size of this model, together with the fact that it has the Markov property, makes it a good candidate for the text-prediction application.

Another conclusion is that a simple method for selecting bilingual word-pair features according to their gain within IBM model 1 offers better performance and significantly lower computational cost than a more general MEMD feature-selection algorithm due to Printz.

Chapter 4

AN IMPROVED MEMD TRANSLATION MODEL

This chapter describes an improved version of the MEMD model described in the previous chapter. The new model introduces a weak dependence on the target text context into the translation model, in the form of knowledge of the position of the current word w. This makes it significantly more powerful, while preserving its efficient search properties. Two methods are compared for incorporating information about the relative positions of bilingual word pairs into the MEMD translation model. The better of the two achieves almost 50% lower test corpus perplexity than an equivalent combination of a trigram language model and the classical IBM translation model 2.

4.1 Introduction

In the previous chapter, I described an efficient maximum entropy/minimum divergence (MEMD) model for $p(w|\mathbf{h},\mathbf{s})$ which incorporates a trigram language model and a translation component which is an analog of the IBM translation model 1 [23]. This model significantly outperforms an equivalent linear combination of a trigram and model 1 in test-corpus perplexity, despite using several orders of magnitude fewer translation parameters. However, it has one very obvious weakness: the probabilities from its translation component are essentially of the form $p(w|\mathbf{s})$, and are therefore completely independent of $\mathbf{h}$. From the standpoint of efficiency this is a good thing, because it guarantees that the overall model will have the desired Markov property, and can therefore support very fast searches. From the standpoint of predictive performance, it is not, because it means that the contributions from the translation

two hundred and fifty paintings will be selected for the exhibition , which will be held from september 23 to 27 .

cinquante cents cinquante cents cinquante cents cinquante cents cinquante cents cinquante cents cinquante ...

Figure 4.1. Example of an English to French translation generated by the MEMD model of chapter 3.

component will remain constant throughout the translation process, with only the trigram language model to preclude repetition of the same word. The weakness of this safeguard can be seen in figure 4.1, which gives an example of an actual translation produced by this model.[1]

A very simple way of giving the translation component some notion of context is to tell it the position i of the current target word, ie use $p(w|i, \mathbf{s})$ instead of $p(w|\mathbf{s})$. As shown in chapter 5, this can be done without destroying its efficient search properties. One way of using this kind of positional information is to incorporate parameters that increase or decrease the strength of the connections between w and its potential translations $s \in \mathbf{s}$ on the basis of how far away these words are from each other. Intuitively, the strengths of pairs s, w that occur in similar positions within their respective sentences should be increased, while those that are very distant from each other should be decreased. This is the information that I build into the MEMD model in this chapter. It is also the idea that underlies the IBM model 2 (IBM2), the next

[1] What is happening in this case is that the two most probable words w according to $p(w|\mathbf{s})$ happen to form a high-probability loop in the trigram's Markov chain (or more precisely, their two possible bigrams do), so they toggle back and forth. This seems (and is) very bad, but it must be kept in mind that the aim is not to use the model to produce entire translations like this, just to predict the next several words from a valid prefix $\mathbf{h}$.

in the classical series after IBM1. This means that, as in the last chapter, the MEMD model has a direct linear analog to which it can be compared.

4.2 Models

This section describes the models to be compared.

4.2.1 Linear Model

As a baseline for comparison I used a linear combination as in (3.1) of a standard interpolated trigram language model and the IBM translation model 2 (IBM2), with the combining weight λ optimized using the EM algorithm. IBM2 is derived as follows:[2]

$$\begin{aligned} p(w|i,\mathbf{s}) &= \sum_{j=0}^{J} p(w,j|i,\mathbf{s}) \\ &\approx \sum_{j=0}^{J} p(w|s_j)p(j|i,J) \end{aligned}$$

where as before J is the length of $\mathbf{s}$. The hidden variable j gives the position in $\mathbf{s}$ of the (single) source token s_j assumed to give rise to w, or 0 if there is none. The model consists of a set of word-pair parameters $p(t|s)$ and a set of position parameters $p(j|i,J)$; in model 1 (IBM1) the latter are fixed at $1/(J+1)$, as each position, including the empty position 0, is considered equally likely to contain a translation for w. Maximum likelihood estimates for these parameters can be obtained with the EM algorithm over a bilingual training corpus, as described in [23].

[2] Model 2 was originally formulated for $p(\mathbf{t}|\mathbf{s})$, but since target words are predicted independently it can also be used for $p(w|\mathbf{h},\mathbf{s})$. The only necessary modification in this case is that the position parameters can no longer be conditioned on $I = |\mathbf{t}|$, as they are in the original formulation.

4.2.2 MEMD Model 1

The starting point for the new models described below is the MEMD model developed in the last chapter (MEMD1), which I reproduce here for convenience. Recall that a MEMD model for $p(w|\mathbf{h},\mathbf{s})$ has the general form:

$$p(w|\mathbf{h},\mathbf{s}) = \frac{q(w|\mathbf{h},\mathbf{s})\exp(\vec{\alpha}\cdot\mathbf{f}(w,\mathbf{h},\mathbf{s}))}{Z(\mathbf{h},\mathbf{s})},$$

where $q(w|\mathbf{h},\mathbf{s})$ is a reference distribution, $\mathbf{f}(w,\mathbf{h},\mathbf{s})$ maps $(w,\mathbf{h},\mathbf{s})$ into an n-dimensional feature vector, $\vec{\alpha}$ is a corresponding vector of feature weights (the parameters of the model), and $Z(\mathbf{h},\mathbf{s}) = \sum_w q(w|\mathbf{h},\mathbf{s})\exp(\vec{\alpha}\cdot\mathbf{f}(w,\mathbf{h}))$ is a normalizing factor.

MEMD1 uses a trigram language model for the reference distribution q, and Boolean features corresponding to bilingual word pairs:

$$f_{st}(w,\mathbf{s}) = \begin{cases} 1, & s\in\mathbf{s} \text{ and } t = w \\ 0, & \text{else} \end{cases}$$

where (s,t) is a (source,target) word pair. Using the notational convention that α_{st} is 0 whenever the corresponding feature f_{st} does not exist in the model, MEMD1 can be written compactly as:

$$p(w|\mathbf{h},\mathbf{s}) = q(w|\mathbf{h})\exp(\sum_{s\in\mathbf{s}}\alpha_{sw})/Z(\mathbf{h},\mathbf{s}).$$

4.2.3 MEMD Model 2

IBM2 incorporates position information by introducing a hidden position variable and making independence hypotheses. This approach is not applicable to MEMD models,[3] whose features must capture events which are directly observable in the training corpus. It would be possible to use pure position features of the form f_{ijJ}, which capture the presence of *any* word pair at position (i,j,J) and are superficially similar to IBM2's position parameters, but these would add almost no information to

[3] At least in their original formulation. See the remarks at the end of this chapter.

MEMD1. On the other hand, features like f_{stijJ}, indicating the presence of a specific pair (s,t) at position (i,j,J), would cause severe data sparseness problems.

Encoding Positions as Feature Values

A simple solution to this dilemma is to let the value of a word-pair feature reflect the current position of the pair rather just its presence or absence. A reasonable choice for this is the value of the corresponding IBM2 position parameter $p(j|i,l)$:

$$f_{st}(w,i,\mathbf{s}) = \begin{cases} p(\hat{\jmath}_s|i,J), & s \in \mathbf{s} \text{ and } t = w \\ 0, & \text{else} \end{cases}$$

where $\hat{\jmath}_s$ is the position of s in $\mathbf{s}$, or the most likely position according to IBM2 if it occurs more than once: $\hat{\jmath}_s = \text{argmax}_{j:s_j=s}\, p(j|i,J)$. Using the same convention as in the previous section, the resulting model (MEMD2R) can be written:

$$p(w|\mathbf{h},\mathbf{s}) = \frac{q(w|\mathbf{h}) \exp(\sum_{s\in\mathbf{s}} \alpha_{sw} p(\hat{\jmath}_s|i,J))}{Z(\mathbf{h},\mathbf{s})}.$$

MEMD2R is simple and compact but poses a technical difficulty due to its use of real-valued features, in that the IIS training algorithm requires integer or Boolean features for efficient implementation. Since likelihood is a concave function of $\vec{\alpha}$, any hillclimbing method such as gradient ascent[4] is guaranteed to find maximum likelihood parameter values, but convergence is slower than IIS and requires tuning a gradient step parameter. Unfortunately, apart from this problem, MEMD2R also turns out to perform slightly worse than MEMD1, as described below.

Using Class-based Position Features

Since the basic problem with incorporating position information is one of insufficient data, a natural solution is to try to group word pair and position combina-

[4] I found that the "stochastic" variant of this algorithm, in which model parameters are updated after each training example, gave the best performance. Appendix C gives further details on training using gradient ascent.

tions with similar behaviour into classes such that the frequency of each class in the training corpus is high enough for reliable estimation. To do this, I made two preliminary assumptions: 1) word pairs (s,t) with similar MEMD1 weights α_{st} should be grouped together; and 2) position configurations with similar IBM2 probabilities should be grouped together. This converts the problem from one of finding classes in the five-dimensional space (s,t,i,j,J) to one of identifying rectangular areas on a 2-dimensional grid where one axis contains position configurations (i,j,J), ordered by $p(j|i,J)$; and the other contains word pairs (s,t), ordered by α_{st}. To simplify further, I partitioned both axes so as to approximately balance the total corpus frequency of all word pairs or position configurations within each partition. Thus the only parameters required to completely specify a classification are the number of position partitions and the number of word-pair partitions. Each combination of a position partition and a word pair partition corresponds to a class, and all classes can be expected to have roughly the same empirical counts.[5] The model (MEMD2B) based on this scheme has one feature for each class; if A designates the set of triples (i,j,J) in a position partition and B designates the set of pairs (s,t) in a word-pair partition, then for all A,B there is a feature:

$$f_{A,B}(w,i,\mathbf{s}) = \sum_{j=1}^{J} \delta[(i,j,J) \in A \ \wedge \ (s_j,w) \in B \ \wedge \ j = \hat{\jmath}_{s_j}],$$

where $\delta[X]$ is 1 when X is true and 0 otherwise. For robustness, I used these position features along with pure MEMD1-style word-pair features f_{st}. The weights $\alpha_{A,B}$ on the position features can thus be interpreted as correction terms for the pure word-pair weights $\alpha_{s,t}$ which reflect the proximity of the words in the pair. The model

[5] Sufficient conditions for having *exactly* the same counts are that the frequency distribution across word pair classes be identical for all position classes; or conversely that the frequency distribution across position classes be identical for all word pair classes. Neither holds precisely, but with as long as the partitions on each axis are not too fine we can expect both to hold approximately.

is:

$$p(w|\mathbf{h},\mathbf{s}) = \frac{q(w|\mathbf{h})\exp(\sum_{s\in\mathbf{s}}\alpha_{sw} + \alpha_{A(i,\hat{\jmath}_s,J),B(s,t)})}{Z(\mathbf{h},\mathbf{s})},$$

where $A(i,\hat{\jmath}_s,J)$ gives the partition for the current position, $B(s,t)$ gives the partition for the current word pair, and following the usual convention, $\alpha_{A(i,\hat{\jmath}_s,J),B(s,t)}$ is zero if these are undefined.

To find the optimal number of position partitions m and word-pair partitions n, I performed a greedy search, beginning at a small initial point (m,n) and at each iteration training two MEMD2B models characterized by (km,n) and (m,kn), where $k > 1$ is a scaling factor (note that both these models contain kmn position features). The model which gives the best performance on a validation corpus is used as the starting point for the next iteration. Since training MEMD models is very expensive, to speed up the search I relaxed the convergence criterion from a training corpus perplexity drop of $< .1\%$ (requiring 20-30 IIS iterations) to $< .6\%$ (requiring approximately 10 IIS iterations). I stopped the search when the best model's performance on the validation corpus did not decrease significantly from that of the model at the previous step, indicating that overtraining was beginning to occur.

4.3 Results

I tested the models on the Canadian Hansard corpus, with English as the source language and French as the target language, as before. The setup was identical to the one described in the previous chapter, with the exception of an additional held-out segment (*held-out 2*) used to optimize the parameters for MEMD2B, as shown in table 4.1. As before, perplexity was used as the test measure, and to ensure a fair comparison, all models used the same target vocabulary.

Figures 4.2 and 4.3 show, respectively, the path taken by the MEMD2B partition search, and the validation corpus perplexities of each model tested during the search.

segment	file pairs	sentence pairs	English tokens	French tokens
train	922	1,639,250	29,547,936	31,826,112
held-out 1	30	54,758	978,394	1,082,350
held-out 2	30	59,435	1,111,454	1,241,581
test	30	53,676	984,809	1,103,320

Table 4.1. Corpus segmentation. The *train* segment was the main training corpus; the *held-out 1* segment was used for combining weights for the trigram and the overall linear model; and the *held-out 2* segment was used for the MEMD2B partition search.

As shown in figure 4.2, the search consisted of 6 iterations. Since on all previous iterations no increase in position partitions beyond the initial value of 10 was selected, on the 5th iteration I tried decreasing the number of position partitions to 5. This model was not selected either, so on the final step only the number of word-pair partitions was augmented, yielding an optimal combination of 10 position partitions and 4000 word-pair partitions.

Table 4.2 gives the final results for all models. The IBM models tested here are those with the optimized set of 1M word-pair parameters, shown in the previous chapter to give slightly lower text-corpus perplexity than the unrestricted set of all 35M word pairs which cooccur within aligned sentence pairs in the training corpus. For the MEMD models, I give results for both the 20k word-pair set used for the position-parameter search, and a 50k word-pair set for which no special position-parameter tuning was done. The basic 50k MEMD1 model (with *no* position parameters) attains almost 40% lower perplexity than the comparable model 2 baseline, and MEMD2B with an optimal-sized set of position parameters achieves in a further drop of over 10%. Interestingly, the relative difference between IBM1 and IBM2's performance (18.5% lower perplexity for IBM2) is almost exactly the same as the difference be-

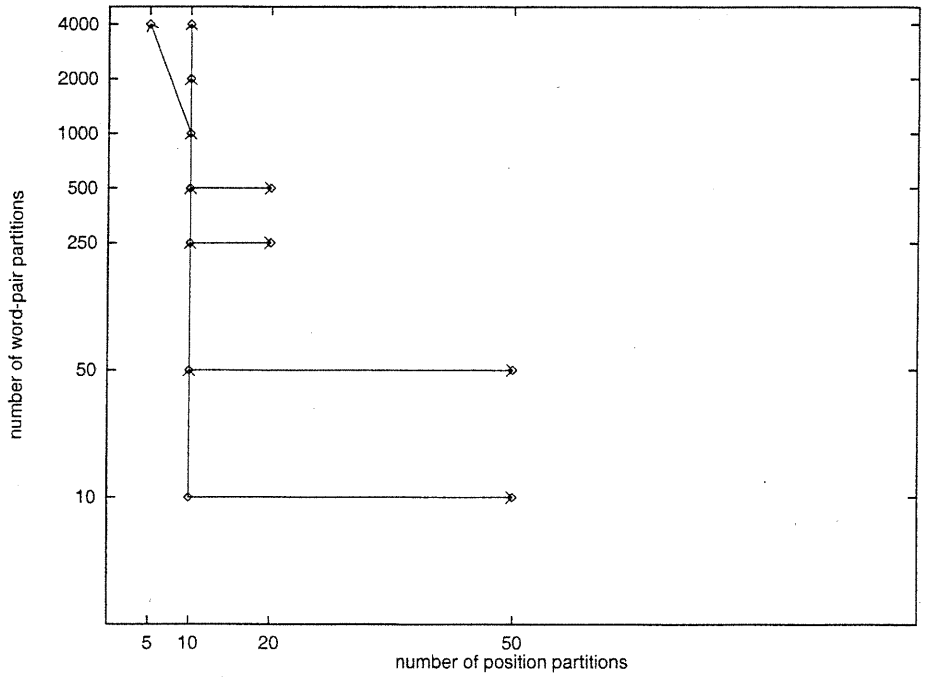

Figure 4.2. MEMD2B partition search path, beginning at the point $(10, 10)$. Arrows out of each point show the configurations tested at each iteration.

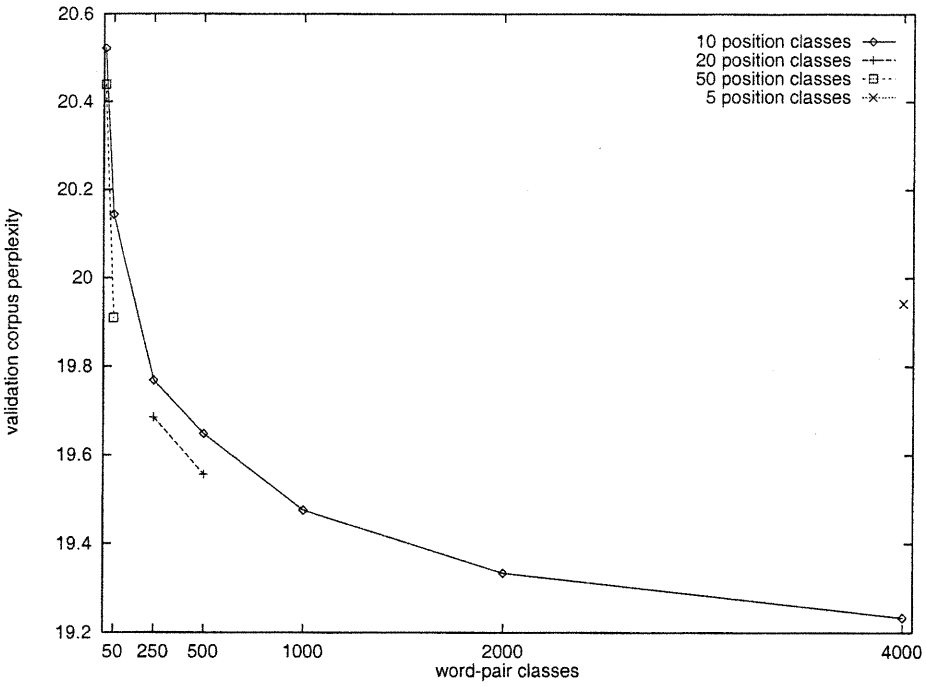

Figure 4.3. Validation corpus perplexities for various MEMD2B models. Each connected line in this graph corresponds to a vertical column of search points in figure 4.2.

model	word-pair parameters	position parameters	perplexity	improvement over baseline
trigram	0	0	61.0	—
trigram + IBM1	1,000,000	0	43.2	—
trigram + IBM2	1,000,000	115,568	35.2	0%
MEMD1	20,000	0	24.5	30.4%
MEMD1	50,000	0	21.8	38.1%
MEMD2R	20,000	0	28.4	19.3%
MEMD2B	20,000	10×10	22.1	37.2%
MEMD2B	20,000	10×4000	20.2	42.6%
MEMD2B	50,000	10×4000	17.7	49.7%

Table 4.2. Model performances. Linear interpolation is designated with a + sign; and the MEMD2B position parameters are given as $m\times$, where m and n are the numbers of position partitions and word-pair partitions respectively.

tween MEMD1 and MEMD2B (18.8% lower for MEMD2B).

4.4 Conclusion and Future Work

The main conclusion from this chapter is that, despite the lack of a hidden variable mechanism in the MEMD framework, it is possible to incorporate word-position information into the MEMD1 model described in the previous chapter. This is accomplished by using position features that depend on the *magnitude* of the MEMD1 word-pair weights they modify, through a classification mechanism that also depends on the magnitude of the IBM2 position parameter that would apply to the current token pair. This approach results in a model (MEMD2B) that obtains the same relative perplexity improvement—almost 20%—over MEMD1 as IBM2 does over IBM1 within a linear combination with a trigram. The relative improvement of MEMD2B

over the IBM2-based linear combination is almost 50%. An alternate approach of using IBM2 position-parameter values directly for MEMD position-feature values was found not to work.

4.4.1 Future Work

There are many ways in which this work could be extended. Although the approach proposed here for incorporating position information is robust and appears to work fairly well, it is not clear that it is the best solution to the problem. The method of classifying word-pair/position combinations is simplistic and rests on the unsubstantiated assumption that MEMD1 weights and IBM2 probabilities are the best indicators for this purpose. It is likely that better classification methods exist. The idea of assigning adjustment weights to capture variation with relative position also seems less elegant that the IBM2 combination $p(f|e)p(j|i,J)$, which does not require classification (though notice that the two methods are similar in that they multiply word-pair and position scores). Given the failure of MEMD2R, however, it is not clear what alternatives exist to a classification approach.

Beyond word positions, another source of information that could be captured without adding any dependence on $\mathbf{h}$ is source units such as *red tape* that translate to a single target word like *paperasserie*. In this case, we would expect the weight on a feature like *red tape/paperasserie* to be higher than the combined weights for *red/paperasserie* and *tape/paperasserie*. This technique could even be used to partially capture many-to-many relations such as *out to lunch/dans les patates*, by adding features like *out to lunch/dans*, *out to lunch/patates*, etc. Of course, this would require some method of discovering many-to-one or many-to-many translations in the first place, and this is far from a trivial problem [76, 93].

The most fruitful enhancements to the models I have described are likely to be those that better capture the relation between w and $\mathbf{h}$. As argued in the previous chapter and in the next one, this would in general destroy the Markov property

required to perform efficient Viterbi searches for upcoming text. However, there are two exceptional cases. First, MEMD features can look at the last two words in **h** without increasing search complexity beyond that of a trigram. This could be useful for fixed one-to-many translations like *potato/pomme de terre*, where a feature of the form *potato/pomme de* could raise the probability of *de* following *pomme* when the source text contained *potato*. However, the scope for improvement over the existing combination (of the trigram's prediction of *de* following *pomme* and the contribution from a word-pair feature *potato/de*) may not be very large.

Another situation in which it does not hurt to look at **h** is when **h** is the text typed by the translator. Of course, for predictions longer than a single word, **h** will contain a suffix that has *not* been typed by the translator, but special models could be developed to distinguish between this suffix and what comes before. This would be increasingly useful toward the end of a target sentence, as the text typed by the translator comes to constitute most of a complete translation.

Features connecting w to **h** (via **s**) should initially aim at enforcing the coverage constraint described in the previous chapter; other possibilities include emulating the fertility and distortion parameters of the IBM models 3–5 (see appendix B). The lack of hidden alignment variables complicates the problem of incorporating this information. For instance, an obvious strategy to enforce the coverage constraint is to add features that lower the contribution from source words that already have translations in **h**, in order to give other source words a chance. But determining whether a source word has a valid translation in **h** generally requires elaborating the global relationship between **h** and **s**—in other words, performing a word alignment. There are many possibilities for getting around the lack of an alignment mechanism, such as the classification approach used in MEMD2B, piggybacking on alignments produced by IBM models, or attempting to aggregate over all reasonable alignments. Each of these possibilities represents a significant research challenge, however. Another alternative would be to look at recently-proposed methods such as [74, 119] for adding

latent information to the basic MEMD framework. In all cases, the application of MEMD should focus on its main strength (which I have exploited in this chapter through the use of overlapping position features and pure word-pair features): the ability to simply and effectively combine information from difference sources.

Chapter 5

PREDICTION FOR TRANSLATORS

The previous two chapters have dealt with the problem of creating efficient models for the next-word distribution $p(w|\mathbf{h}, \mathbf{s})$ independently of any particular application. This chapter returns to the main topic of the thesis, and shows how these models can be used to predict text for translators in a way that is designed to maximize the usefulness of the predictive tool.

5.1 Overview

As described in chapter 1, the aim of the predictor is to find the prediction that maximizes the expected benefit to the user:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmax}}\, B(\mathbf{x}, \mathbf{h}, \mathbf{s}), \tag{5.1}$$

where $\mathbf{x}$, $\mathbf{h}$, and $\mathbf{s}$ are character strings. See figure 5.1, which reproduces the example from chapter 1 for convenience. Assuming the main factor that determines the true benefit to be the length k of the longest prefix of $\mathbf{x}$ that matches $\mathbf{x}^*$, the expected benefit becomes:

$$B(\mathbf{x}, \mathbf{h}, \mathbf{s}) = \sum_{k=0}^{l} p(k|\mathbf{x}, \mathbf{h}, \mathbf{s})\, B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k), \tag{5.2}$$

so the three problems that need to be solved in this approach are estimating the prefix probabilities $p(k|\mathbf{x}, \mathbf{h}, \mathbf{s})$, estimating the user benefit function $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k)$, and searching for $\hat{\mathbf{x}}$. My solutions to these problems are described in the following sections.

s: Let us return to serious matters.

t: $\overbrace{\text{On va r}}^{\mathbf{h}}\overbrace{\textit{evenir aux choses sérieuses.}}^{\mathbf{x}^*}$

x: *evenir à*

Figure 5.1. Example of a prediction for English to French translation. s is the source sentence, h is the part of its translation that has already been typed, $\mathbf{x}^*$ is what the translator wants to type, and x is a prediction. The surrounding contexts S and H are omitted.

5.2 *Estimating Prefix Probabilities*

The first step in estimating the prefix probabilities $p(k|\mathbf{x}, \mathbf{h}, \mathbf{s})$ is to convert them into a form that deals explicitly with character strings. To do this, note that $p(k|\mathbf{x}, \mathbf{h}, \mathbf{s})$ is the probability that the first k characters of $\mathbf{x}$ are correct *and* that the $k+1$th character (if there is one) is incorrect. For $k < l$:

$$\begin{aligned} p(k|\mathbf{x}, \mathbf{h}, \mathbf{s}) &= p(x_1^k, \bar{x}_{k+1}|\mathbf{h}, \mathbf{s}) \\ &= \sum_x p(x_1^k, x|\mathbf{h}, \mathbf{s}) - p(x_1^{k+1}|\mathbf{h}, \mathbf{s}) \\ &= p(x_1^k|\mathbf{h}, \mathbf{s}) - p(x_1^{k+1}|\mathbf{h}, \mathbf{s}) \end{aligned}$$

where x_i is the ith character in $\mathbf{x}$, $\bar{x}_{k+1}$ denotes the occurrence of any character other than x_{k+1}, and $x_1^k = x_1 \ldots x_k$. If $k = l$, $p(k|\mathbf{h}, \mathbf{s}) = p(\mathbf{x}|\mathbf{h}, \mathbf{s})$. Also, $p(x_1^0) \equiv 1$. It is straightforward, though expensive, to calculate exact estimates for the required string probabilities $p(x_1^k|\mathbf{h}, \mathbf{s})$ from the word-based model $p(w|\mathbf{h}, \mathbf{s})$. To see if this expense could be reduced without hurting performance, I experimented with two simple approximations to $p(x_1^k|\mathbf{h}, \mathbf{s})$, and compared the results to those given by the exact estimates.

A preliminary to all three methods of estimating string probabilities is to account

for cases where $\mathbf{h}$ does not end on a word boundary. In general, $\mathbf{h}$ can be tokenized into $\mathbf{h}', u$, where $\mathbf{h}'$ is a token sequence, and u is possibly null word prefix that follows. For example, in figure 5.1, $\mathbf{h}'$ is *on va*, and u = "r". Note that u will be null only if the last token in $\mathbf{h}$ can be unambiguously determined to have ended, for instance if it is followed by a blank character or if it is a punctuation mark. Then:

$$\begin{aligned} p(x_1^k|\mathbf{h},\mathbf{s}) &= p(x_1^k|\mathbf{h}',u,\mathbf{s}) \\ &= c\,p(u,x_1^k|\mathbf{h}',\mathbf{s}) / \sum_{w:w=u*} p(w|\mathbf{h}',\mathbf{s}), \end{aligned}$$

where the sum on the last line is over all words in the vocabulary that begin with u. To speed up this calculation, the vocabulary for this purpose was taken to be the active vocabulary V_A described in section 5.4—a subset of words considered very likely given $\mathbf{s}$. The correction factor $c = \sum_{w\in V_A} p(w|\mathbf{h},\mathbf{s})$ compensates for the slight underestimation of the sum in the denominator this induces. It is equivalent to assuming that the extensions to u occur in the same proportion in the main vocabulary as they do in V_A.

In the following sections, I describe how $p(u,x_1^k|\mathbf{h}',\mathbf{s})$ can be calculated. To simplify the presentation, however, I leave u implicit, and just write this as a normal string probability $p(x_1^k|\mathbf{h},\mathbf{s})$ that happens to have an integral number of tokens in $\mathbf{h}$.

5.2.1 Calculating Exact String Probabilities

The method of calculating "exact" string probabilities from the word based model is similar to the method for dealing with situations where $\mathbf{h}$ ends in a partial word. I first assume that the mapping between strings (that contain an integral number of words) and token sequences is one-to-one. This is a reasonable approximation given that the words in my lexicon do not contain whitespace or internal punctuation.[1] In this way, a string x_1^k can be tokenized unambiguously into a sequence of words

[1] A given token sequence can produce only one string, but some strings can be tokenized in several different ways, eg "m." can produce either $m+$. or m. Since the strings used for predictions were

of the form $w_1, \ldots, w_{m-1}, u_m$, where u_m is the final word prefix in x_1^k, null iff w_{m-1} unambiguously ends. The probability of this sequence can be obtained from the word-based model in the usual way:

$$p(w_1, \ldots, w_{m-1}, u_m | \mathbf{h}, \mathbf{s}) = \prod_{i=1}^{m-1} p(w_i | \mathbf{h}, w_1, \ldots, w_{i-1}, \mathbf{s}) p(u_m | \mathbf{h}, w_1, \ldots, w_{m-1}, \mathbf{s}).$$

The last factor is just:

$$p(u_m | \mathbf{h}, w_1, \ldots, w_{m-1}, \mathbf{s}) = \sum_{w : w = u_m *} p(w | \mathbf{h}, w_1, \ldots, w_{m-1}, \mathbf{s}), \tag{5.3}$$

where, as above, the sum is over all words in the vocabulary that begin with u_m.

It remains to account for the possible presence of separators after each word. One way to do this would be to introduce an additional factor of the form:

$$\prod_{i=1}^{m-1} p(s_i | w_i),$$

where s_i is the separator string—eg, the empty string or one or more blanks, tabs, etc—that follows the ith word (by definition, there is no separator after u_m), whose probability is assumed here to depend only on the preceding word. In keeping with the one-to-one assumption between token and character sequences, however, I set all these probabilities to 1.

Since string probabilities have to be calculated for each prefix of a given proposal $\mathbf{x}$, the sum in equation (5.3) over all words in the vocabulary must be performed l times (where l is the length of $\mathbf{x}$). To avoid the considerable expense of doing this in the most obvious way, I stored the vocabulary as a trie, and calculated the sums for all prefixes of all words simultaneously for each new context $\mathbf{h}, w_1, \ldots, w_{i-1}$ in a single recursive pass over the vocabulary. Storing the results against the nodes of the trie allows for quick retrieval of the required sums for each u_m by just following the

always generated from token sequences, any ambiguities were resolved in favour of the original token sequence.

appropriate link from the node for the previous u_m (at the next smaller value of k). This technique is similar to the one used in [49], and is explained in greater detail there.

5.2.2 Linear Estimates

To bypass the complexity (both time and coding) of the exact calculation, I tried an alternate approach that involves establishing exact probabilities only at the end of each word in $\mathbf{x}$, and assuming that the probabilities of the intervening characters vary linearly between these points. Suppose the last character in each word w_i in $\mathbf{x}$ is at position k_i. Then for $k = k_{i-1} \ldots k_i$:[2]

$$p(x_1^k|\mathbf{h},\mathbf{s}) \approx p(w_1^{i-1}|\mathbf{h},\mathbf{s}) + (k - k_{i-1})\frac{p(w_1^i|\mathbf{h},\mathbf{s}) - p(w_1^{i-1}|\mathbf{h},\mathbf{s})}{k_i - k_{i-1}}.$$

Any whitespace after the end of w_{i-1} is included in w_i. The difference between the probability of each prefix and the next longer one in this range is:

$$\begin{aligned} p(x_1^k|\mathbf{h},\mathbf{s}) - p(x_1^{k+1}|\mathbf{h},\mathbf{s}) &\approx (k - k_{i-1} - (k+1-k_{i-1}))\frac{p(w_1^i|\mathbf{h},\mathbf{s}) - p(w_1^{i-1}|\mathbf{h},\mathbf{s})}{k_i - k_{i-1}} \\ &= \frac{p(w_1^{i-1}|\mathbf{h},\mathbf{s}) - p(w_1^i|\mathbf{h},\mathbf{s})}{k_i - k_{i-1}}. \end{aligned}$$

Since this term is constant between word endings, it can be factored out of the sum in equation (5.2) as follows:

$$\begin{aligned} B(\mathbf{x},\mathbf{h},\mathbf{s}) &= \sum_{k=0}^{l-1}(p(x_1^k|\mathbf{h},\mathbf{s}) - p(x_1^{k+1}|\mathbf{h},\mathbf{s}))B(\mathbf{x},\mathbf{h},\mathbf{s},k) + p(x_1^l|\mathbf{h},\mathbf{s})B(\mathbf{x},\mathbf{h},\mathbf{s},l) \\ &\approx \sum_{i=1}^{m}\sum_{k=k_{i-1}}^{k_i-1}\frac{p(w_1^{i-1}|\mathbf{h},\mathbf{s}) - p(w_1^i|\mathbf{h},\mathbf{s})}{k_i - k_{i-1}}B(\mathbf{x},\mathbf{h},\mathbf{s},k) + p(w_1^l|\mathbf{h},\mathbf{s})B(\mathbf{x},\mathbf{h},\mathbf{s},l) \\ &= \sum_{i=1}^{m}\frac{p(w_1^{i-1}|\mathbf{h},\mathbf{s}) - p(w_1^i|\mathbf{h},\mathbf{s})}{k_i - k_{i-1}}\sum_{k=k_{i-1}}^{k_i-1}B(\mathbf{x},\mathbf{h},\mathbf{s},k) + p(w_1^l|\mathbf{h},\mathbf{s})B(\mathbf{x},\mathbf{h},\mathbf{s},l) \end{aligned}$$

where $k_0 = 0$ and $p(w_1^0|\mathbf{h}, \mathbf{s}) = 1$. This can be carried out very efficiently, modulo the expense of summing the benefit function.

5.2.3 At-End Estimate

The sum over the prefix lengths in equation (5.2) will distinguish between two different proposals that have the same length and overall probability of occurring in the current context, on the basis of their contents. An obvious simplifying hypothesis is that this does not matter, that is, that the benefit of a proposal depends only on whether it is correct as a whole. To test this hypothesis, I used the following approximation:

$$B(\mathbf{x}, \mathbf{h}, \mathbf{s}) \approx p(\mathbf{x}|\mathbf{h}, \mathbf{s}) B(\mathbf{x}, \mathbf{h}, \mathbf{s}, l).$$

Notice that this still allows for hypotheses with identical lengths and probabilities to be distinguished by $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, l)$. (although not by the specific benefit function described in the next section). Also note that, unlike the two previously described prefix probability estimation methods, this is a heuristic because the distribution $p(k|\mathbf{x}, \mathbf{h}, \mathbf{s})$ is not normalized over all k.

5.2.4 Model Accuracy

The accuracy of the prefix probability estimates made in the previous sections depends on the accuracy of next-word probability estimates $p(w|\mathbf{h}, \mathbf{s})$ given by the model. To assess this, I compared the word-classification accuracy of the best configuration of the MEMD2B model described in the previous chapter to its own projected accuracy

[2] A confession: the main motivation for this formula was to simplify an implementation that needed to get done fast, by allowing the probability term to be factored out of the benefit sum. A far better approximation would be to interpolate log probabilities instead: $\log p_k = \log p_1 + \frac{k-k_1}{k_2-k_1}(\log p_2 - \log p_1)$, where p_k is short for $p(x_1^k|\mathbf{h}, \mathbf{s})$, etc. The motivation for this is that the number of possible strings increases exponentially with length, so on average their probabilities will decrease exponentially, rather than linearly, with length.

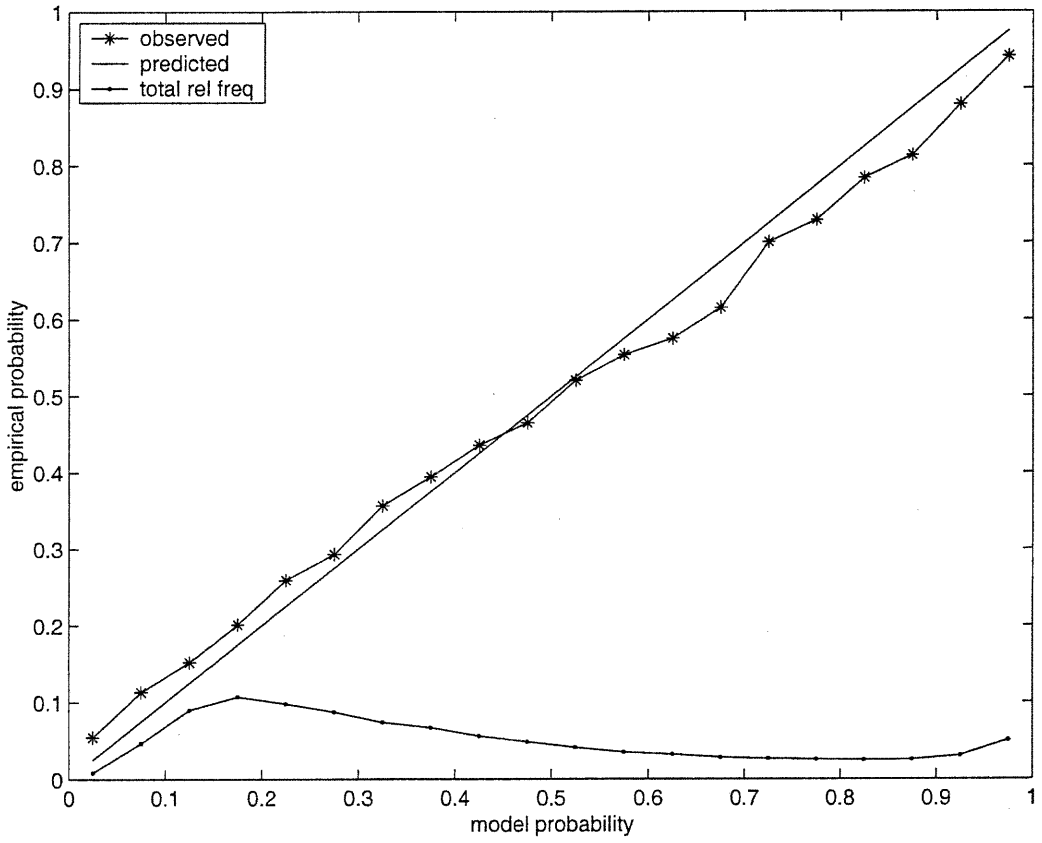

Figure 5.2. Agreement between model and empirical probabilities. Points on the *observed* **curve represent the proportion of times** $\hat{w} = \text{argmax}_w p(w|\mathrm{h}, \mathrm{s})$ **was correct among all** $\hat{w}$ **to which the model assigned a probability in the interval surrounding the point on the x axis. The** *total rel freq* **curve gives the number of predictions in each interval over the total number of predictions made.**

on the test corpus. Specifically, this involved recording, for each token position in the corpus, the probability of the most likely word according to the model, along with whether or not that word matched the actual next word. The results are shown in figure 5.2 in the form of a histogram. As can be seen, the agreement between the model and empirical probabilities is generally very good, though the model has a tendency to underestimate low probabilities and overestimate high ones. Interestingly, the crossover point between these two behaviours occurs almost exactly at .5. Cross-validation smoothing with a uniform distribution did not substantially change this picture. I speculate that this property may be due to the use of maximum entropy, which is supposed to yield inherently smooth models. For the record, the global accuracy of the model on the next-word classification problem was .42, which is not too bad given that the size of the model's vocabulary is over 50,000, and given the existence of a substantial number of out-of-vocabulary tokens which the model automatically gets wrong.

5.3 User Model

The purpose of the user model is to determine how a translator will react to a prediction, and to estimate the resulting cost or benefit. The interaction scenario is as follows. The system displays a prediction, which the translator either reads or ignores and continues to type. If it is read, the translator then decides whether or not to accept it. If so, they[3] use a special command to incorporate it into the text, then erase any wrong characters from the end of the prediction until only a correct prefix remains. This simple procedure minimizes assumptions about the user's pattern-recognition and editing capabilities. The system is free to make another prediction once the current one has been rejected and a subsequent character has been typed,

[3] He or she.

or once it has been accepted and editing is finished.[4]

As described in chapter 1, the user model provides an estimate of the benefit function:

$$B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k) = \sum_{a \in \{0,1\}} p(a|\mathbf{x}, \mathbf{h}, \mathbf{s}, k)\, B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k, a), \tag{5.4}$$

in which the main elements are the probability $p(a|\mathbf{x}, \mathbf{h}, \mathbf{s}, k)$ that a translator will accept or reject a prediction $\mathbf{x}$ whose first k characters are correct; and $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k, a)$, the benefit they derive from doing so. Recall that a is a random variable that takes on the values 1 for acceptance and 0 for rejection. The parameters for both elements were estimated from data collected during an evaluation of the TransType prototype. To give some context, I begin with a brief description of this evaluation; more details are available in [78].

5.3.1 TransType Evaluation

The parameters for the user model were induced from trace data produced by the second of the TransType user trials described in chapter 2.[5] Nine skilled translators took part in this trial, which consisted of three phases:

1. manual typing without TransType suggestions (5-8 minutes)

2. typing with TransType suggestions (15-20 minutes)

3. typing with TransType suggestions augmented by a handcrafted term lexicon (5-8 minutes)

[4] It is assumed that the interface can detect when editing is finished, for instance when leftward motion of the cursor has stopped. In practice it does not have to be very precise in this, because if it makes proposals while the user *is* still editing, they are likely to simply be ignored and thus not incur any cost.

[5] This did not include any of the enhancements described in this thesis.

The 1st and 2nd phases were preceded by familiarization periods to let the user become accustomed to the environment. The texts for these phases were drawn from the Hansard corpus on which TransType's statistical models were trained, and the text for the final phase was a more technical one to allow the effect of the specialized term lexicon to be assessed. All significant activity involving the interface—user actions, and system proposals and responses—was logged and timestamped during all three phases.

Although the average productivity loss was 17% for the evaluation, there was a large variation among different translators, ranging from a minimum of -39% to a maximum of +88%. The latter score was obtained by one translator who was a fairly slow typist, and was the only improvement in productivity recorded. Apart from this, although typing speed also varied considerably across translators, it seems to be only very weakly correlated with gains or losses in productivity.

For measuring the users' raw typing speed, I relied on data from phase 1, as described below. For inducing all other aspects of the user model, I used the combined results of phases 2 and 3.

5.3.2 Acceptance Probability

A model for $p(a|\mathbf{x}, \mathbf{h}, \mathbf{s}, k)$ according to the scenario given at the beginning of this section would ideally take into account whether the user actually reads the proposal before accepting or rejecting it, eg:

$$p(a|\mathbf{x}, \mathbf{h}, \mathbf{s}, k) = \sum_{r \in \{0,1\}} p(a|r, \mathbf{x}, \mathbf{h}, \mathbf{s}, k)\, p(r|\mathbf{x}, \mathbf{h}, \mathbf{s}, k)$$

where r is a Boolean "read" variable. However, this information is hard to extract reliably from the available data; and even if it were obtainable, many of the factors which influence whether a user is likely to read a proposal are extra-textual and therefore not available to the current predictor. Consequently, I chose to model $p(a|\mathbf{x}, \mathbf{h}, \mathbf{s}, k)$ directly.

The first step in deriving the model is to assume that the probability of acceptance is independent of both **h** and **s**. The most problematic of these assumptions is that **h** plays no role, because there is some evidence from the data [78] that users are more likely to accept a suggestion before they start typing a new word. However, as shown below, this does not appear to have a severe effect on the quality of the model.

The second assumption is that the user's propensity to accept a prediction depends directly on what they stand to gain from it, defined in the spirit of the scenario above as the amount by which the length of the correct prefix of **x** exceeds the length of the wrong suffix:

$$g(\mathbf{x}, k) = k - (l - k) = 2k - l,$$

where $g(\mathbf{x}, k)$ is the gain. For example, the gain for the prediction in figure 5.1 would be $2 \times 7 - 8 = 6$. This estimate is best for small k. For instance, it seems reasonable to assign a gain of 10 for erasing the last character of a 12 character proposal, but much less so for erasing the last 45 characters of a 100 character proposal. A mitigating factor is the conservative nature of the editing cost model described below, in which the cost of erasing 45 characters is 45 keystrokes. In practice, most users would tend to use a quicker method, such as by selecting with the mouse, to erase this much text—and would have this possibility in mind when deciding whether or not to accept. In any case, long proposals on the order of 100 characters are relatively rare in the experiments I have performed, so the effects of this problem are likely to be minimal.

Letting l be the length in characters of **x**, the two assumptions above can be summarized as:

$$\begin{aligned} p(a|\mathbf{x}, \mathbf{h}, \mathbf{s}, k) &\approx p(a|\mathbf{x}, k) \\ &\approx p(a|2k - l), \end{aligned}$$

Figure 5.3 shows empirical estimates of $p(a = 1|2k - l)$ for various gain values obtained from the TransType data. There is a certain amount of noise intrinsic

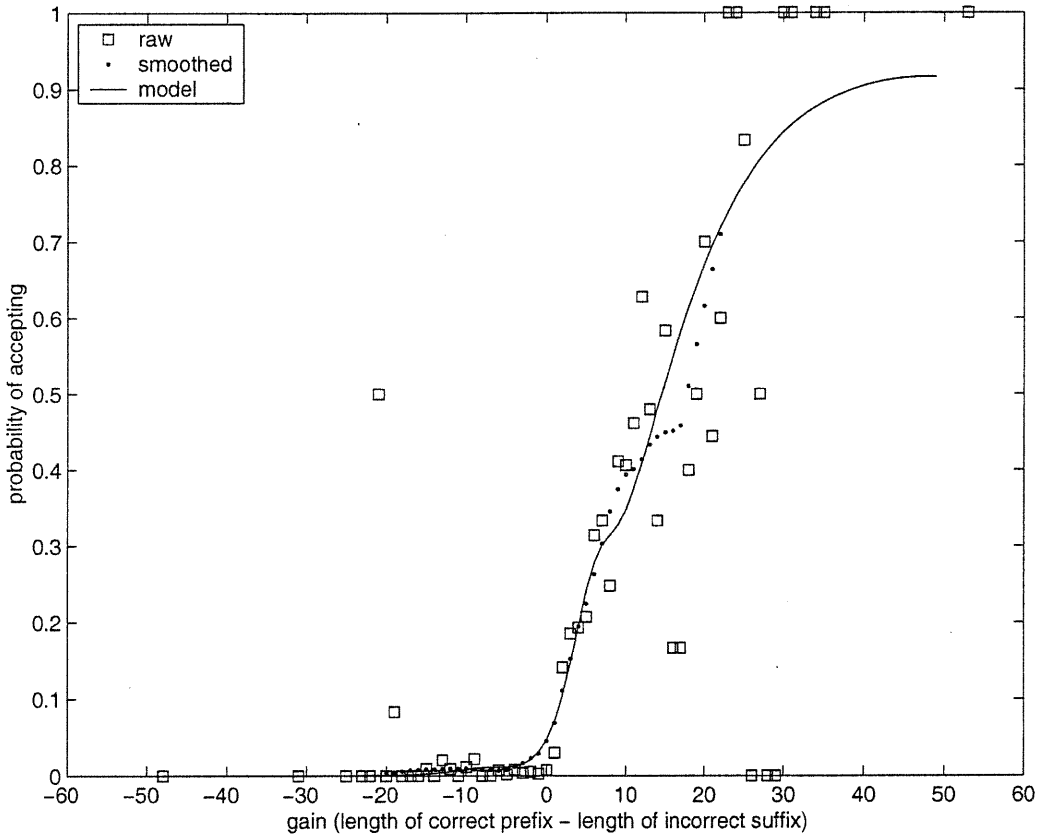

Figure 5.3. Probability that a prediction will be accepted versus its gain.

to the estimation procedure, since it is difficult to determine $\mathbf{x}^*$, and therefore k, reliably from the data in some cases (eg when the user is editing the text heavily). Nonetheless, it is apparent from the plot that gain is a useful abstraction, because the probability of acceptance is very low when it is less than zero and rises rapidly and monotonically as it increases. The points labelled *smoothed* in figure 5.3 were obtained from a sliding-average smoother, and the curve was obtained using two-component Gaussian mixtures to fit the smoothed empirical estimates for $p(\text{gain}|a = 0)$ and $p(\text{gain}|a = 1)$. The model probabilities are taken from the curve at integral gain values. As an example, the probability of accepting the prediction in figure 5.1 would be about .25.

5.3.3 Benefit

As with the acceptance probabilities, I assume that the benefit associated with a prediction $\mathbf{x}$ is a function only of $\mathbf{x}$ itself and k, the length of its longest correct prefix, as well as whether it gets accepted or not. This is broken down as follows:

$$B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k, a) \approx B(\mathbf{x}, k, a) = \begin{cases} T(\mathbf{x}, k) - R_1(\mathbf{x}) - E(\mathbf{x}, k), & a = 1 \\ -R_0(\mathbf{x}), & a = 0 \end{cases}$$

where $T(\mathbf{x}, k)$ is the cost of manually typing the first k characters of $\mathbf{x}$, $R_1(\mathbf{x})$ is the cost of reading and accepting it, $E(\mathbf{x}, k)$ is the edit cost of accepting it and erasing to the end of its first k characters, and $R_0(\mathbf{x})$ is the cost of reading and rejecting it. As in the previous section, read costs are interpreted as expected values with respect to the probability that the user actually does read $\mathbf{x}$, eg, assuming 0 cost for not reading, $R_0(\mathbf{x}) = p(r = 1|\mathbf{x})R'_0(\mathbf{x})$, where $R'_0(\mathbf{x})$ is the unknown true cost of reading and rejecting $\mathbf{x}$.

A natural unit for $B(\mathbf{x}, k, a)$ is the number of keystrokes saved, so all elements of the above equation are converted to this measure. This is straightforward in the case of $T(\mathbf{x}, k)$ and $E(\mathbf{x}, k)$, which are estimated as k and $l - k + 1$ respectively. For

$E(\mathbf{x}, k)$, this corresponds to one keystroke for the command to accept a prediction, and one to erase each wrong character. This assumes that all keys on the keyboard, as well as some chords, require the same amount of effort to access. While this is almost certainly not the case, any inaccuracies it causes are likely to average out over all predictions. Other assumptions are that the user would have typed x_1^k without making mistakes, and that they erase the wrong prefix in a similarly flawless manner. The effects of these seem likely to cancel on average, since they are symmetrical and opposite in sign.[6]

To determine reading costs, I measured the average elapsed time in the TransType data from the point at which a proposal was displayed to the point at which the next user action occurred—either an acceptance or some other command signalling a rejection. Times greater than 5 seconds were treated as indicating that the translator was distracted (or thinking) and were filtered out. As shown in figures 5.4 and 5.5, read times are much higher for predictions that get accepted, reflecting both a more careful perusal by the translator and the fact the rejected predictions are often simply ignored. Here the number of characters read was assumed to include the whole contents of the TransType menu in the case of rejections, and only the proposal that was ultimately accepted in the case of acceptances.[7]

In both cases there appears to be a weak linear relationship between the number of characters read and the time taken to read them, which I modeled using the least squares lines shown in the graphs. (In the case of the rejection model, the line shown is shifted down by the average time required to produce a keystroke when no proposal is displayed, since all we are interested in is the extra time required when a proposal *is* displayed.) Both plots are quite noisy and would benefit from a more sophisticated

[6] In fact, this is slightly conservative, since users are more likely to make errors while typing than while erasing.

[7] Justification for the latter is that the majority of acceptances were via the keyboard, which only permitted the top proposal in the list to be accepted directly.

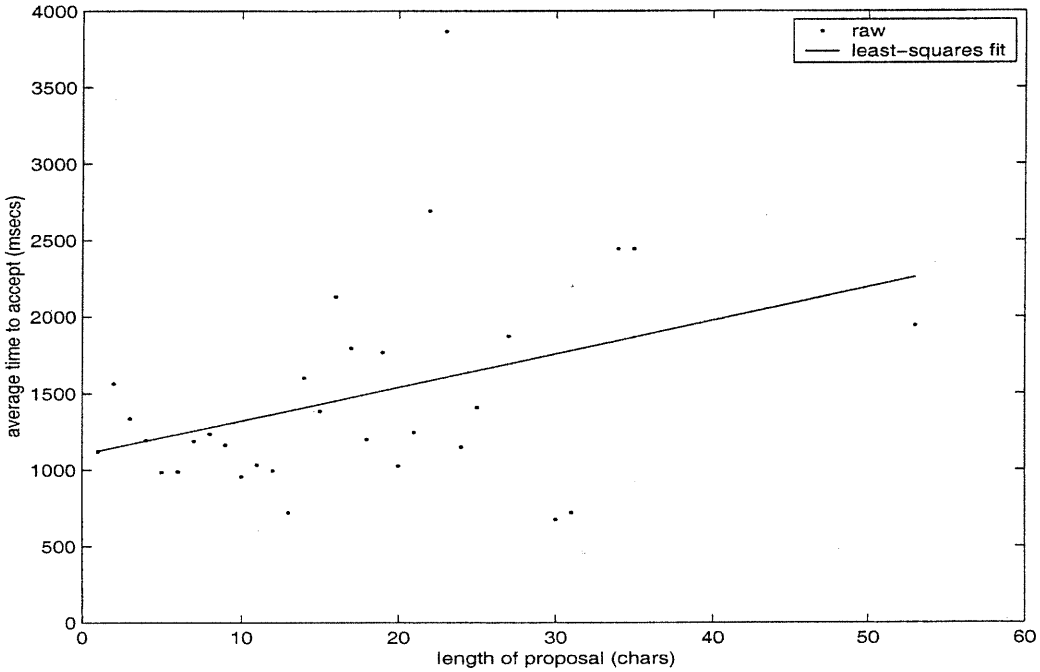

Figure 5.4. Time to read and accept proposals versus their length

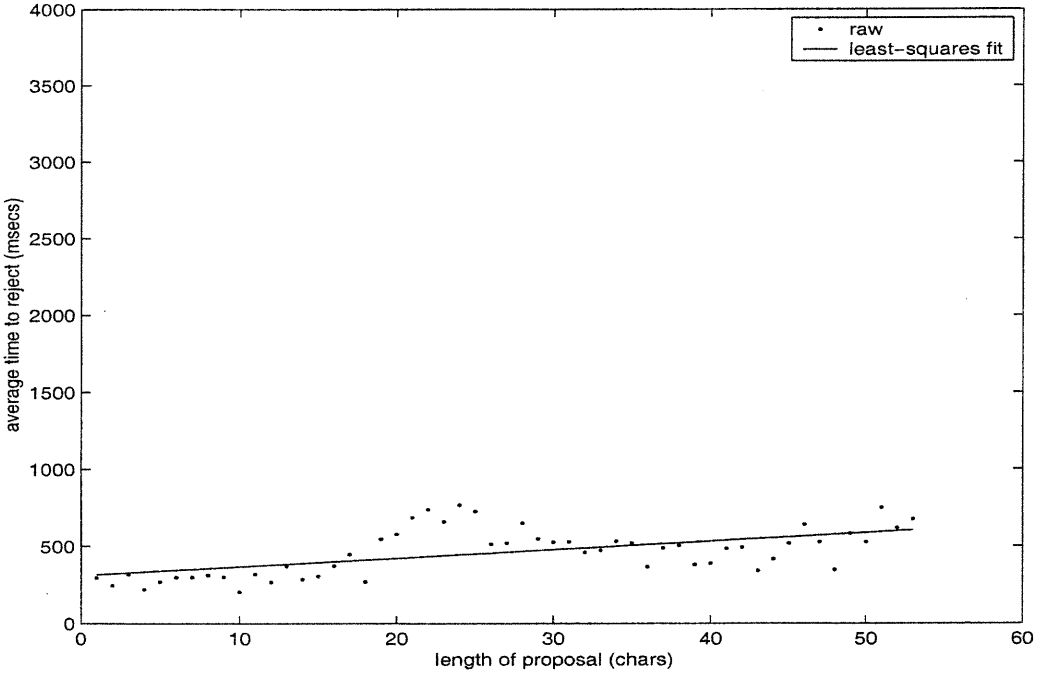

Figure 5.5. Time to read and reject proposals versus their length

psycholinguistic analysis, but they are plausible first approximations.

To convert reading times to keystrokes for the benefit function, I calculated an average time per keystroke based on sections of the trial where translators were rapidly typing and when predictions were not displayed (phase 1). The global average across all translators was 304 milliseconds.[8] This gives an upper bound for the per-keystroke cost of reading and therefore results in a conservative estimate of benefit (compare to, for instance, simply dividing the total time required to produce a text by the number of characters in it). In the figure 5.1 example, the benefit of accepting would be $7-2-4.2 = .8$ keystrokes (7 correct characters minus 2 keystrokes to erase to the end of the correct prefix minus the equivalent of 4.2 keystrokes to read and accept the suggestion); and the benefit of rejecting would be $-.2$ keystrokes. Combining these with the acceptance probability gives an overall expected benefit $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k = 7)$ for this proposal of 0.05 keystrokes, just above the breakeven point.

5.4 Search

The search procedure to find the best prediction $\hat{\mathbf{x}}$ takes place in two main stages, as described in chapter 1: first find the most likely sequence of words $\hat{\mathbf{w}}_m$ for each length m up to a maximum M, then synthesize each into a corresponding character string $\hat{\mathbf{x}}_m$ and output the $\hat{\mathbf{x}}_m$ with highest expected benefit $B(\hat{\mathbf{x}}_m, \mathbf{h}, \mathbf{s})$, or the empty string (ie, no prediction) if all benefits are negative.[9]

Since the second search step is trivial, I focus here on the first step, which aims

[8] The average given in [78] is more than twice as much (650 ms). The difference is due to my exclusion from the average of any gaps between successive keystrokes of more than 5 seconds.

[9] Notice that M word sequences are always generated—there is no stop condition, such as a test for a sentence end. The second search step will ideally eliminate predictions that extend past the end of the target sentence due the their having low probabilities.

at finding $\hat{\mathbf{w}}_m$ according to:

$$\hat{\mathbf{w}}_m = \underset{\mathbf{w}_m}{\operatorname{argmax}}\, p(\mathbf{w}_m|\mathbf{h}, \mathbf{s}).$$

For all the models I describe above, $\hat{\mathbf{w}}_m$ can be found exactly using the Viterbi algorithm [115] in $O(mJ|V|^3)$ time, where J is the length of $\mathbf{s}$ and V is the target vocabulary. The factor $m|V|^3$ comes from the general property of this algorithm for Markov processes of order 3, and J comes from the need to sum contributions from at most J source words when calculating the probability of each target word. That the models for $p(w|\mathbf{h}, \mathbf{s})$ define a Markov process over target words for a given source sentence arises from the Markov nature of the trigram language model, together with the observation that, for a fixed $\mathbf{s}$, the only other context dependence in these models is on the *length* of $\mathbf{h}$. Imposing a prior limit on that length (which is in practice the case with my implementation) reduces this to a finite state process. (See [49] for a more detailed version of this argument.)

It is instructive to compare this result to the one for models used in statistical MT. As discussed in section 3.5.1, Knight [71] has shown that the search problem:

$$\hat{\mathbf{t}} = \underset{\mathbf{t}}{\operatorname{argmax}}\, p(\mathbf{t})p(\mathbf{s}|\mathbf{t})$$

is NP-complete, even when $p(\mathbf{t})$ is uniform and $p(\mathbf{s}|\mathbf{t})$ is IBM1. This is avoided in my approach by essentially turning the translation component around to remove the dependence on $\mathbf{h}$; the SMT-style formulation would be $p(\mathbf{t})^\alpha p(\mathbf{t}|\mathbf{s})$, where α is a log-linear scaling factor on the language component. Fast dynamic-programming based heuristics have been proposed for SMT search, but even with fairly stringent restrictions on word re-orderings, their complexity is still polynomial in J (eg $O(J^4|V|^3)$ in [108]).

5.4.1 Heuristics

Cubic complexity in the size of a target vocabulary on the order of 10^4 is too expensive for generating proposals in real time, so I used two heuristics to speed up

the search. The first is the active vocabulary technique used in TransType: limit the search to some subset V_A of the vocabulary that has high probability given $\mathbf{s}$. In TransType, this was generated by taking the n most likely words according to the IBM1 probabilities:

$$p(w|\mathbf{s}) = \sum_{j=0}^{J} p(w|s_j)/(J+1).$$

I took this approach when using the 3G+IBM models for searching. For the MEMD models, I simply placed into V_A all target words connected by a word-pair feature to some word in $\mathbf{s}$. This was feasible due to the small size of the feature sets for these models.

The first word position ($m = 1$) is a special case due to the requirement to match a possible word prefix u. To enforce this, I created a corresponding active vocabulary V_{A1} containing only those words in V_A that begin with u. If there are no such, all words in V are tried. If there are still no matches, the UNK word is considered to match u (in its entirety—that is, u is considered to be a complete word) in order to allow for searches beyond $m = 1$.

The second heuristic is to use the "beam" technique commonly used with the Viterbi algorithm. This involves pruning any potential extensions to a current path (hypothesized partial sequence of target words) that have probability lower than $\hat{P}/f$, where $\hat{P}$ is the probability of the best extension to the path, and f is the beam factor. I found that $f = 10$ gave a good balance between speed and accuracy. This value is used in all experiments described below.

Some empirical search timings are shown in table 5.1. These compare the MEMD2B and 3G+IBM2 configurations described in the next section over various sequence lengths M, on the *Realistic* user simulation with linear estimates for string prefix probabilities. The MEMD2B model contains 50k word-pair features and is used with all "triggered" target words in the active vocabulary; the 3G+IBM2 contains 1M word-pair features and is used with a fixed 500-word active vocabulary. The average

M	average times		max times	
	MEMD2B	3G+IBM2	MEMD2B	3G+IBM2
1	0.0012	0.0008	0.01	0.01
2	0.0038	0.0134	0.23	0.46
3	0.0097	0.0647	0.51	1.75
4	0.0184	0.1362	0.55	2.46
5	0.0285	0.2004	0.57	2.70

Table 5.1. Approximate times in seconds to generate predictions of maximum word sequence length M, on an 1.2GHz processor, for MEMD and 3G+IBM models.

times for the MEMD model are acceptable up to $M = 5$, but those for 3G+IBM2 only up to at most $M = 3$. The maximum times for both models would cause perceptible delays for M greater than 1, but given the low average times these seem to be rare. I suspect that it would be easy to develop heuristics to detect these cases a priori and revert to a fallback strategy to avoid causing the user to wait.

5.5 Experiments

I evaluated the predictor using the training and test setup described in the previous chapter, except with a smaller test set consisting of only the first three Hansard files in the test corpus. This consisted of 5,020 sentence pairs and approximately 100k words in each language. As before, all letter case distinctions are ignored; however, all other differences between strings, including whitespace, were considered significant.

To get an idea of the effects of different components of the user model, I defined three different, increasingly realistic, simulation profiles. *Rational* simulates a user who takes no time to read suggestions and who accepts all and only predictions having positive benefit—this gives an upper bound on the estimated keystroke sav-

ings. *RationalReader* is like rational but with read costs added. Finally, *Realistic* matches the user model described in section 5.3. For each simulation, the predictor was given access to the corresponding user model, and it optimized expected benefits accordingly.

The results in terms of keystroke reductions for each simulation are shown in tables 5.2, 5.3, and 5.4. For all simulations, I tried a number of different predictor configurations, listed in the *benefit estimator* column:

- fixed: always propose the most likely M tokens, ignoring benefit
- at-end: the prefix probability estimator described in section 5.2.3
- linear: the linear prefix probability estimator described in section 5.2.2
- exact: the "exact" prefix probability estimator described in section 5.2.1
- best: choose the m that maximizes the actual benefit to the user (by cheating and looking at $\mathbf{x}^*$)

For the Rational and RationalReader simulations, I used only the best MEMD2B model described in the previous chapter, with 50,000 word-pair parameters, and 10×4000 position parameters. For the Realistic simulation, I compared this to the best linear combination model 3G+IBM2, with 1M word-pair parameters.

5.5.1 Summary of Results

The results can be characterized along several axes. First, there are large productivity drops due to reading costs and probabilistic acceptance, as evidenced by the differences between the Rational and RationalReader simulations, and the RationalReader and Realistic simulations. The biggest cost is due to reading, which lowers the best possible keystroke reduction by almost 50% for $M = 5$, and by even more than this

benefit estimator	M 1	2	3	4	5
fixed	56.5	49.6	39.8	31.4	24.9
at-end	50.9	54.7	54.0	53.6	53.5
linear	56.5	54.4	52.7	52.2	52.0
exact	48.6	53.5	51.8	51.1	50.9
best	56.5	63.1	64.9	65.4	65.6

Table 5.2. *Rational* **simulation, using MEMD2B model. Numbers give estimated percent reductions in keystrokes.**

benefit estimator	M 1	2	3	4	5
fixed	-2.7	11.0	6.8	-3.3	-14.3
at-end	3.0	9.7	9.4	7.9	7.3
linear	13.1	17.4	16.3	15.8	15.6
exact	11.7	17.8	17.2	16.4	16.1
best	14.9	25.6	31.3	33.8	34.9

Table 5.3. *RationalReader* **simulation, using MEMD2B model. Numbers give estimated percent reductions in keystrokes.**

benefit	M				
estimator	1	2	3	4	5
fixed	-8.5	-0.4	-3.60	-11.6	-20.8
at-end	-3.8	0.92	0.3	-2.0	-3.2
linear	6.1	9.40	8.8	8.1	7.8
exact	5.3	10.10	**10.7**	10.0	9.7
best	7.9	17.90	24.5	27.7	29.2

benefit	M				
estimator	1	2	3	4	5
fixed	**-11.5**	-9.3	-15.1	-22.0	-28.2
at-end	-6.0	-5.5	-7.8	-9.7	-11.7
linear	4.3	5.9	6.4	**6.5**	6.5
exact	3.0	4.3	5.0	5.2	5.2
best	6.2	12.1	15.4	16.7	17.3

Table 5.4. *Realistic* **simulations, using the MEMD2B model (top table), and the 3G+IBM2 model (bottom table). Numbers give estimated percent reductions in keystrokes.**

for lower M, due to the fixed overhead involved in reacting to a proposal. Having acceptance be probabilistic results in a further drop of about 15% for $M = 5$, rising to almost 50% for $M = 1$. The greater proportional drop for shorter predictions is in this case due to lower probabilities of acceptance for smaller gains.

Focussing next on the Realistic simulations for different models, it is clear that the MEMD2B model significantly[10] outperforms the 3G+IBM2 combination, obtaining a maximum keystroke reduction of 10.7% compared to 6.5% for regular search; 29.2% compared to 17.3% for the *best* configuration; and producing systematically better results for every other configuration tested.

The figure of -11.5% in bold corresponds roughly to the TransType configuration, and corroborates the validity of the simulation. Although the drop observed with real users was lower at about 20% (= 17% reduction in speed), there are many many differences between experimental setups that could account for the discrepancy.[11]

Looking at the results for the Realistic simulation using the MEMD2B model, it is clear that estimating expected benefit is a much better strategy than making fixed-word-length proposals, since the latter causes productivity to drop for all values of M. It is also much better than just choosing among different m on the basis of each prediction's length and overall probability, as shown by the results for the *at-end* estimator. In general, making "exact" estimates of string prefix probabilities works better than a linear approximation, but the difference is fairly small.[12]

---

[10] The t test score over a sample of 1,272 sentences was 7.7915 for regular search, so the difference is highly statistically significant (below 0.0005).

[11] One that seems especially significant is that the productivity loss in phase 3 of the TransType trials was much higher, due to using a text from a different domain. If this phase is excluded, the loss is only 10%, closer to my results. Other differences are: the use of different (Hansard) test corpora; and the use in TransType of letter case handling, unknown-word handling, a large French dictionary, position-dependent interpolation parameters for I3G+IBM2, and a unit lexicon.

[12] I do not know why the result for single-word predictions is worse for exact estimates; it may be connected to the use of the V_A correction factor c described in section 5.2, and not making a

The main disappointment in the MEMD2B Realistic results is that fact that performance peaks when $M = 3$ rather than continuing to improve as the predictor is allowed to consider longer word sequences. Since the predictor knows the exact benefit for each value of k, the only possible cause for this is that the estimates for $p(\hat{\mathbf{w}}_m|\mathbf{h}, \mathbf{s})$ become worse with increasing m. Unfortunately, because the magnitude of $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k)$ also generally increases with the length of $\mathbf{x}$, the effect of any errors will tend to be amplified at larger m.

The cause of increasing inaccuracy with length is the difference between $p(w|\mathbf{h}, \mathbf{s})$ and $p(w|\mathbf{h}, \mathbf{w}', \mathbf{s})$, where $\mathbf{w}'$ is some non-empty prefix of $\hat{\mathbf{w}}_m$. Both probabilities are generated by the same model, but only in the former case is there any guarantee that all of the conditioning information represents good target text. Significantly, the performance levels off at three words, just as the search loses direct contact with $\mathbf{h}$ through the trigram. Fortunately though, the *potential* benefit from longer predictions (as measured by the *best* results in the table) continues to increase. This means that there may still be some correlation between model and true probabilities that could be exploited (if there were no correlation, the search would essentially be producing nonsensical results at this stage).

About the simplest form such a correlation could take is a systematic over- or underestimation of $p(\hat{\mathbf{w}}_m|\mathbf{h}, \mathbf{s})$. To test whether this was the case, I used modified probabilities of the form $\lambda_m p(\hat{\mathbf{w}}_m|\mathbf{h}, \mathbf{s})$, where λ_m is a length-specific correction factor, tuned so as to optimize keystroke reduction on a cross-validation corpus. The values of λ_m begin around 1, and gradually decrease to a minimum of about .5 for λ_5, indicating that the model is indeed overestimating the probabilities of longer sequences. The corresponding keystroke reduction results on the test corpus are shown in table 5.5 for the MEMD2B Realistic configuration. In this case, there is a slight but monotonic rise in performance with M for both string probability estimators tested. To verify

parallel adjustment for the sum over suffixes when calculating string prefix probabilities.

benefit	M				
estimator	1	2	3	4	5
thresh	-2.2	1.3	2.9	3.2	3.2
linear	6.4	10.3	11.4	11.3	11.3
exact	5.8	10.7	12.0	12.5	**12.6**
best	7.9	17.90	24.50	27.7	29.2

Table 5.5. *Realistic* **simulations using the MEMD2B model, with corrected probability estimates. Numbers give estimated percent reductions in keystrokes.**

that this was not just due to the cross-validation tuning, I also established optimum probability thresholds for each length, and used these together with a strategy of proposing the longest prediction that exceeded the corresponding threshold. The results are shown in the line marked *thresh*, and show that there is little to be gained with this approach.

The best keystroke reduction gain recorded in these experiments for the Realistic simulation is 12.6% with MEMD2B and corrected probability estimates. This is only slightly higher than the 10.7% obtained with the plain estimates, but its value to the translator is likely to be somewhat more than is indicated by the score difference. Figure 5.6 shows the distribution of proposals by length for $M = 5$, with plain and corrected estimates, compared to the optimal profile. With the plain estimates, the user is clearly being presented with many more wrong proposals than with the corrected estimates, for which the distribution is quite similar to the optimal. Although the current user model does not take past prediction accuracy into account, it seems obvious that being deluged with incorrect predictions will lessen user confidence in the tool and therefore lower the probability that future proposals will be accepted.

Examples of predictions made for various values of M are shown in appendix A.

m	plain	corrected	best
0	231437	333309	219761
1	6704	14624	13433
2	25597	41751	17888
3	41852	28843	9965
4	33393	7411	4769
5	19953	2104	4217

Table 5.6. Number of proposals by length, for *Realistic* simulation with the MEMD2B model and $M = 5$.

5.6 Conclusion and Future Work

I have described an approach to text prediction for translators that tries to take user characteristics into account when deciding what proposal to make in a given context—or whether to make a proposal at all. The main conclusion is that a tool based on this principle would have the potential to benefit a "typical translator" by reducing typing time by slightly over 10%. Since this is a fairly small improvement, it is clear that not all translators would find this kind of tool useful. However, because I have been careful to make conservative estimates of benefit, I feel that some non-negligible proportion of translators would indeed find it useful. Of course, the only way to be certain would be to do an evaluation with real users.

Another conclusion is that the maximum-entropy based models described in the previous chapter outperform a linear combination of a trigram and the IBM model 2 for the prediction task, resulting in a 65% relative improvement in keystroke reduction. They also allowed significantly faster searches in my implementation (due to having smaller feature sets).

5.6.1 Future Work

There are many possibilities for extending this work. Apart from improvements to the model for $p(w|\mathbf{h}, \mathbf{s})$, as described in previous chapters, all three main predictor components offer the potential for enhancement: the prefix model $p(k|\mathbf{x}, \mathbf{h}, \mathbf{s})$ derived from $p(w|\mathbf{h}, \mathbf{s})$, the user model $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k)$, and the search procedure for finding the best prediction.

Estimates for prefix probabilities could be improved by using a richer character-sequence model that would relax the assumption of a one-to-one correspondence between token sequences and strings. This would have to handle uppercase letters in order to be completely realistic. Since different underlying words can be expressed in different ways by different authors (eg *machine translation* as "machine translation" or "Machine Translation"), a very useful feature of the character model would be the ability to learn a preferred spelling or capitalization pattern. The problem of how to compensate for the use of an active vocabulary also merits further study, as do approximations like the log-linear interpolation suggested above to avoid having to sum over the whole vocabulary at each string position. Finally, the ad hoc calibration of $p(\hat{\mathbf{w}}_m|\mathbf{h}, \mathbf{s})$ for sequences of various lengths could be replaced by an adaptive smoothing approach, which would calibrate the model according to its performance on recent text.

Many aspects of the user model could be improved as well. The model for acceptance probability could be made more accurate by giving it information about past prediction performance (including the number of times the current proposal has already been displayed), and other extra-textual factors. One way to take advantage of this would be to incorporate explicit read probabilities as suggested in section 5.3.2, in combination with a notion of "typing inertia" that makes people less likely to pay attention to proposals while in the middle of typing a word. The models for read costs could be improved by basing them on sounder psycholinguistic principles, not

to mention cleaner response-time data. The edit model could be enriched to include alternate editing strategies, particularly for long proposals. Finally, given the large variation among translators, it makes sense to have user model adapt to the behaviour and preferences of its current user, starting with an initial universal model like the one described in this chapter and gradually shaping it to conform to what it has observed. The data for this would be easy to glean in real time from an instrumented interface like TransType's.

The search algorithm could also be improved to look more directly for $\hat{\mathbf{x}}$—a capability which would become more valuable as the user model improved. One simple possibility would be to generate a small set of most-likely word sequences at each length, rather than just one, and optimize the benefit over these.

Concerning the overall strategy adopted for the predictor, I feel that the idea of creating explicit user models to guide the behaviour of interactive systems is likely to have applications in other areas of NLP besides translator's tools. For one thing, most of the approach described here carries over more or less directly to monolingual text prediction, which is an important tool for the handicapped [28, 38]. Other possibilities include virtually any application where a human and a machine communicate through a language-rich interface.

Chapter 6

CONCLUSION AND FUTURE WORK

This thesis deals with several aspects of text prediction for translators, seen as a form of interactive machine translation. The work presented extends previous work on text prediction (within the TransType project) in several ways: using more accurate maximum entropy/minimum divergence translation models; using a search procedure to explore the space of multi-word predictions rather than just completing the next word or lexicalized unit; and taking the expected benefit to the user into account when generating predictions so as to avoid swamping translators with a large number of useless proposals.

6.1 Contributions

My contributions can be divided into three main areas:

- Interactive MT for translators. I have proposed a new *target-text mediated* style of IMT based on the use of the target text as the medium of interaction with the computer. Text prediction is one of the many possibilities this approach encompasses.

- Efficient translation models for text prediction. I have defined two new MEMD models that support linear-time searches. Compared to directly analogous linear combinations of a trigram language model and the classical IBM models 1 and 2, these models achieve approximately 50% lower test-corpus perplexity, despite requiring orders of magnitude fewer parameters. For selecting MEMD features, I have shown that a simple algorithm based on word-pair gains within the IBM model 1 outperforms a general feature-selection algorithm due to Printz.

- User-centric text prediction. I have described an approach to text prediction that explicitly seeks to maximize the benefit to the user according to a model of user characteristics. In simulated results, this method reduces the time required to type a translation by over 10% when using the best of the new translation models to make real-time predictions of up to 5 words in length.

6.2 Future Work

There are many ways in which this work could be built upon, as described in the final sections of chapters 3, 4, and 5. In this section I recapitulate some of the more promising of these ideas, and discuss a few other possibilities.

The most significant gains will probably come from improved models for $p(w|\mathbf{h}, \mathbf{s})$, which are the core of the predictor. The modeling approach I have taken for this application is to start with very simple and efficient formulations, then gradually add more sophistication and predictive accuracy without losing too much efficiency in the process. An opposite approach would be to start with sophisticated noisy-channel SMT models such as the classical IBM models 3–5 and their recent enhancements, then try to develop custom search heuristics to permit real time prediction. Some evidence in favour of this approach is to be found in by recent work by Germann et al [54], where the model, rather than the search procedure, was found to be responsible for the majority of translation errors—this may indicate that more radical heuristics than those used in SMT might sill leave translation quality adequate for prediction. However, my approach has the advantage of simplicity, of making it easier to obtain true probability estimates for benefit estimation.

The most obvious way to improve the MEMD models I have described is to add features that connect w to $\mathbf{h}$ through $\mathbf{s}$, for instance by reducing the contributions of source words that have already been translated, as described in chapter 4. This of course would destroy the models' Markov property, but it would not preclude the use

of efficient heuristics based on a modified version of the current Viterbi beam search. An advantage to this modeling framework over the classical IBM one is that the lack of latent variables makes the models *far* simpler to understand and work with, obviating for instance the specialized heuristics that are necessary to do EM-based training with the former. However, latent variables are an obvious way to model the hidden connections between the source and target texts, so it may also be worthwhile to explore their use within the MEMD framework, as suggested by [74, 119].

An aspect of the maximum entropy technique that could be further exploited is its ability to combine information from different sources. One possibility here is the integration of the unit lexicon technique [76] and the unknown-word handling technique [49] used in TransType. A related issue is the use of preprocessing and external knowledge sources. In the interest of simplicity, of facilitating comparisons, and of maintaining language independence, I have avoided any processing of the inputs to the models beyond tokenization and lettercase mapping. There are many other steps that would potentially be useful for either reducing ambiguity or combatting data sparseness, such as lemmatization, tagging, word-sense disambiguation, grammatical regularization, etc. External knowledge sources such as monolingual and bilingual dictionaries also have potential. All these techniques would have to be used selectively, but the ME framework would make it easy to integrate them into the current models.

Another major opportunity for improvement is offered by the presence of the surrounding context H and S, which could be used to adapt $p(w|\mathbf{h}, \mathbf{s})$ to the current text and to the preferences of its current user. Since there has been no research on this question to date, it is not clear how much is to be gained by it, but there seem to be many interesting possibilities such as being able to learn the translation of new words and terms from the translator, or to learn which of many permissible translations is preferred in the current context.

Apart from improvements to the models for predicting text, it is clear that many

improvements could also be made to the user model. A simple example is to build in some notion of the significance of words (as opposed to just characters) for both reading and editing, which is completely ignored in the current version. Other possibilities including adaptation have been discussed at the end of chapter 5. As mentioned earlier, however, further development should probably await more user-based research on interfaces for prediction.

Taking a step back from the prediction problem as it has been defined here, there are some other ways of using the basic predictive capability that could be explored, such as waiting for a prompt from the user to display proposals, or waiting until the user has paused for a certain period of time, or displaying the results of longer and longer searches progressively until stopped by the user. Many of the techniques developed here could be used in these scenarios with little modification. Taking another step back, there are many other interesting possibilities inherent in the idea of TTMIMT, such as the editing mode described in chapter 2, that could rely on similar techniques. These and other ideas will be investigated in the upcoming TransType 2 project, recently begun under the auspices of the European Commission's 5th Framework programme.

REFERENCES

[1] *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)*, Hong Kong, October 2000.

[2] Yaser Al-Onaizan, Jan Curin, Michael Jahr, Kevin Knight, John Lafferty, Dan Melamed, Franz-Josef Och, David Purdy, Noah A. Smith, and David Yarowsky. Statistical machine translation: Final report, JHU workshop 1999. Technical report, The Center for Language and Speech Processing, The Johns Hopkins University, www.clsp.jhu.edu/ws99/projects/mt/final_report, 1999.

[3] Hiyan Alshawi, Srinivas Bangalore, and Shona Douglas. Automatic acquisition of hierarchical transduction models for machine translation. In COLING-98 [37], pages 41–47.

[4] J.C. Amengual, J.M. Benedi, F. Casacuberta, A. Castano, A. Castellanos, V. M. Jimenez, D. Llorens, A. Marzal, M. Pastor, F. Prat, E. Vidal, and J. M. Vilar. The EUTrans-I speech translation system. *Machine Translation*, 2001. To appear.

[5] Doug Arnold. Why translation is difficult for computers. In H.L. Somers, editor, *Computers and Translation: a handbook for translators.* John Benjamins, 2000.

[6] Doug Arnold, Louisa Sadler, and R. Lee Humphreys. Evaluation: An assessment. *Machine Translation*, 8:1–24, 1993.

[7] *Proceedings of the Advanced Research Projects Research Agency (ARPA) Workshop on Machine Translation Evaluation*, 1994.

[8] L. R. Bahl, J. K. Baker, F. Jelinek, and R. L. Mercer. Perplexity: a measure of difficulty of speech recognition tasks. In *94th Meeting of the Acoustical Society of America*, Miami, December 1977.

[9] Lalit R. Bahl, Frederick Jelinek, and Robert L. Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, PAMI-5(2):179–191, March 1983.

[10] Srinivas Bangalore and Guiseppe Riccardi. A finite-state approach to machine translation. In *Proceedings of the 2nd Meeting of the North American American Chapter of the Association for Computational Linguistics*, pages 135–142, Pittsburgh, June 2001.

[11] Y. Bar-Hillel. The present status of automatic translation of languages. *Advances in Computers*, 1:91–163, 1960.

[12] Yoshua Bengio, Rejean Ducharme, and Pascal Vincent. A neural probabilistic language model. In *Advances in Neural Information Processing Systems*, volume 13, 2001.

[13] A. Berger, P. Brown, S. Della Pietra, V. Della Pietra, J. Gillett, J. Lafferty, H. Printz, and L. Ures. The Candide system for machine translation. In ARPA-MT94 [7], pages 157–163.

[14] Adam Berger and Harry Printz. A comparison of criteria for maximum entropy / minimum divergence feature selection. In *Proceedings of the 3nd Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 97–106, Granada, Spain, 1998.

[15] Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A

Maximum Entropy approach to Natural Language Processing. *Computational Linguistics*, 22(1):39–71, 1996.

[16] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford, 1995.

[17] H. Blanchon. Interactive disambiguation of natural language input: a methodology and two implementations for French and English. In *Proceedings of IJCAI 97*, volume 2, pages 1042–1047, Nagoya, Japan, August 1997.

[18] Hervé Blanchon. Problèmes de désambiguïsation interactive et TAO personnelle. In *L'environnement Traductionnel*, Journées scientifiques du Réseau thématique de recherche "Lexicologie, terminologie, traduction", pages 31–48, Mons, April 1991.

[19] Christian Boitet. Towards personal MT. In COLING-90 [34], pages 30–35.

[20] Christian Boitet and Hervé Blanchon. Multilingual dialog-based MT for monolingual authors: the LIDIA project and a first mockup. *Machine Translation*, 9(2):99–132, 1995.

[21] J. Brousseau, C. Drouin, G. Foster, P. Isabelle, R. Kuhn, Y. Normandin, and P. Plamondon. French speech recognition in an automatic dictation system for translators: the TransTalk project. In *Eurospeech 95*, pages 193–196, Madrid, Spain, September 1995.

[22] Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Frederick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85, June 1990.

[23] Peter F. Brown, Stephen A. Della Pietra, Vincent Della J. Pietra, and Robert L. Mercer. The mathematics of Machine Translation: Parameter estimation. *Computational Linguistics*, 19(2):263–312, June 1993.

[24] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, John D. Lafferty, and Robert L. Mercer. Analysis, statistical transfer, and synthesis in machine translation. In TMI-92 [109].

[25] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, Jennifer C. Lai, and Robert L. Mercer. An estimate of an upper bound for the entropy of English. *Computational Linguistics*, 18(1), 1991.

[26] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. A statistical approach to sense disambiguation in machine translation. In *Proceedings of the Fourth DARPA workshop on Speech and Natural Language*, 1991.

[27] Ralf D. Brown and Sergei Nirenburg. Human-computer interaction for semantic disambiguation. In COLING-90 [34], pages 42–47.

[28] Alice Carlberger, Johan Carlberger, Tina Magnuson, Sira E. Palazuelos-Cagigas, M. Sharon Hunnicutt, and Santiago Aguilera Navarro. Profet, a new generation of word prediction: an evaluation study. In *Proceedings of the 2nd Workshop on NLP for Communication Aids*, Madrid, Spain, July 1997.

[29] Stanley F. Chen and Joshua T. Goodman. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Computer Science Group, Harvard University, 1998.

[30] M. Chevalier, J. Dansereau, and G. Poulin. TAUM-METEO: description du système. Technical report, TAUM, Université de Montréal, 1978.

[31] Larry G. Childs. Alpnet terminology tools. In *Proceedings of the 30th Annual Conference of the American Translator's Association*, pages 455–461, Washinton, D.C., October 1989.

[32] K. Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the 2nd Conference on Applied Natural Language Processing (ANLP)*, Austin, Texas, 1988.

[33] Kenneth W. Church and Eduard H. Hovy. Good applications for crummy machine translation. *Machine Translation*, 8:239–258, 1993.

[34] *Proceedings of the International Conference on Computational Linguistics (COLING) 1990*, Helsinki, Finland, August 1990.

[35] *Proceedings of the International Conference on Computational Linguistics (COLING) 1992*, Nantes, France, August 1992.

[36] *Proceedings of the International Conference on Computational Linguistics (COLING) 1996*, Copenhagen, Denmark, August 1996.

[37] *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics (ACL) and 17th International Conference on Computational Linguistics (COLING) 1998*, Montréal, Canada, August 1998.

[38] Ann Copestake. Some experiments on statistical word prediction for a speech prosthesis. In *Proceedings of the 1st Workshop on NLP for Communication Aids*, Dundee, Scotland, September 1996.

[39] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., New York, NY, 1991.

[40] Ido Dagan, Kenneth W. Church, and William A. Gale. Robust bilingual word alignment for machine aided translation. In *Proceedings of the 1st ACL Workshop on Very Large Corpora (WVLC)*, Columbus, Ohio, 1993.

[41] John J. Darragh and Ian H. Witten. *The Reactive Keyboard.* Cambridge University Press, 1992.

[42] J.N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *Annals of Mathematical Statistics*, 43:1470–1480, 1972.

[43] S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. Technical Report CMU-CS-95-144, CMU, 1995.

[44] William C. Ogden et al. Cibola: A translator's support system. Technical Report MCCS-95-285, Computing Research Laboratory at New Mexico State University, 1995.

[45] Eurolang Optimizer product description. www.lant.be/eurolang_details.html, 1998.

[46] George Foster. Incorporating position information into a Maximum Entropy / Minimum Divergence translation model. In *Proceedings of the 4th Computational Natural Language Learning Workshop (CoNLL)*, Lisbon, Portugal, September 2000. ACL SigNLL.

[47] George Foster. A Maximum Entropy / Minimum Divergence translation model. In ACL-00 [1].

[48] George Foster, Pierre Isabelle, and Pierre Plamondon. Word completion: A first step toward target-text mediated IMT. In COLING-96 [36].

[49] George Foster, Pierre Isabelle, and Pierre Plamondon. Target-text Mediated Interactive Machine Translation. *Machine Translation*, 12:175–194, 1997.

[50] George Foster, Philippe Langlais, and Guy Lapalme. User-friendly text prediction for translators. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Philadelphia, PA, 2002.

[51] R. Frederking, S. Nirenburg, D. Farwell, S. Helmreich, E. Hovy, K. Knight, S. Beale, C. Domanshnev, D. Attardo, D Grannes, and R. Brown. Integrating translations from multiple sources within the Pangloss Mark III machine translation system. In *Proceedings of the First AMTA Conference*, pages 73–80, 1994.

[52] Robert Frederking, Dean Grannes, Peter Cousseau, and Sergei Nirenburg. An MAT tool and its effectiveness. In *DARPA HLT Workshop*, Princeton, NJ, 1993.

[53] Ismael García-Varea, Francisco Casacuberta, and Hermann Ney. An iterative, DP-based search algorithm for statistical machine translation. In ICSLP-98 [60], pages 1135–1138.

[54] Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. Fast decoding and optimal decoding for machine translation. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL)*, Toulouse, July 2001.

[55] Xiuming Huang. A machine translation system for the target language inexpert. In COLING-90 [34], pages 364–367.

[56] John Hutchins. The development and use of machine translation systems and computer-based translation tools. In *International Conference on Machine*

Translation & Computer Language Information Processing, pages 1–16, Beijing, June 1999.

[57] John Hutchins. Retrospect and prospect in computer-based translation. In *Machine Translation Summit VII*, pages 30–34, Kent Ridge Digital Labs, Singapore, September 1999.

[58] W. J. Hutchins. *Macine Translation: Past, Present, Future.* Ellis Horwood, 1986.

[59] *Proceedings of the 5th International Colloquium on Grammatical Inference (ICGI)*, Lisbon, Portugal, September 2000.

[60] *Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP) 1998*, Sydney, Australia, December 1998.

[61] Pierre Isabelle. The state of machine translation in 1996. Invited report prepared for the National Research Council of the U.S.A., 1996.

[62] E. T. Jaynes. *Probability Theory: The Logic of Science.* Unpublished Manuscript, 1996. ftp://bayes.wustl.edu/.

[63] E.T Jaynes. Information theory and statistical mechanics. *Physical Review*, 5(106):620–630, 1957.

[64] F. Jelinek and R. L. Mercer. Interpolated estimation of Markov source parameters from sparse data. In E. S. Gelsema and L. N. Kanal, editors, *Pattern Recognition in Practice.* North-Holland, Amsterdam, 1980.

[65] F. Jelinek, B. Merialdo, S. Roukos, and M. Strauss. A dynamic language model for speech recognition. In *Fourth DARPA Speech and Natural Language Work-*

shop, pages 293–295, Pacific Grove, California, February 1991. Morgan Kaufmann.

[66] Frederick Jelinek. *Statistical Methods for Speech Recognition.* MIT Press, 1997.

[67] Frederick Jelinek, Robert L. Mercer, and Salim Roukos. Principles of lexical language modeling for speech recognition. In Sadaoki Furui and M. Mohan Sondhi, editors, *Advances in Speech Signal Processing*, pages 651–699. Marcel Dekker, New York, 1992.

[68] Martin Kay. The MIND system. In R. Rustin, editor, *Natural Language Processing*, pages 155–188. Algorithmics Press, New York, 1973.

[69] Martin Kay. The proper place of men and machines in language translation. Technical Report CSL-80-11, XEROX PARC, October 1980.

[70] Martin Kay. It's still the proper place. *Machine Translation*, 12(1–2):35–38, 1997.

[71] Kevin Knight. Decoding complexity in word-replacement translation models. *Computational Linguistics, Squibs and Discussion*, 25(4), 1999.

[72] Marian Kugler, Khurshid Ahmad, and Gregor Thurmair. Translator's workbench. Technical report, EEC Esprit Project, 1995.

[73] Roland Kuhn and Renato De Mori. A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 12(6):570–583, June 1990.

[74] John D. Lafferty. Gibbs-markov models. In *Computing Science and Statistics: Proceedings of the 27th Symposium on the Interface.* Interface Foundation, 1995.

[75] Ph. Langlais and G. Foster. Using context-dependent interpolation to combine statistical language and translation models for interactive MT. In *Content-Based Multimedia Information Access (RIAO)*, Paris, France, April 2000.

[76] Philippe Langlais, George Foster, and Guy Lapalme. Unit completion for a computer-aided translation typing system. *Machine Translation*, 15(4):267–294, December 2000.

[77] Philippe Langlais, George Foster, and Guy Lapalme. Integrating bilingual lexicons in a probabilistic translation assistant. In *MT Summit VIII*, Santiago de Compostela, Spain, September 2001.

[78] Philippe Langlais, Guy Lapalme, and Marie Loranger. TransType: From an idea to a system. *Machine Translation*, 2002. To Appear.

[79] Philippe Langlais, Guy Lapalme, and Sébastien Sauvé. User interface aspects of a translation typing system. In *AI 2001—Advances in Artificial Intelligence*, pages 245–256, Ottawa, June 2001.

[80] Philippe Langlais, Sébastien Sauvé, George Foster, Elliott Macklovitch, and Guy Lapalme. A comparison of theoretical and user-oriented evaluation procedures of a new type of interactive MT. In *Second International Conference On Language Resources and Evaluation (LREC)*, pages 641–648, Athens, Greece, June 2000.

[81] Raymond Lau, Ronald Rosenfeld, and Salim Roukos. Trigger-based language models: A maximum entropy approach. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 1993*, pages 45–48, Minneapolis, Minnesota, 1993. IEEE.

[82] Ting Liu, Ming Zhou, Jianfeng Gao, Endong Xun, and Changning Huang. Pens: A machine-aided english writing system for chinese users. In ACL-00 [1], pages 529–536.

[83] S. C. Loh and L. Kong. An interactive on-line machine translation system. In B. M. Snell, editor, *Translating and the Computer: Proceedings of a Seminar*, pages 135–148. North-Holland, 1979.

[84] E. Macklovitch, M. Simard, and Ph. Langlais. Transsearch: A free translation memory on the world wide web. In *Second International Conference On Language Resources and Evaluation (LREC)*, volume 3, pages 1201–1208, Athens, Greece, June 2000.

[85] Elliott Macklovitch. Evaluating commercial MT systems. In *Proceedings of ISSCO Evaluators' Forum*, Les Rasses, Switzerland, 1991.

[86] Elliott Macklovitch. The future of MT is now and Bar-Hillel was (almost entirely) right. In *Proceedings of the Fourth Bar-Ilan Symposium on the Foundations of Artificial Intelligence*, Ramat Gan, Israel, 1994.

[87] Elliott Macklovitch. Using bi-textual alignment for translation validation: the TransCheck system. In *Proceedings of the 1st Conference for Machine Translation in the Americas (AMTA)*, pages 157–168, Columbia, Maryland, October 1994.

[88] Elliott Macklovitch. Personal communication, 1998.

[89] Elliott Macklovitch and Graham Russell. What's been forgotten in translation memory. In *Proceedings of the 6th Conference for Machine Translation in the Americas (AMTA)*, Cuernavaca, Mexico, October 2000.

[90] Bente Maegaard. Machine translation. In Eduard Hovy, Nancy Ide, Robert Frederking, Joseph Mariani, and Antonio Zampolli, editors, *Multilingual Information Management: Current Levels and Future Abilities*, chapter 4. U.S. National Science Foundation, April 1999.

[91] Hiroshi Maruyama and Hideo Watanabe. An interactive Japanese parser for machine translation. In COLING-90 [34], pages 257–262.

[92] I. Dan Melamed. Automatic detection of omissions in translations. In COLING-96 [36], pages 764–769.

[93] I. Dan Melamed. *Empirical Methods for Exploiting Parallel Texts.* MIT Press, 2001.

[94] Alan Melby. Some notes on The Proper Place of Men and Machines in Language Translation. *Machine Translation*, 12(1–2):29–34, 1997.

[95] Alan Melby. Machine translation and philosophy of language. *Machine Translation Review*, (9):6–17, April 1999.

[96] T. Mitamura and E. H. Nyberg. Controlled English for knowledge-based MT: experience with the kant system. In *Proceedings of the 6th Conference on Theoretical and Methodological Issues in Machine Translation (TMI)*, pages 158–172, Leuven, Belgium, 1995.

[97] Makoto Nagao. A framework of a mechanical translation between Japanese and English by analogy principle. In A. Elithorn and R. Banerji, editors, *Artificial and human intelligence.* Elsevier, 1984.

[98] S. Niessen, S. Vogel, H. Ney, and C. Tillmann. A DP based search algorithm for statistical machine translation. In COLING-98 [37], pages 960–967.

[99] Sergei Nirenburg. Tools for machine-aided translation: The CMU TWS. *META*, 37(4):709–720, 1992.

[100] E.H. Nyberg and T. Mitamura. The KANT system: Fast, accurate, high-quality translation in practical domains. In COLING-92 [35], pages 1069–1073.

[101] Franz Josef Och, Christoph Tillmann, and Hermann Ney. Improved alignment models for statistical machine translation. In *Proceedings of the 4nd Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 20–28, College Park, Maryland, 1999.

[102] Eugenio Picchi, Carol Peters, and Elisabetta Marinai. A Translator's Workstation. In COLING-92 [35], pages 972–976.

[103] Harry Printz. Fast computation of Maximum Entropy/Minimum Divergence feature gain. In ICSLP-98 [60], pages 2083–2086.

[104] L. R. Rabiner and B. H. Juang. An introduction to Hidden Markov Models. *IEEE ASSP Magazine*, pages 4–16, January 1986.

[105] Philip Resnik. Evaluating multilingual gisting of web pages. In *Proceedings of the AAAI Spring Symposium on Cross-Language Text and Speech Retrieval*, Stanford University, March 1997.

[106] Ronald Rosenfeld. A maximum entropy approach to adaptive statistical language modelling. *Computer Speech and Language*, 10:187–228, 1996.

[107] Michel Simard, George F. Foster, and Pierre Isabelle. Using cognates to align sentences in bilingual corpora. In TMI-92 [109].

[108] C. Tillmann and H. Ney. Word re-ordering and DP-based search in statistical machine translation. In *Proceedings of the International Conference on Computational Linguistics (COLING) 2000*, Saarbrucken, Luxembourg, Nancy, August 2000.

[109] *Proceedings of the 4th Conference on Theoretical and Methodological Issues in Machine Translation (TMI)*, Montréal, Québec, 1992.

[110] Masaru Tomita. Disambiguating grammatically ambiguous sentences by asking. In *Proceedings of the International Conference on Computational Linguistics (COLING) 1984*, pages 476–480, Stanford, California, July 1984.

[111] Masaru Tomita. Feasibility study of personal/interactive machine translation systems. In *Proceedings of the 1st Conference on Theoretical and Methodological Issues in Machine Translation (TMI)*, pages 289–297, Colgate University, Hamilton, New York, 1985.

[112] Star Transit Computer-Aided Translation System, product description. www.star-ag.ch/products/transit.html, 1998.

[113] IBM TranslationManager2, product description. www.software.ibm.com/ad/translat, 1998.

[114] Trados Translator's Workbench 2, product description. www.trados.com/workbench/index.html, 1998.

[115] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(4):260–269, April 1967.

[116] S. Vogel, H. Ney, and C. Tillmann. HMM-based word alignment in statistical translation. In COLING-96 [36], pages 836–841.

[117] Stephan Vogel and Hermann Ney. Translation with cascaded finite state transducers. In ACL-00 [1], pages 23–36.

[118] W. Wahlster, editor. *Verbmobil: Foundations of Speech-to-Speech Translation.* Springer-Verlag, Heidelberg, 2000.

[119] S. Wang, D. Schuurmans, Y. Zhao, and R. Rosenfeld. The latent maximum entropy principle. *IEEE Trans. on Information Theory*, 2002. Submitted.

[120] Ye-yi Wang and Alex Waibel. Fast decoding for statistical machine translation. In ICSLP-98 [60], pages 2775–2778.

[121] Ye-yi Wang and Alex Waibel. Modeling with structures in statistical machine translation. In COLING-98 [37], pages 1357–1363.

[122] Andrew Way and Michael Carl, editors. *Proceedings of the MT Summit VIII Workshop on Example-Based MT*, Santiago de Compostela, Spain, September 2001. International Association for Machine Translation.

[123] Warren Weaver. Translation. In *Machine translation of languages.* MIT Press, Cambridge, MA, 1955.

[124] P. J. Whitelock, M. McGee Wood, B. J. Chandler, N. Holden, and H. J. Horsfall. Strategies for interactive machine translation: the experience and implications of the UMIST Japanese project. In *Proceedings of the International Conference on Computational Linguistics (COLING) 1986*, pages 329–334, Bonn, West Germany, 1986.

[125] Dekai Wu. A polynomial-time algorithm for statistical machine translation. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 152–158, Santa Cruz, California, September 1996.

[126] Kiyoshi Yamabana, Muraki Kazunori, Shin ichiro Kamei, Kenji Satoh, Shinichi Doi, and Shinko Tamura. An interactive translation support facility for non-professional users. In *Proceedings of the 5th Conference on Applied Natural Language Processing (ANLP)*, pages 324–331, Washinton, D.C., 1997.

[127] Rémi Zajac. Interactive translation: A new approach. In *Proceedings of the International Conference on Computational Linguistics (COLING) 1988*, pages 785–790, Budapest, Hungary, August 1988.

[128] Rémi Zajac. Glossary-based MT engines in a multilingual analyst's workstation architecture. *Machine Translation*, 12(1–2):131–151, 1997.

Appendix A

PREDICTION EXAMPLES

In this appendix, I give examples of proposals generated using the best predictor configuration (the MEMD2B model with correction factors and "exact" string prefix probabilities) for the *Realistic* simulation. The top line for each example is the source sentence, and the following lines show the growing target text. Each line corresponds to a new proposal, and the difference between successive lines indicates whether the user accepted the proposal or just typed the next character.

The text of each proposal is shown within slashes, where the 1st slash is at the current cursor position. Proposals are followed by information of the form $(B'/B\ p)$, where B' is the benefit $B(\mathbf{x}, \mathbf{h}, \mathbf{s})$ estimated by the predictor, B is the actual benefit $B(\mathbf{x}, \mathbf{h}, \mathbf{s}, k, a)$ to the user, and p is the corrected probability assigned to the proposal by the model. Empty proposals are shown by two consecutive slashes.

The three examples show predictions for increasing M, from 1 to 3. For single-word completion, most of the time the system makes no proposal, because it knows that short words cannot benefit the user. Both times it does make a proposal the user profits. For two-word and three-word settings, the predictor makes more proposals, reflecting several high probability sequences. These help the simulated user to establish the text in fewer keystrokes, though note that correct proposals are not always accepted as soon as they appear.

Example 1: $M = 1$

```
i wish all 16 participating teams good luck

// (0/0 1)
j// (0/0 1)
je// (0/0 1)
je // (0/0 1)
je s/ouhaite / (0.0835765/2.8043 0.485098)
je souhaite // (0/0 1)
je souhaite b// (0/0 1)
je souhaite bo// (0/0 1)
je souhaite bon// (0/0 1)
je souhaite bonn// (0/0 1)
je souhaite bonne// (0/0 1)
je souhaite bonne // (0/0 1)
je souhaite bonne c// (0/0 1)
je souhaite bonne ch// (0/0 1)
je souhaite bonne cha// (0/0 1)
je souhaite bonne chan// (0/0 1)
je souhaite bonne chanc// (0/0 1)
je souhaite bonne chance// (0/0 1)
je souhaite bonne chance // (0/0 1)
je souhaite bonne chance a// (0/0 1)
je souhaite bonne chance au// (0/0 1)
je souhaite bonne chance aux// (0/0 1)
je souhaite bonne chance aux // (0/0 1)
```

```
je souhaite bonne chance aux 1// (0/0 1)
je souhaite bonne chance aux 16// (0/0 1)
je souhaite bonne chance aux 16 // (0/0 1)
je souhaite bonne chance aux 16 é// (0/0 1)
je souhaite bonne chance aux 16 éq// (0/0 1)
je souhaite bonne chance aux 16 équ// (0/0 1)
je souhaite bonne chance aux 16 équi// (0/0 1)
je souhaite bonne chance aux 16 équip// (0/0 1)
je souhaite bonne chance aux 16 équipe// (0/0 1)
je souhaite bonne chance aux 16 équipes// (0/0 1)
je souhaite bonne chance aux 16 équipes /participantes /
   (2.41019/8.37438 0.823257)
je souhaite bonne chance aux 16 équipes participantes // (0/0 1)
```

Example 2: $M = 2$

```
i wish all 16 participating teams good luck

// (0/0 1)
j// (0/0 1)
je// (0/0 1)
je // (0/0 1)
je s/ouhaite / (0.0835765/2.8043 0.485098)
je souhaite /bonne chance / (0.180152/-0.259992 0.283069)
je souhaite b/onne chance / (0.913672/6.51768 0.863957)
je souhaite bonne chance // (0/0 1)
je souhaite bonne chance a// (0/0 1)
je souhaite bonne chance au// (0/0 1)
je souhaite bonne chance aux// (0/0 1)
je souhaite bonne chance aux /équipes participantes /
   (0.485547/-0.423284 0.115703)
je souhaite bonne chance aux 1// (0/0 1)
je souhaite bonne chance aux 16// (0/0 1)
je souhaite bonne chance aux 16 // (0/0 1)
je souhaite bonne chance aux 16 é/quipes participantes /
   (2.39215/-0.405141 0.39527)
je souhaite bonne chance aux 16 éq/uipes participantes /
   (2.20953/13.9445 0.414935)
```

Example 3: $M = 3$

```
i wish all 16 participating teams good luck

// (0/0 1)
j// (0/0 1)
je// (0/0 1)
je // (0/0 1)
je s/ouhaite bonne chance / (0.0923722/-0.405141 0.137312)
je so/uhaite bonne chance / (0.10894/13.9445 0.164713)
je souhaite bonne chance // (0/0 1)
je souhaite bonne chance a// (0/0 1)
je souhaite bonne chance au// (0/0 1)
je souhaite bonne chance aux/ équipes participantes /
   (0.140124/-0.441428 0.112948)
je souhaite bonne chance aux /équipes participantes /
   (0.485547/-0.423284 0.115703)
je souhaite bonne chance aux 1// (0/0 1)
je souhaite bonne chance aux 16// (0/0 1)
je souhaite bonne chance aux 16 // (0/0 1)
je souhaite bonne chance aux 16 é/quipes participantes /
   (2.39215/-0.405141 0.39527)
je souhaite bonne chance aux 16 éq/uipes participantes /
   (2.20953/13.9445 0.414935)
je souhaite bonne chance aux 16 équipes participantes // (0/0 1)
```

Appendix B

STATISTICAL MACHINE TRANSLATION

Modern statistical machine translation began with the seminal work by the IBM group, described in [23] and several other papers [13, 15, 22, 24, 26]. Although some radically different approaches have been proposed—for example, Wu's stochastic inversion transduction grammars [125], Alshawi's head automata [3], and other work on stochastic finite-state transducers [10, 117]—the field has for the most part concentrated on the series of models defined by the IBM group, or fairly minor variations thereof.[1] The bulk of research in recent years has been devoted to search algorithms (see the list of cites in section 3.1), but some attention has also been paid to the models themselves, with notable enhacements including the introduction of HMM-based alignments [40, 116], and the incorporation of syntactic constraints [101, 121].

In this chapter, I give a very brief overview of the IBM translation models, along with the extension to HMM-based alignments, which seems the most obvious and natural of the proposed enhancements to the original framework. To make the main ideas easier to follow, I omit most of the technical detail from Brown et al's original description.

B.1 The Noisy Channel

The IBM models are based on a noisy-channel (or source-channel) conceptualization, in which an original text in some language is considered to have been corrupted dur-

[1] This excludes related work in example-based MT, which is currently a very active field—see for example the papers in [122]; and also some slightly more distant work on learning non-probabilistic finite-state transducers—see the papers in [59].

ing passage through a noisy channel (or an encoding process), to produce an observed text in another language. The task of translation is therefore seen as *decoding* or reversing the effect of the corruption to recover the original text from the observed one. This conceptualization leads to many insights based on parallels to cryptography and communication theory (for example, that it may be possible to learn model parameters without needing aligned texts [71]), but it has the unfortunate consequence of creating ambiguity about what the term *source text* should refer to. Some authors prefer to follow the natural convention in MT—used throughout this thesis—in which the source text is the given text to be translated; others adopt noisy-channel notation in which the source text is the original text to be recovered (ie, the target text in the natural MT convention). To avoid confusion, in this chapter I will follow Brown et al's lead and describe models for the specific case of French to English translation.

The problem in SMT is to find the most likely English translation $\hat{\mathbf{e}}$ for a given French text[2] $\mathbf{f}$ according to a model for $p(\mathbf{e}|\mathbf{f})$:

$$\begin{aligned} \hat{\mathbf{e}} &= \underset{\mathbf{e}}{\operatorname{argmax}}\, p(\mathbf{e}|\mathbf{f}) \\ &= \underset{\mathbf{e}}{\operatorname{argmax}}\, p(\mathbf{e})p(\mathbf{f}|\mathbf{e}) \end{aligned}$$

where the second line embodies the noisy channel, and is referred to by Brown et al as the "fundamental equation of SMT". This converts the original problem of modeling $p(\mathbf{e}|\mathbf{f})$ into two related problems: building a language model $p(\mathbf{e})$ and a "reversed" translation model $p(\mathbf{f}|\mathbf{e})$. Both are simpler than the original, because $p(\mathbf{e})$ is concerned only with the statistical properties of English, and $p(\mathbf{f}|\mathbf{e})$ only with the translation relation between English and French. The idea is that, during the search for $\hat{\mathbf{e}}$, $p(\mathbf{f}|\mathbf{e})$ can afford to sanction ungrammatical candidates for $\mathbf{e}$ that appear to be strongly related to $\mathbf{f}$, because these will get filtered out by $p(\mathbf{e})$ in any case. See section 3.5.1 for further comments on the effects of reversing the translation model.

[2] As is usual in MT, $\hat{\mathbf{e}}$ and $\mathbf{f}$ are sentences. More precisely, in the IBM Candide SMT system

B.2 Alignments

A key concept in the IBM models for $p(\mathbf{f}|\mathbf{e})$ is that of an *alignment* that describes the translation relation between $\mathbf{e}$ and $\mathbf{f}$. Viewing $\mathbf{e}$ and $\mathbf{f}$ as sets of tokens, $\mathbf{e} = \{e_1, \ldots, e_I\}$ and $\mathbf{f} = \{f_1, \ldots, f_J\}$, an alignment is a subset of the Cartesian product $\mathbf{e} \times \mathbf{f}$. The occurrence of a pair (e_i, f_j) within an alignment means that there is some valid relation between e_i and f_j. Although this is quite a general way of capturing the relation between $\mathbf{e}$ and $\mathbf{f}$, it is not expressive enough to represent all phenomena of interest in translation. For example, it cannot simultaneously capture the fact that *drug manufacturer* is translated as *fabricant de médicaments* in some larger context *and* that *fabricant* is a direct translation of *manufacturer* but *médicaments* is not. This is because the only way to represent non-decomposable translation relations between groups of words is add their whole Cartesian product to the alignment. To address this deficiency, some notion of hierarchy would be required, or perhaps a system of differentiating between different links within an alignment.

Alignments are used within the IBM models as latent variables that motivate independence hypotheses; that is, via the expansion:

$$p(\mathbf{f}|\mathbf{e}) = \sum_{\mathbf{a}} p(\mathbf{f}, \mathbf{a}|\mathbf{e}) \tag{B.1}$$

where $\mathbf{a}$ denotes an alignment in $\mathbf{e} \times \mathbf{f}$. To facilitate independence assumptions, and to permit training with the EM algorithm in the (typical) absence of hand-annotated alignment data, permissible alignments are restricted to those in which each French word connects to either 1 English word or none at all. The latter situation is denoted by a connection to a special null word e_0, which is deemed to be prepended to $\mathbf{e}$. Thus the alignments used within all IBM models are asymmetrical, permitting relations like

[13] they are transformed versions of sentences input to a synthesis procedure and output from an analysis procedure, respectively. Since analysis and synthesis are usually very rudimentary in SMT (eg casemapping, local grammatical regularization, etc), the distinction between the actual sentences and their transformed versions is not important for understanding the model.

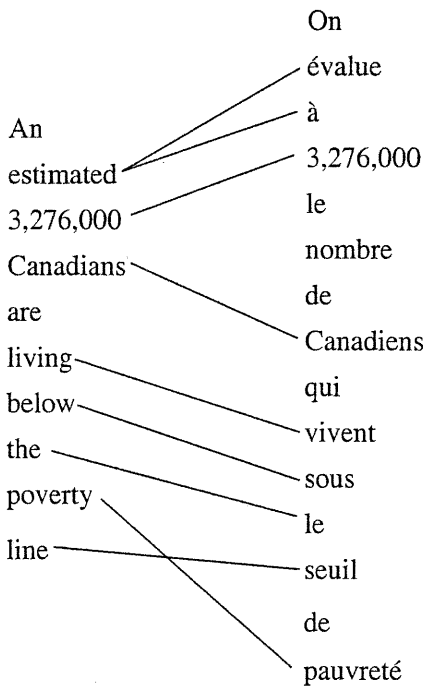

Figure B.1. One possible alignment between the English text on the left and the French text on the right (for French to English translation). French words connected to the null word are shown without lines.

potato/pomme de terre in which a single English word connects to many French words, but not ones like *red tape/paperasserie* in which the converse holds. A consequence of this is that alignments can be represented very simply as vectors which, for each French word, give the index of the English word that it connects to: $\mathbf{a} = a_1, \ldots, a_J$, where each a_j is in $0 \ldots I$. Figure B.1 shows a well-formed English/French alignment.

B.3 Training

The IBM models form a sequence, numbered from 1 to 5. In general, each model adds extra parameters to the previous one in the sequence and/or provides a slightly different interpretation of the original parameters. The point of this is that, for the more powerful models 3 to 5, there is no known algorithm for efficiently computing

the sum over all alignments in equation (B.1). A way around this might be to approximate the sum by the probability $p(\mathbf{f}, \hat{\mathbf{a}}|\mathbf{e})$ of the most likely (Viterbi) alignment $\hat{\mathbf{a}}$, but unfortunately this cannot be found in reasonable time either. By themselves, therefore, these models cannot be used for practical translation. In fact, they cannot even be trained on sentence-aligned text, because there is no way of computing the alignment probabilities $p(\mathbf{a}|\mathbf{e}, \mathbf{f}) = p(\mathbf{f}, \mathbf{a}|\mathbf{e})/p(\mathbf{f}|\mathbf{e})$ required for the EM algorithm.[3]

Brown et al's solution to this is to begin with two simpler models for which $p(\mathbf{f}|\mathbf{e})$ can be computed exactly and efficiently in $O(I, J)$ time. Model 1 has the additional property that parameter estimates calculated with the EM algorithm from sentence-aligned text give a global maximum for the training corpus likelihood, rather than just the usual local maximum. In other words, its final parameter values are independent of whatever initial values have been assigned. Model 2 does not have this property, so its lexical parameters (described below) are initialized with the final values from model 1.

The training algorithm for models 3, 4, and 5 relies on the fact that, although it is not feasible to calculate the sum over all alignments for these models, it is possible to calculate the probability of any given alignment. All of these models therefore approximate $\sum_{\mathbf{a}} p(\mathbf{f}, \mathbf{a}|\mathbf{e})$ by $\sum_{\mathbf{a} \in S} p(\mathbf{f}, \mathbf{a}|\mathbf{e})$, where S is a small set of "good" alignments according to some heuristic. For model 3, S is established by exploring alignments close to model 2's Viterbi alignment, and keeping the most probable subset of these.[4] For model 4, S is established the same way, except that model 3 is used to

[3] Training on word-aligned text *would* be possible, but unlike sentence alignments, making high-quality word alignments requires expensive human annotation.

[4] More precisely, define the neighbourhood of $\mathbf{a}$ to be the set of all alignments reachable by moving a single link, swapping two links, or making no change. Starting with model 2's Viterbi alignment, do a greedy search through successive neighbourhoods—always picking the best new alignment according to model 3—until the best alignment stops changing. Then add all alignments in its neighbourhood to S. Do the same thing starting with all possible *pegged* model 2 Viterbi alignments, where a pegged Viterbi alignment is the most probable alignment constrained to have

do an initial pruning because calculating $p(a|\mathbf{e},\mathbf{f})$ is much more expensive with model 4.[5] Model 5 uses model 4's S, with some of the least probable alignments according to model 4 removed (thus model 5 is not used at all to define its own S). Each of these models begins training with parameter estimates derived from the previous ones. Also, because all preceding models except for model 1 are required to find S, any parameters they do not share with the later models need to be updated as the later ones are trained.

B.4 Models 1, 2, and HMM

Models 1 and 2, as well as the HMM model proposed in [116], are based on the following approximation:

$$\begin{aligned} p(\mathbf{f},\mathbf{a}|\mathbf{e}) &= p(\mathbf{f}|\mathbf{a},\mathbf{e})p(\mathbf{a}|\mathbf{e}) \\ &\approx p(J|\mathbf{e}) \prod_{j=1\ldots J} p(f_j|e_{a_j})p(\mathbf{a}|\mathbf{e}). \end{aligned}$$

Thus, in all models, the probability of each French word depends only on the English word to which it is linked through the current alignment. This is captured by lexical parameters of the form $p(f|e)$, which can be thought of as a large statistical bilingual dictionary with an entry for every possible pair of English/French words. $p(J|\mathbf{e})$ is necessary in order to normalize $p(\mathbf{f}|\mathbf{e})$ over all lengths of $\mathbf{f}$, and is modeled as uniform up to some large value of J.

These models differ only in how they assign probabilities to different alignments. Model 1 considers all alignments equally likely, with probability $p(\mathbf{a}|\mathbf{e}) = 1/(I+1)^J$. Model 2 assumes that connections depend only on the positions of the words being

some specified link.

[5] Each neighbourhood is first ranked according to model 3, and the best alignment is defined as the highest ranked one which, according to model 4, improves on the original alignment.

linked:

$$p(\mathbf{a}|\mathbf{e}) = \prod_{j=1}^{J} p(a_j|j, I, J),$$

where the alignment parameters $p(i|j, I, J)$ give the probability that the jth French word connects to the ith English word. The HMM model assumes that the probability of each connection depends on the previous one:

$$p(\mathbf{a}|\mathbf{e}) = \prod_{j=1}^{J} p(a_j|a_{j-1}, I),$$

In this model, since there is no dependence on the current position j in the French sentence, it is reasonable to make the parameters $p(i|i', I)$ depend only on the distance $|i - i'|$ between the English positions connected to consecutive French words, rather than the actual values of those positions. This can be accomplished with standard parameter-tying techniques [66].

Because connections are independent of each other in models 1 and 2, the sum over all alignments in equation (B.1) can be factored in a particularly efficient way. Letting $x_{a_j} = p(f_j|e_{a_j})p(a_j|j, I, J)$:

$$\begin{aligned}
\sum_{\mathbf{a}} p(\mathbf{f}, \mathbf{a}|\mathbf{e}) &= \sum_{\mathbf{a}} \prod_{j=1}^{J} p(f_j|e_{a_j})p(a_j|j, I, J) = \sum_{\mathbf{a}} \prod_{j=1}^{J} x_{a_j} \\
&= \sum_{a_1=0}^{I} \cdots \sum_{a_J=0}^{I} (x_{a_1} \cdots x_{a_J}) \\
&= (\sum_{a_1=0}^{I} x_{a_1})(\sum_{a_2=0}^{I} x_{a_2}) \cdots (\sum_{a_J=0}^{I} x_{a_J}) \\
&= \prod_{j=1}^{J} \sum_{a_j=0}^{I} x_{a_j} = \prod_{j=1}^{J} \sum_{i=0}^{I} p(f_j|e_i)p(i|j, I, J), \qquad \text{(B.2)}
\end{aligned}$$

where for model 1, $p(i|j, I, J) = 1/(I+1)$. This reduces the time required to calculate the sum to $O(IJ)$ time, compared to $O(J(I+1)^J)$ for the naive algorithm. A similar method can be used to find the Viterbi alignment in $O(IJ)$ time. Although this factorization does not work for the HMM model, there is an obvious isomorphism

between it and the classical HMM model [104], so both EM training (via the forward-backward algorithm) and the Viterbi algorithm can be carried out in $O(I^2J)$ time.

B.5 Models 3, 4, and 5

Models 1 and 2 are based on a generative process in which, for each position in the French sentence, first a connecting position in the English sentence is chosen, then the French word is filled in on the basis of the English word in that position. In the later models this is reversed. For each English word, first the number of French words it produces (its *fertility*) is established, then the identities of those words, and finally their positions. These models differ from models 1 and 2 by explicitly parameterizing fertilities, and also in the way they assign word positions, which is not based explicitly on alignments (though it results in exactly the same kind of restricted alignment as used by the earlier models). One consequence of this is that the factorization in (B.2) no longer applies. Models 3, 4, and 5 share with models 1 and 2 the fact that the identity of each French word depends only on the English word to which it is connected within an alignment. Thus the main bulk of the parameters—the table of lexical probabilities $p(f|e)$—is common to all 5 models. The later models differ from each other only in the way they assign positions to French words. Since the equations describing these models are substantially more complex than for the earlier models, I give only a brief description of their parameters below.

Model 3 includes a set of fertility parameters of the form $p(\phi|e)$ which capture the propensity of different English words to translate to differing numbers of French words. For example, one would expect $p(3|potato)$ to be higher than $p(2|potato)$, reflecting the fact that *potato* usually translates into *pomme de terre.* Note that fertilities have no connection to the actual words selected as translations, so that model 3 would be happier with *potato/pomme pomme pomme* than with *potato/pomme de terre* if $p(pomme|potato)$ were higher than the other lexical probabilities involved

in this expression. In keeping with the generative process described above, model 3 replaces model 2's alignment probabilities with a set of *distortion* probabilities $p(j|i, I, J)$, that describe the distribution of French positions for a given English position. The null word e_0 is treated as a special case for both fertility and distortion parameters, but I will omit the details here.

Model 4 replaces model 3's distortion parameters with parameters intended to model the way the set of French words generated by a single English word tends to behave as a unit for the purpose of assigning positions. There are two families of parameters, one for placing the first word in a unit, and another for placing subsequent words. The first word in the ith unit (the translation of e_i) is assigned to position j in the French sentence with probability $p(j - \overline{U_{i-1}}|e_i, f_j)$, where $\overline{U_{i-1}}$ is the average position of the words in the $i-1$th unit.[6] Note that this assignment depends on the identities of both the English word and the first French word in its translation. To avoid sparseness problems, Brown et al make these parameters depend on 50 equivalence classes defined over the French and English vocabularies, rather than the actual words themselves. The positions j of the subsequent words in a unit each depend on the position of the previous one, with probability $p(j - U_{i,j-1}|f_j)$, where $U_{i,j}$ gives the position of the jth word in the ith unit. As before, the dependence on f_j is expressed through an equivalence class.

Models 3 and 4 are *deficient* in that they assign non-zero probability to French "sentences" in which more than one word may occupy the same position. This is because there is nothing in their distortion distributions that precludes this possibility. The purpose of model 5 is to correct model 4's deficiency—it adds no extra information, just ensures that the final distribution $p(\mathbf{f}|\mathbf{e})$ sums to 1. The details are highly technical, so I omit them here.

---

[6] Where i here is a reindexing of the positions in the English sentence that excludes empty units, ie English words that translate to nothing.

Appendix C

FEATURE SELECTION IN A FRENCH MEMD LANGUAGE MODEL

This appendix describes an exercise to construct a dynamic MEMD language model based on the use of trigger features to boost (or lower) the probabilities of certain words when other triggering words have occurred recently. Although the results of the experiments are somewhat disappointing, the techniques used are similar to those in chapters 3 and 4. Many of these, such as the MEMD framework itself, Berger et al's gains algorithm, Printz' algorithm, and the IIS algorithm are explained in much greater technical detail here than in the previous chapters.

C.1 Introduction

The *trigram* [64] is the most widely-used language model. It is based on the assumption that only the last two words in a history are significant when predicting what will come next: $p(w|\mathbf{h}) \approx p(w|w'', w')$, where $\mathbf{h}$ ends with w'', w'. Advantages of the trigram are that it is conceptually simple, very efficient to train and run, and gives surprisingly good results. Its main disadvantage is that it relies on a very large number of parameters. This complicates implementation and makes maximum likelihood estimates from relative frequencies unreliable (for example, any trigram not observed in the training corpus will be assigned a probability of zero, whereas intuitively even the largest training corpora will not include a large number of rare but linguistically valid trigrams). An effective and popular solution to the latter problem is to smooth raw ML trigram estimates by incorporating information from more reliable bigram and unigram estimates.

Although the smoothed trigram is obviously flawed as a model of natural language, it has proven surprisingly difficult to come up with alternatives that significantly outperform it. One problem is that it is hard to find an effective way of incorporating additional information into a trigram. To illustrate this, consider a *cache* model [65, 73] consisting of a unigram distribution estimated from relative frequencies over the last several hundred words of $\mathbf{h}$. Clearly this captures information about recent lexical preferences that would be useful in enhancing static trigram predictions at the current position. However, the optimum method for combining trigram and cache distributions is not obvious. A standard approach is to take a linear combination of the form $a\ p_{\text{trigram}}(w|\mathbf{h}) + b\ p_{\text{cache}}(w|\mathbf{h})$, where $a + b = 1$, but this method has the drawback that it tends to average over the strengths and weaknesses of both models. In a context where the trigram is able to make a confident prediction but the cache is unsure, for example, the combining weights will dilute the strength of the trigram's prediction. In principle this could be remedied by making the combining weights depend on $\mathbf{h}$, but in practice (as in this example) it is not always obvious how to do this in such a way that the weights themselves can be estimated reliably.

An alternate approach for combining information from different sources has recently become popular for NLP applications. This technique, known as Maximum Entropy/Minimum Divergence (MEMD) modeling, uses a collection of *features*—real-valued functions $f(w, \mathbf{h})$—to capture arbitrary relations between a word and its context. There are no restrictions on what can constitute a feature. For example, a feature could return the frequency of w in the last hundred words of $\mathbf{h}$ (analogous to a cache model); or it might capture the fact that w belongs to a particular grammatical category and that the last few words in $\mathbf{h}$ belong to a particular sequence of grammatical categories; or it might indicate w's semantic class. The main strength of MEMD modeling is that it provides great flexibility for combining disparate sources of information; and furthermore, in contrast to techniques like linear combination, it does so in a theoretically-justified way, as will be described in the next section.

The drawback to MEMD is that it is very computationally expensive. For example, although it would be possible to construct the equivalent of a trigram model within this framework, the implementation would be very cumbersome. Fortunately, there is a way around this: MEMD also provides for a *reference* distribution, which plays a role similar to that of a Bayesian prior. In the case of language modeling, a natural candidate for this distribution is a smoothed trigram. Using a trigram as a reference model gives the MEMD model good baseline performance, and one can hope to find a set of features that will improve substantially on this baseline without sacrificing too much of the trigram's efficiency as a result.

The main challenge in MEMD modeling is finding a maximally informative set of features. Linguistically-motivated heuristics can be used to define feature sets, but a better approach is to use linguistic knowledge to define an initial large pool of candidate features and then select a subset from this pool on the basis of empirical evaluation (the assumption here is that the initial pool is either too large to be practical or so large as to overfit the training corpus). Berger et al [15] have described a powerful and natural algorithm for feature selection, but unfortunately it is too expensive to be used for full-scale language models trained over large corpora. In a recent paper [103], Printz suggests a much more efficient variation on Berger et al's algorithm, which has the potential to make automatic feature selection more viable for language modeling.

In this work I describe a MEMD language model for French which uses a smoothed trigram as a reference distribution and a set of *triggers* as features. Triggers are functions of the form:

$$f_{uv}(w, \mathbf{h}) = \begin{cases} 1, & u \text{ occurs in the last } L \text{ words of } \mathbf{h}, \text{ and } v = w \\ 0, & \text{else} \end{cases}$$

where u and v are words in the vocabulary, and the "window length" parameter L is the same for all triggers. Triggers are a kind of generalization of a unigram cache model (which can be thought of as a collection of "self triggers" f_{uu}), but

with frequency replaced by a binary indicator function for reasons of efficiency and robustness. They were first used by Rosenfeld [81, 106] within a ME framework very similar to the one I describe here, but with the important difference that in Rosenfeld's case trigrams were incorporated as features instead of within a reference distribution. Triggers have also been used by Berger and Printz [14].

The aim of my experiments was twofold: to evaluate Printz' algorithm, both from a practical standpoint and empirically by comparing it to a simple heuristic method for selecting trigger features; and to measure the performance improvement over a trigram obtainable by using trigger features within an MEMD framework, in comparison to the improvement given by a roughly equivalent cache model in a linear combination.

The rest of this appendix is structured as follows: section 2 gives a formal description of the MEMD method, section 3 describes the feature selection algorithms, section 4 describes the training algorithms, section 5 describes the experimental setup and results, and section 6 concludes.

C.2 Maximum Entropy/Minimum Divergence Model

A MEMD model has the form:

$$p(w|\mathbf{h}) = q(w|\mathbf{h}) \exp(\vec{\alpha} \cdot \mathbf{f}(w, \mathbf{h}))/Z(\mathbf{h}), \tag{C.1}$$

where $q(w|\mathbf{h})$ is a reference distribution, $\mathbf{f}(w, \mathbf{h})$ maps $(w, \mathbf{h})$ into an n-dimensional feature vector, $\vec{\alpha}$ is a vector of feature weights (the parameters of the model), and $Z(\mathbf{h})$ is a normalizing factor (the sum over all w of the numerator term). Letting $q(w|\mathbf{h}) = \exp(f_0(w, \mathbf{h}))$, we can write:

$$\log s(w, \mathbf{h}) = f_0(w, \mathbf{h}) + \vec{\alpha} \cdot \mathbf{f}(w, \mathbf{h}) \tag{C.2}$$

where $s(w, \mathbf{h})$ is the numerator in (C.1). It is easy to see that the right hand side of (C.2) represents a single-layer neural net with $n + 1$ inputs, a fixed weight of 1 on

input 0, and a single output which gives a score for $(w, \mathbf{h})$. (C.1) applies a softmax function summed over all words to this output. Alternately, the model can be thought of as computing a function from $\mathbf{h}$ to a vector of scores, one for each word. In this case, the net consists of $|W|$ units, where W is the vocabulary, the input and output of w'th unit are described by (C.2), and the vector of weights $\vec{\alpha}$ is constrained to be identical for all units.

As their name implies, MEMD models can also be given an information-theoretic interpretation. Suppose we are confident that the expected values of the feature functions with respect to the empirical distribution $\tilde{p}$ defined by the data are an accurate reflection of their true values. If this is all that is known about the true distribution, a reasonable estimation strategy is to use the distribution that has maximum entropy among all those which have the desired expected values. The idea is to avoid making any inferences which are not supported by the data. If, in addition to the data, some prior (or reference) distribution q is available, then finding the distribution with maximum entropy can be generalized to finding the distribution p with minimum Kullback-Leibler divergence (relative entropy) $D(p||q)$ from the reference, where $D(p||q) = \sum_x p(x) \log[p(x)/q(x)]$. Given q and a set of expected-value constraints of the form:

$$E_p[f_i(w, \mathbf{h})] = E_{\tilde{p}}[f_i(w, \mathbf{h})], \; i = 1 \ldots n,$$

it can be shown [15] that the distribution which satisfies the constraints with minimum divergence from q is unique and has the form:

$$p(w, h) = q(w, h) \exp(\vec{\alpha} \cdot \mathbf{f}(w, \mathbf{h}))/Z$$

where Z is the sum over all $(w, \mathbf{h})$ of the numerator. (For most NL applications, Z is too expensive to compute, so the conditional form (C.1) is used instead of the joint—fortunately, Z cancels out when deriving the conditional using Bayes law.) Remarkably, it can also be shown that the minimum discrimination information distribution is also the maximum likelihood distribution with respect to $\vec{\alpha}$, and furthermore that

likelihood is a convex function of $\vec{\alpha}$, so hillclimbing techniques are guaranteed (in principle) to find the MDI/ML distribution in this case.

In certain fields such as physics (where the technique originated [62, 63]), this result is of direct practical utility, because the expected values of a relatively small number of parameters can be measured with precision. However, for most NLP applications—including language modeling—it is not, because the reliable statistics on which the model should be based are unknown. Features must therefore be established using heuristics, as described in the next section.

It is interesting to examine intuitively how a model of the form (C.1) differs from linear combination as a method of combining different sources of information. To simplify the comparison, suppose that there are two equally-valuable sources of information to be combined, captured in the case of linear combination by two distributions p_1 and p_2 with a weight of .5, and in the case of MEMD by two feature functions f_1 and f_2, each with a weight of 1. Suppose that, in some context, p_1 and f_1 both make a maximally strong prediction of some word w, while p_2 and f_2 are maximally uncertain. Then the resulting linear combination will assign a probability of .75 to w, but the MEMD model will assign a probability approaching 1. Thus MEMD exhibits a built-in bias in favour of strong predictions, (and also strong anti-predictions) which may lead it to outperform a linear combination whenever such preferences are based on reliable empirical evidence.

C.3 Feature Selection

The intuition which underlies Berger et al's feature-selection algorithm is simply that, because training a MEMD model involves maximizing the likelihood, the optimum set of features of a given size should be the one which assigns highest likelihood to the corpus. Since determining the likelihood associated with all feature sets is combinatorially impossible, a sensible strategy is to grow a set one feature at a time

by greedily adding at each step the new feature which gives the greatest increase in likelihood over the current model. However, even this is very expensive, because it necessitates training a new model for each candidate feature that is to be evaluated. Berger et al therefore advocate holding all parameters of the current model fixed and optimizing only the weight associated with the current candidate feature. Formally, define the *gain*, G_f, due to a feature f to be its associated maximum log-likelihood ratio:

$$G_f = \max_{\alpha} G_f(\alpha) = \max_{\alpha} \frac{1}{T} log \frac{p_{f\alpha}(\mathbf{w})}{p(\mathbf{w})}$$

where $\mathbf{w} = w_1, \ldots, w_T$ is the training corpus, p is the model at the current step, and $p_{f\alpha}$ is p with the addition of the feature f with a weight of α. The feature which gets added to the model at each step will be the one with the highest gain.

The essential step in feature selection is therefore that of maximizing $G_f(\alpha)$ in α. Berger et al observe that this can be done in parallel for all candidate features at once, and suggest using Newton's method to find the root of the derivative $G'_f(\alpha)$. Each iteration of Newton's method requires one pass over the corpus to calculate $G'_f(\alpha)$ and $G''_f(\alpha)$ for the current value of α.[1] Although this is not substantially more expensive than training a model (see the next section), it must be kept in mind that the purpose of feature selection is to winnow a very large initial pool of features down to a manageable size. If the initial pool is too large to permit training in a reasonable time, it will also be too large to permit gains to be computed in a reasonable time.[2] Another difficulty with this method of feature selection stems from its incremental

[1] Alternately, certain values associated with each feature can be calculated in a single pass over the corpus and stored in memory, allowing the algorithm to proceed without any future references to the corpus. For large feature sets and corpora the memory requirements of this technique are prohibitive.

[2] Note, though, that there is one important difference between training and computing gains: features interact during training but not when computing gains. This means that gains could still be computed, sequentially, for a set of features that was too big to fit in memory at once.

nature: searching a large pool of features for the best subset of size 100k, for example, would require 100k feature-addition steps and therefore at least this many passes over the corpus.

Printz' method addresses both these problems. First, he suggests that Newton's algorithm can be replaced by an approximated function method which requires only a single pass over the corpus. Second, he advocates doing away with repeated feature addition steps by simply using the gain over the reference model as a measure of a feature's worth in the final model. His method can therefore be summarized as follows: compute approximate gains with respect to the reference model in a single pass over the corpus, rank features in order of decreasing approximate gain, then select the optimum cutoff point in this ranking. In theory, this seems a radically suboptimal search, but it is justified if it allows for the fruitful exploration of very large feature sets which would otherwise remain inaccessible.[3]

The core of Printz' algorithm is the approximated function method, which in essence works as follows. For each feature f (in parallel, if desired), compute the value of $G'_f(\alpha)$ over a pre-determined set of test points $\{\alpha_1, \ldots, \alpha_N\}$. Fit these points using least-squares polynomial interpolation, then use the interpolating function to solve numerically for the root, which approximates $\hat{\alpha}$, the optimum weight for f. Finally, integrate the interpolating function numerically (over a range which depends on $\hat{\alpha}$) to get an approximation for G_f.[4] Of all these steps, only the computation of $G'_f(\alpha)$ depends on the corpus, so the algorithm requires only a single pass (assuming

[3] My own experience underscores this last point: machine-time limitations forced me to evaluate Printz' algorithm by comparing it only to a simple heuristic, rather than to Berger et al's algorithm as well, which would have been more interesting.

[4] I have glossed over many details here, in the process obscuring the reason for using numerical rather than analytic integration when the interpolating function is apparently a polynomial. In actual fact, for technical reasons, the interpolating function is the reciprocal of a polynomial, and the least-squares fit is to the reciprocals of the test points.

a memory overhead of S times the number of features, for storing intermediate values of $G'_f(\alpha_1), \ldots, G'_f(\alpha_N)$).

The approximated function method has a fairly large number of parameters which must be tuned, including stopping criteria for the three numerical algorithms used (least-squares polynomial, Newton-Raphson root-finding, and Romberg integration), order of the interpolating polynomial, and number and location of the test points. Of these, I found that performance was particularly sensitive to the polynomial order and the location of test points, with severe degradation in cases where the test points failed to bracket $\hat{\alpha}$, or where $\hat{\alpha}$ was a large negative value and the polynomial order was low. Furthermore, I found that the accuracy of the numerical approximation to G_f was very sensitive to small errors in the estimate of $\hat{\alpha}$, suggesting that, where time permits, it might be better to use an additional pass over the corpus to calculate the true values of $G_f(\tilde{\alpha})$, (where $\tilde{\alpha}$ is the approximation to $\hat{\alpha}$) instead of relying on numerical integration. Table C.3 shows the performance of Printz' algorithm on a small set of triggers, and illustrates that—for this test at least—the resulting feature ordering does not correspond to the true ordering. However, because the results seem fairly reasonable, to save time I adopted Printz' parameter values verbatim, and used a 4th order polynomial with the (logscale) test points $\{-1, 0, 2, 6, 10, 14\}$ for all experiments.

C.4 Training

A simple way to train MEMD models is by gradient ascent. The log-likelihood of the corpus is given by:

$$\log p(\mathbf{w}) = \sum_{t=1}^{T} \log q(w_t|\mathbf{h}_t) + \vec{\alpha} \cdot \mathbf{f}(w, \mathbf{h})) - \log Z(\mathbf{h}_t).$$

$G_f(\hat{\alpha})$	$G_f(\tilde{\alpha})$	$\tilde{G}_f(\tilde{\alpha})$	$\hat{\alpha}$	$\tilde{\alpha}$	trigger
7.43e-05	7.20e-05	6.78e-05	0.810	1.012	m. → m.
2.50e-05	1.98e-05	7.38e-05	-0.389	-2.510	gouvernement → président
1.84e-05	1.26e-05	4.88e-05	-0.385	-1.490	je → loi
8.17e-06	6.41e-06	1.61e-06	0.281	0.186	président → est
7.65e-06	7.21e-06	1.17e-05	-0.258	-0.645	ministre → canada
3.82e-06	3.82e-06	3.89e-06	-0.103	-0.104	est → je
2.69e-06	2.42e-06	4.52e-06	-0.133	-0.182	a → gouvernement
2.69e-06	2.01e-06	5.84e-06	-0.161	-0.364	gouvernement → canada
9.87e-07	3.44e-07	3.05e-06	-0.098	-0.238	nous → ministre
6.88e-07	6.50e-07	1.04e-06	-0.070	-0.234	loi → canada

Table C.1. Performance of Printz' algorithm on a set of trigger features selected randomly from among frequent words, over a 390k word training corpus. Features are ordered by decreasing true gain $G_f(\hat{\alpha})$ calculated by training, for each feature, a single-feature model with a reference trigram distribution, to get the optimum weight $\hat{\alpha}$. $\tilde{\alpha}$ denotes the approximation to $\hat{\alpha}$, and $\tilde{G}_f$ is the approximated gain.

Differentiating with respect to the ith weight:

$$\begin{aligned}\frac{\partial \log p(\mathbf{w})}{\partial \alpha_i} &= \sum_{t=1}^{T} f_i(w_t, \mathbf{h}_t) - \frac{\partial \log Z(\mathbf{h}_t)}{\partial \alpha_i} \\ &= \sum_{t=1}^{T} f_i(w_t, \mathbf{h}_t) - \sum_{w} \frac{q(w|\mathbf{h}_t) f_i(w, \mathbf{h}_t) \exp(\vec{\alpha} \cdot \mathbf{f}(w, \mathbf{h}_t))}{Z(\mathbf{h}_t)}.\end{aligned}$$

This expression can be used to update each parameter in the usual way, either in "batch" mode:

$$\alpha_i \leftarrow \alpha_i + \eta \frac{\partial \log p(\mathbf{w})}{\partial \alpha_i},$$

or by using the individual terms in the sum to update parameters at each position in the corpus.

Although gradient ascent is simple to implement, its convergence properties are heavily dependent on the gradient step η, and it can be difficult to specify ahead of time an optimum value (or progression of values) for this variable. Berger et al [15] describe a specialized training algorithm for MEMD called *Improved Iterative Scaling* (IIS), based on earlier work by Darroch and Ratcliff [42], which converges much faster than gradient ascent and does not require setting a step parameter.

Each iteration of IIS involves solving numerically (via Newton's algorithm) for a weight update quantity Δ_i in the following equation:

$$E_p[f_i(w, \mathbf{h}) \exp(\Delta_i \mathrm{f}^{\#}(w, \mathbf{h}))] = E_{\tilde{p}}[f_i(w, \mathbf{h})],$$

where $\mathrm{f}^{\#}(w, \mathbf{h})$ gives the number of features which are active (take on non-zero values) for $(w, \mathbf{h})$, and the expected values are with respect to $p(w, \mathbf{h})$ and $\tilde{p}(w, \mathbf{h})$ respectively. Once Δ_i has been calculated for each weight α_i, all weights are updated according to: $\alpha_i \leftarrow \alpha_i + \Delta_i$, and the process continues until convergence.

A major problem with this equation is that, for language models, the expectation with respect to the joint model $p(w, \mathbf{h})$ is impossible to compute due to the sum over $\mathbf{h}$. To get around this, a common solution is to use the distribution $\tilde{p}(\mathbf{h})p(w|\mathbf{h})$ instead of $p(w, \mathbf{h})$; in other words, to constrain the marginal of the joint model to its empirical

value during training. Under this assumption, and the fact that $\tilde{p}(\mathbf{h}_t) = 1/T$ at each position t in the corpus, the above equation becomes:

$$\sum_{t=1}^{T} \sum_{w} p(w|\mathbf{h}_t) f_i(w, \mathbf{h}_t) \exp(\Delta_i \mathrm{f}^{\#}(w, \mathbf{h}_t)) = T E_{\tilde{p}}[f_i(w, \mathbf{h})]$$

The key to implementing IIS efficiently is to observe that this can be rewritten as a polynomial by making the transformation $\gamma_i = \exp(\Delta_i)$:

$$\sum_{t=1}^{T} \sum_{w} p(w|\mathbf{h}_t) f_i(w, \mathbf{h}_t) \gamma_i^{\mathrm{f}^{\#}(w, \mathbf{h}_t)} = T E_{\tilde{p}}[f_i(w, \mathbf{h})].$$

Thus if the set of (integer) values that $\mathrm{f}^{\#}()$ takes on whenever feature i is active throughout the corpus is known,[5] the main part of the algorithm reduces to collecting the values of the coefficients corresponding to these exponents. At the end of each iteration, when the coefficients are known for all features, Newton's algorithm can be applied to solve efficiently for the value of γ_i that satisfies the polynomial equation for each feature.

The costly part of IIS is computing $p(w|\mathbf{h})$ for each word in the vocabulary and each history in the corpus. Rather than describing in detail how I dealt with this, I will sketch an efficient method for calculating $Z(\mathbf{h})$, which indicates the essence of the approach; similar techniques were applied to gradient ascent and Printz' feature selection algorithm. The main idea is to organize the lookup for the features which are active (ie, non zero) on $\mathbf{h}$ to supply a set of words to which at least one of these features applies. Calling this set A, we can write:

$$\begin{aligned} \sum_{w} q(w|h) \exp(\vec{\alpha} \cdot \mathbf{f}(w, \mathbf{h})) &= \sum_{w \in A} q(w|h) \exp(\vec{\alpha} \cdot \mathbf{f}(w, \mathbf{h})) + \sum_{w \notin A} q(w|h) \\ &= \sum_{w \in A} q(w|h)[\exp(\vec{\alpha} \cdot \mathbf{f}(w, \mathbf{h})) - 1] + 1, \end{aligned}$$

[5] This can be established in a single preliminary pass through the corpus, since these values don't change during training.

eliminating the sum over all "non-active" words in the vocabulary, which can increase efficiency considerably when only a relatively small set of words are active for each history, and when the reference model takes non-negligible time to compute.

In practical tests, I found that IIS gave much better results than gradient ascent, usually converging within 20 iterations or so compared with 50 or more—for models that can take days to train, this is a significant savings. I used IIS for all MEMD training described in the next section.

C.5 Experiments

C.5.1 Test Setup

I ran all experiments using the French text of the Canadian Hansard corpus (transcripts of parliamentary proceedings), for the years 86-94. After certain filtering operations,[6] the corpus consisted of about 34M words in 1008 files, with each file usually corresponding to a single day's parliamentary record. An important characteristic of the Hansard is that it is chronologically ordered, so adjacent files tend to be similar, and later files contain more information about earlier ones than the converse. Since most language model applications will involve new (ie, future) Hansard text, this means that using a random selection of files as a test corpus will give optimistic results when the rest of the corpus has been used for training. A more realistic evaluation can be obtained by training on some initial portion of the Hansard and testing on a later portion. Because of this, I split the corpus into contiguous train and test portions with the training set further subdivided into a main block (A) and two "held-out" blocks (B and C), as shown in table C.5.1. Due to time constraints I

[6] Eliminating sentences longer than 40 words, and those which do not translate into a single English sentence. This was done for other work, and has no relevance to language modeling, but available disk space did not permit me to maintain a separate version of the corpus with these constraints removed.

name	purpose	files	words
train A	main training	918	31,709,800
train B	trigram interpolation parameters	30	1,082,350
train C	feature cutoff	30	1,241,581
test	test corpus	30	1,103,320

Table C.2. Corpus division; note that the four segments shown are contiguous and in chronological order.

used only a single train/test split.

All training and testing was performed on a version of the corpus in which both tokens (ie, word occurences) and sentences had been automatically identified. To control for out-of-vocabulary words, all models used the same vocabulary, consisting of all words which appeared more than once in blocks A and B, plus one special unknown word *UNK*. During training, any word with frequency 1 in the corpus was mapped to UNK; during testing, any word not found in the vocabulary was mapped to UNK, and the probability of each UNK token was divided by the total number of out-of-vocabulary words encountered in the text.

The models' performance was evaluated using the standard *perplexity* measure [8]. This is a geometric average over token probabilities, $p(\mathbf{w})^{-1/T}$, where $p(\mathbf{w}) = \prod_{t=1}^{T} p(w_t|w_1 \ldots, w_{t-1})$ is the probability assigned to the corpus by the model. Perplexity is a useful measure because it takes on convenient values (on the order of 100), is independent of corpus size, and has an intuitive interpretation as the size of a uniform distribution (assumed to contain the correct word) which would give the same performance as the model being evaluated. One thing to note about this measure is that its value is infinite whenever a model assigns zero probability to a word in the corpus.

C.5.2 Reference Model

The reference model for the MEMD models used in all experiments was a standard interpolated trigram, of the form:

$$\begin{aligned} p(w|\mathbf{h}) &= \phi_3(w'', w')\tilde{p}_3(w|w'', w') + \phi_2(w'', w')\tilde{p}_2(w|w') + \\ &= \phi_1(w'', w')\tilde{p}_1(w) + \phi_0(w'', w')p_0(w), \end{aligned}$$

where $\tilde{p}_i()$ is the empirical distribution over i-grams, $p_0(w)$ is a uniform distribution over all words in the vocabulary, and ϕ_i is the weight associated with the ith distribution when w'', w' are the last two words in $\mathbf{h}$. Following standard practice, I let the weights depend on the frequency of the conditioning bigram, ie $\phi(w'', w') = \phi(\text{freq}(w'', w'))$, so there is one set of weights for each distinct bigram frequency in the training corpus.[7]

The first step in creating this model was to count ngrams over block A. More specifically, for $i = 1 \ldots 3$, i-gram frequencies were collected over sentences by prepending $i - 1$ special markers to the beginning of each sentence, then sliding a window of length i along the sentence, counting one i-gram at each position, until the end of the window reached the end of the sentence. This resulted in 7,267,001 trigrams, 1,786,611 bigrams, and 66,509 unigrams; the total vocabulary contains 66,718 words. For each i, the associated empirical distribution is defined as:

$$\tilde{p}_i(w_i|w_1, \ldots, w_{i-1}) = \frac{\text{freq}(w_1, \ldots, w_i)}{\sum_{w_i} \text{freq}(w_1, \ldots, w_i)}.$$

The next step was to estimate maximum likelihood values for the combining weights $\phi_i(w'', w')$. The corpus used for this must be distinct from the one used to collect ngram frequencies, otherwise there will be a bias toward higher-order ngrams; it is easy to see that in the case when both corpora are identical, maximum likelihood

[7] This is not the optimum way of constructing an ngram model [29], but it gives good results and is easy to implement.

will assign a weight of 1 to the empirical trigram distribution. Because there are far fewer combining weights than ngram parameters, (26,570 versus over 9M), I used the much smaller block B to estimate them. Maximum likelihood values were obtained from the EM algorithm, which in this case takes a particularly simple form. Each iteration collects expected values associated with the current model in each bigram context:

$$c_i(w'', w') = \sum_{t:\text{suff}(\mathbf{h}_t)=w'',w'}^{T} \frac{\tilde{p}_i(w_t|\mathbf{h}_t)\phi_i(w'', w')}{p(w_t|\mathbf{h}_t)} \quad i = 0 \ldots 3, \ \forall(w'', w') \tag{C.3}$$

where $\mathbf{h}_t = w_1, \ldots, w_{t-1}$, and the sum is over all such histories that end in w'', w'. At the end of each iteration, each parameter is updated according to:

$$\phi_i(w'', w') \leftarrow \frac{c_i(w'', w')}{\sum_{i=0}^{3} c_i(w'', w')}, \quad i = 0 \ldots 3, \ \forall(w'', w'). \tag{C.4}$$

Table C.5.2 shows some of the perplexities associated with the trigram model and its components. These are quite low, indicating that the Hansard is a fairly homogeneous corpus.

C.5.3 Evaluating Printz' Method

I evaluated Printz's feature selection method by comparing it to an obvious heuristic for selecting trigger features: the mutual information (MI) between trigger and target words, defined as:

$$I(u; v) = \sum_{u,v} \tilde{p}(u, v) \log \frac{\tilde{p}(u, v)}{\tilde{p}(u)\tilde{p}(v)}$$

where $\tilde{p}(u, v)$ is the empirical joint distribution over trigger pairs $u \rightarrow v$, $\tilde{p}(u)$ and $\tilde{p}(v)$ are the left and right marginals of this, and the sum is over the four terms (u, v), $(\bar{u}, v)$, $(u, \bar{v})$, and $(\bar{u}, \bar{v})$.

To establish $\tilde{p}(u, v)$, I counted every occurrence of a word preceding another word in the A and B corpora. In order to reduce the size of the table that needed to be stored, pairs were limited to those where the trigger and target both occurred within

model	perplexities			
	train A	train B	train C	test
$\tilde{p}_3$	18.51			
$\tilde{p}_2$	66.24			
$\tilde{p}_1$	703.61			
p_0	67549.6			
p	23.57	47.17	55.98	61.05

Table C.3. Perplexities of the trigram reference distribution p and its components on different segments of the corpus. The rise in perplexity with "distance" from block A reflects the chronological nature of the Hansard. The perplexities of B, C, and test for the empirical models are not shown because they are infinite.

the same sentence, and where neither word was among the 45 most frequent function words in the vocabulary. To make the triggers complementary to the information captured by the trigram model, pairs where the target occurred within 2 words of the trigger were also eliminated. Of the resulting approximately 20M pairs, I retained the 2M with the highest MI scores which also had pair frequencies greater than or equal to five.

My original plan was to use these pairs as an initial pool for feature selection using Printz' algorithm. However, this would not have been feasible in the available time, so I ran the algorithm on a much smaller pool of just the top 100k pairs (which still required several days to process). (Unfortunely, 100k features is suboptimal—the perplexity of a held-out training corpus versus number of features continues to drop past this point, so the comparison is less interesting than it might have been.) In order that the information available to the algorithm would be similar to that on which MI scores were based, I used a MEMD model with a fixed trigger-window length of 15.

MI ranking	gain ranking
m. → président	<< → >>
<< → >>	ne → pas
m. → monsieur	ils → ils
président → paproski	suppléant → monsieur
hon. → monsieur	seulement → mais
hon. → président	elle → elle
hon. → ministre	m. → m.
voix → !	monsieur → suppléant
m. → –	non → mais
– → président	n' → pas

Table C.4. Top ten trigger pairs for MI and gain ranking methods. (The angle brackets are French quotation marks, rendered incorrectly by LaTeX.)

After computing approximate gains with Printz' algorithm, I sorted the trigger list in order of decreasing gain and compared it to the same list sorted in order of decreasing MI score. The results are significantly different, as shown in table C.5.3. Empirical comparisons were made by training MEMD models (again with window length 15) using the top n features in both lists, for each n in $\{1000, 2000, 5000, 10000, 20000, 50000, 100000\}$. Due to time constraints, the training corpus was limited to the last 100 files in A and B (3.6M tokens in total). The performances of these models over the test corpus are shown in figure C.1 and table C.5. In all cases (except on the 100k feature set, where the two models are identical), the gain-based model outperforms the MI-based model by a small margin.

To test the significance of this result, I computed individual perplexities for each of the 30 files in the test corpus for both MI and gain models at each feature-set size. Since these perplexities show significant variation from file to file, with only a barely discernable upward trend with time, it seems reasonable to treat them as iid. Although the standard deviation for any particular model is high, the deviation of the difference between MI and gain perplexities at a given feature-set size is low; most of the global deviation appears to be caused by the reference trigram, which is by far the strongest determinant of a file's perplexity. Computing the standard normal cutoff points $\sqrt{30}\bar{\Delta}/S$ for perplexity differences Δ using the figures in the last column of table C.5 gives values in the approximate range 10–30, so the null hypothesis that the gain-based models are no better than MI-based models has vanishingly small probability.

C.5.4 Evaluation of MEMD Models

The second set of tests I performed was simply aimed at measuring the improvement of the trigger-based MEMD models over the reference trigram, and comparing this improvement to that given by a cache model incorporated within a linear combination. Although, as mentioned earlier, the MEMD and cache models have certain

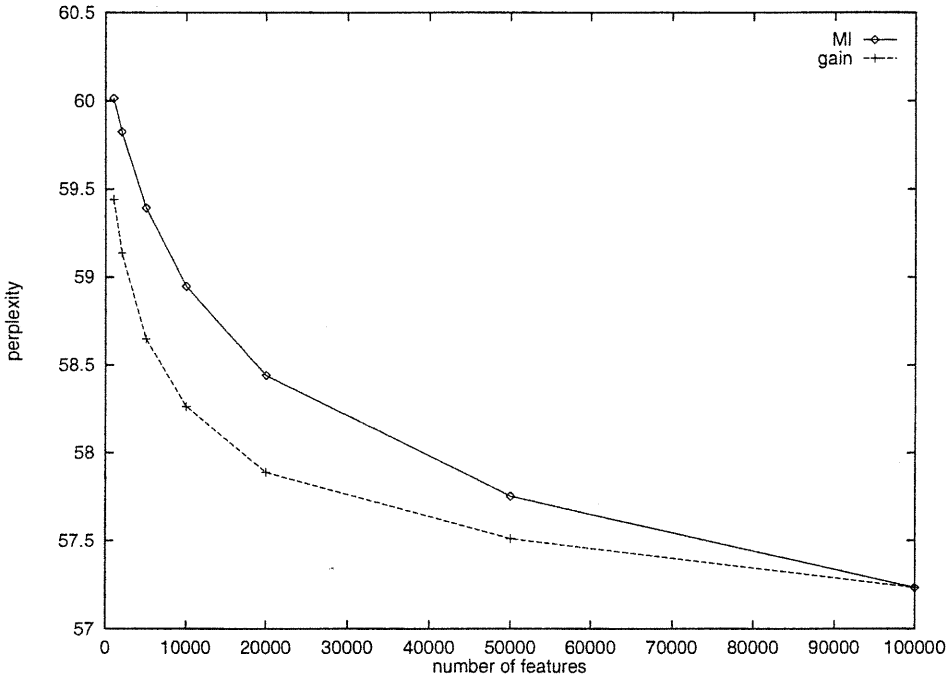

Figure C.1. Performance of MI and gain feature selection methods. Each point represents the performance over the test corpus of a MEMD model using a feature set of the given size.

number of features	MI		gain		Δ	
	avg	sdev	avg	sdev	avg	sdev
1000	60.02	7.38	59.44	7.35	0.57	0.112
2000	59.82	7.38	59.14	7.34	0.69	0.106
5000	59.39	7.36	58.65	7.35	0.74	0.107
10000	58.95	7.34	58.26	7.35	0.68	0.127
20000	58.44	7.32	57.89	7.35	0.55	0.129
50000	57.75	7.29	57.51	7.34	0.24	0.120
100000	57.23	7.20	57.23	7.20	—	—

Table C.5. Average and standard deviation of perplexity over files in the test corpus, for top MI- and gain-ranked feature sets of the given sizes. The column marked Δ reflects the differences in perplexities between each MI model and the corresponding gain model.

similarities, they are not similar enough to permit well-founded claims that, for instance, MEMD combinations are superior to linear combination or the contrary. The interest in comparing these two models is therefore mainly practical, to see whether MEMD models give more improvement over the baseline trigram than cache models, which are far simpler to implement and far more efficient.

As described above, a basic cache model is just a fixed-length buffer of the last L words in $\mathbf{h}$, over which an empirical unigram distribution is calculated. For each value of L in $\{15, 50, 100, 200, 400\}$, I combined the resulting cache model with the reference trigram using a single pair of context-independent weights, estimated over corpus B with the EM algorithm as a special case of equations (C.3) and (C.4).

The results are shown in table C.6 (the MEMD results in this table are reproduced from table C.5). Although no model gives a tremendous improvement over the trigram, the MEMD models clearly outperform the cache models, with even the

model	perplexity		Δ	
	avg	sdev	avg	sdev
trigram	60.48	7.39	—	—
cache 15	60.57	7.44	-0.09	0.123
cache 50	60.17	7.31	0.31	0.146
cache 100	60.07	7.26	0.41	0.183
cache 200	60.05	7.25	0.43	0.208
cache 400	60.08	7.25	0.40	0.215
MEMD 10^3	59.44	7.38	1.04	0.153
MEMD 10^4	58.26	7.34	2.27	0.201
MEMD 10^5	57.23	7.20	3.25	0.312

Table C.6. Average and standard deviation of perplexity over files in the test corpus, for each model listed. The numbers beside the cache models indicate the window length L, and the numbers beside the MEMD models indicate the number of features. The column marked Δ reflects the differences in perplexities between each model and the reference trigram.

smallest 1000-feature model doing significantly better than the optimum 200-word cache model. Interestingly, the cache model which is based on the last 15 words actually performs worse than the reference trigram over the test corpus—this indicates that the MEMD models are more efficient than the cache in using the relatively limited amount of context available to them. The best model tested was the 100k-feature MEMD model, which achieved over 5% lower perplexity than the reference, albeit at significantly increased computational cost. By the same analysis as used in the previous section, all these results are statistically significant.

C.6 Conclusion

I have described some theoretical and practical aspects of the MEMD framework for statistical modeling, as applied to the problem of natural language modeling. I implemented a class of MEMD language models which use binary trigger (word pair) features to improve on the performance of a reference trigram model, and tested them over a large French corpus drawn from the Canadian Hansard. I found that a recent algorithm for automatic feature selection due to Printz yields trigger models which are significantly better than those containing the same number of features selected on the basis of mutual information scores. MEMD models also outperformed an optimized linear combination of a trigram and a cache model, even when the number of triggers used was as small as 1000, and even though the context available to the trigger model was limited to the previous 15 words. Despite these positive results, however, a major obstacle to the use of MEMD techniques for language modeling remains the fact that, for any significant number of features, these models are very computationally expensive to train and run.