English text and Markov chains

A THESIS

Submitted in partial fulfillment of the requirement of the award of the degree of

BACHELOR OF SCIENCE

in

ELECTRICAL ENGINEERING

and

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

By

LEKHANA VUSSE

EE15B119

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS, Chennai

May 2020

CERTIFICATE

This is to certify that the project titled "**English text and Markov Chains**" being submitted to the Indian Institute of Technology Madras by **Lekhana Vusse (EE15B119)**, in partial fulfilment of the requirements for the award of the degrees of **Bachelor of Technology in Electrical Engineering and Master of Technology in Electrical Engineering** is a bona fide record of work carried out by him/her under my supervision. The contents of this project report, in full or in parts, have not been submitted to any other institute or university for the award of any degree or diploma.

Dr. Bharath Bhikkaji
Project Guide
Professor
Dept. of Electrical Engineering
Indian Institute of Technology,
Madras

Dr. David Kolli Pillai
Head of the Department
Professor
Department of Electrical Engineering
Indian Institute of Technology,
Madras

Place: Chennai
Date: 1st May 2020

Acknowledgement

I take immense pleasure to place on this report to my heartfelt gratitude and thankfulness to the following people for helping me to complete the project successfully.

I would like to express my sincere gratitude and thankfulness to my project guide, **Prof. Dr. Bharath Bhikkaji**, Department of Electrical Engineering, IIT Madras for providing me an opportunity, invaluable guidance and supervision throughout the course of the project.

I am highly indebted to him for devoting his valuable time. I sincerely thank him for the help and motivation he provided in order execute this work in good time. His moral support, unreserved cooperation and generosity, which enabled me to complete the work successfully, will be everlasting in my memory.

Finally, I would like to thank all my friends who made my institute life memorable. Also, words cannot express how grateful I am to my beloved parents for their unconditional support.

- **Lekhana Vusse**

Abstract:

Machine translation (MT) system translates one natural language into another language with the help of computers. MT is a key application in the field of natural language processing. **Statistical machine translation** (**SMT**) is a machine translation paradigm where translations are generated on the basis of statistical models whose parameters are derived from the analysis of bilingual text corpora. The statistical approach contrasts with the rule-based approaches to machine translation as well as with example-based machine translation.

The first ideas of statistical machine translation were introduced by Warren Weaver in 1949, including the ideas of applying Claude Shannon's Information theory. Statistical machine translation was re-introduced in the late 1980s and early 1990s by researchers at IBM's Thomas J. Watson Research Centre and has contributed to the significant resurgence in interest in machine translation in recent years. Before the introduction of Neural Machine translation, it was by far the most widely studied machine translation method.

In proposed system, Statistical Machine Translation (SMT) approach has been used for developing English to French Machine Translation System. In SMT, every sentence in the target language i.e. English is a translation of the source language i.e. French with some probability and the best translation will be of high probability which the system will attain in the form of sentence. The key activities involved during translation process are evaluating n-gram language models, finding alignment probabilities using Hidden Markov Model and parameter estimation for language model and translation model. N-gram Language model plays an important role in statistical machine translation. The idea of the model is to make the alignment probabilities dependent on the absolute positions. To achieve this goal, the approach uses IBM models for the word alignment problem.

List of Figures

Table of Contents

Chapter 1

INTRODUCTION

1.1. MOTIVATION

The field of machine translation is almost as old as the modern digital computer. Large, machine-readable corpora are readily available now a days. Statistical methods have proven their value in automatic speech recognition and have recently been applied to lexicography and to natural language processing. We feel that it is time to give them a chance in machine translation. The job of a translator is to render in one language the meaning expressed by a passage of text in another language. This task is not always straightforward. We take the view that every sentence in one language is a possible translation of any sentence in the other.

In this paper, we consider only the translation of individual sentences. Although there are many expert systems which incorporate deep linguistic knowledge-based learning strategies, these are expensive to maintain cost wise as well as difficult to implement on new language pairs. Also, they use one target sentence for each source sentence whereas Statistical Machine Translation gives various different target sentences with a probability for each one. Against Statistical Machine Translations are not confined to specific pair of languages and they can be trained within days to develop parallel corpus to give the translation.

1.2. PROBLEM STATEMENT

We want to describe how these very low-level ideas of SMTs can be used with surprising effectiveness even in situations where very high-level subtle patterns are present. Machine translation started very naively with the idea of putting a two-language dictionary on-line and simply substituting words. This works very badly: one gets virtually unrecognizable phrases (e.g., classic example is the English idiom “out of sight, out of mind,” translated to Chinese and back again and emerging as “the blind idiot”).

The IBM group proposed putting together
(1) a two-language dictionary with probabilities and
(2) a simple model of probable short word sequences in the target language, and combining them by using Bayes’ theorem.

To be specific, let f be a French word string, and e be an English word string. Assume we observe f. A French-to-English dictionary gives a probability distribution $p(e/f)$, whose large values are tentative translations e. Instead, we can use an English-to-French dictionary and a prior on $\tilde{e}$ (e.g., from an n-gram model), and then use Bayes’ theorem:

$$p(e,f) = p(e/f)\,p(f) = p(f/e)\,p(e)$$

We fix f and then seek $\tilde{e}$ = argmax [$P(f/e)\,p(e)$]. The probability $P(f/e)$ is given by an English-to-French dictionary, and the probability $p(e)$ is given by an n-gram prior (e.g., word triples). The resulting e is called the maximum-a-posteriori (MAP) estimate of $\tilde{e}$.

1.2.1. Objective

we look for the patterns that occur in the signals generated by the world around us and try to develop stochastic models for them. We then use these models to detect patterns and structures in these signals, even when, as usually happens, they are ambiguous or obscured by various distortions. We begin with a well-known type of signal and the simplest patterns in it. The "signals" are just the strings of English characters found in any written English text. Any other language would be just as good if it is written with a small alphabet (e.g., Chinese would work a bit differently). Such signals are examples of discrete one-dimensional signals; they are made up of sequences $\{\boldsymbol{a_n}\}$ whose elements lie in a finite set. Strings of written text have, of course, layers and layers of patterns: they break up into words, sentences, and paragraphs with partial repetitions, cross references, systematic changes in letter frequencies, and so forth.

1.2.2. Scope

Statistical Machine translation uses a naïve Bayes classifier function to generate the probability function for different text sentences. It also supports the basic translation systems which can be expanded by adding some extra functions, so the prediction is more accurate. This is one of the interpretations of statistical machine translation systems as an extension towards Bayes decision rule. We will see another interpretation in the form direct maximum entropy model by adding a feature of entropy-based model. This approach can be quite useful in error reduction as well as optimization of the output of machine translation. Several applications of these exist in the field of pattern recognitions.

1.2.3. Principle

A very natural way to start to describe the patterns of some English text is to simply take random substrings of small fixed length n from a large body of text and, if enough data are available, to work out a complete table of frequencies of occurrence of each string of characters of length n**. This probability table defines a stochastic model for strings of arbitrary length** if we assume the Markov property that, conditional on fixing any string of $\{n-1\}$ consecutive characters, the two characters immediately before and after an occurrence of this string in English text are independent. This is called the **nth-order Markov approximation** to the full language of strings.

As Shannon showed very convincingly, these models get increasingly better as n increases, and the **convergence** of these models to reasonable-sounding English can be **measured by entropy**

1.3. NEED FOR SUCH A SYSTEM

Why do we need machine translation? Before answering we will see "what is machine translation?". Machine translation is the task of automatically converting one natural language into another, preserving meaning of input text, and producing fluent text in output language.

Machine translation is useful when you have large amounts of user-generated content that needs to be translated quickly and improves consistency. For example, Machine translation makes it possible to quickly translate and review customer reviews, online comments and social media posts. Human translators can then follow up if necessary.

There are four types of machine translation:
1) Statistical Machine translation
2) Rule based machine translation
3) Hybrid machine translation
4) Neural Machine translation.

Statistical Machine learning (SMT): It is built on the premise of probabilities. For each segment of source text, there are a number of possible target segments with a varying degree if probability of being the correct translation. The segment with highest statistical probability is considered.

SMT works by referring to statistical models that are based on the analysis of large volumes of bilingual text. It aims to determine the correspondence between a word from the source language and a word from the target language. A good example of this is Google Translate.

Rule based machine translation (RBMT): RBMT, on the other hand, translates on the basics of grammatical rules. It conducts a grammatical analysis of the source language and the target language to generate the translated sentence. However, RBMT requires extensive proofreading, and heavy dependence on lexicons means that efficiency is achieved after a long period of time.

Hybrid machine translation (HMT): HMT, as the term indicates, is a blend of RBMT and SMT. It leverages a translation memory, making it far more effective in terms of quality. However, even HMT has its share of drawbacks, the greatest of which is the need for extensive editing. Human translators will be required.

Neural machine translation (NMT): NMT is a type of machine translation that depends on neural network models (based on the human brain) to develop statistical models for the purpose of translation. The primary benefit of NMT is that it provides a single system that can be trained to decipher the source and the target text.

“Which one to choose?”. **SMT is the most preferred approach today**. This is because of the rules of language change which impacts the RMBT approach. These rules need to be constantly updated. SMT, however, does not depend on rules and its system can be constructed in much less time compared to RBMT systems. The training data needed to run SMT is also widely available on the Internet due to the publication of multilingual content.

Finally, its need for a high amount of processing power is also easy to meet now, because of the cloud computing solutions. NMT, on the other hand, is definitely the most advanced option. However,

training models for NMT is an expensive affair, which means small-to-medium businesses will have to consider the cost to profit ratio.

1.4. CHALLENGES AND ISSUES IN SMT

SMT is great for basic translation, but its greatest drawback is that it does not factor in context, which means translations can be often be erroneous. In other words, don't expect high-quality translations.

Problems that statistical machine translation have to deal with include:

Proper Names/ Nouns: This is a big problem. Lots of work on recognizing and translating, sometimes using sounds as a guide.

translation to be done	Human translator (efficiency)	MT system (efficiency)
Person name	60%	82%
Organization name	72%	47%

Statistical anomalies:
Real-world training sets may override translations of, say, proper nouns. An example would be that "I took the train to Berlin" gets mis-translated as "I took the train to Paris" due to an abundance of "train to Paris" in the training set.

Solution for better performance of SMT is providing more training data so that unknown words will be minimized, better modelling using Syntax, Semantics, Pragmatics, compound splitting, Web n-grams etc.

Idioms:
Depending on the corpora used, idioms may not translate "idiomatically". For example, using Canadian Hansard as the bilingual corpus, "hear" may almost invariably be translated to "Bravo!" since in Parliament "Hear, Hear!" becomes "Bravo!".

This problem is connected with word alignment, as in very specific contexts the idiomatic expression may align with words that result in an idiomatic expression of the same meaning in the target language. However, it is unlikely, as the alignment usually doesn't work in any other contexts. For that reason, idioms should only be subjected to phrasal alignment, as they cannot be decomposed further without losing their meaning. This problem is therefore specific for word-based translation.

Out of vocabulary (OOV) words:
SMT systems typically store different word forms as separate symbols without any relation to each other and word forms or phrases that were not in the training data cannot be translated. This might be because of the lack of training data, changes in the human domain where the system is used, or differences in morphology.

Sentence Alignment:
Most translation systems work sentence-by-sentence. In parallel corpora single sentences in one language can be found translated into several sentences in the other and vice versa. But often there

are cross-sentence overlaps. Long sentences may be broken up, short sentences may be merged. There are even some languages that use writing systems without clear indication of a sentence end. Therefore, aligning sentences determine which sentence in language A goes to exactly which sentence in language B is important.

Word Alignment:

To learn e.g. the translation model, however, we need to know which words align in a source-target sentence pair. Solutions are the IBM-Models or the HMM-approach. One of the problems presented is function words that have no clear equivalent in the target language.

For example, when translating from English to German the sentence "John does not live here," the word "does" doesn't have a clear alignment in the translated sentence "John wohnt hier nicht." Through logical reasoning, it may be aligned with the words "wohnt" (as in English it contains grammatical information for the word "live") or "nicht" (as it only appears in the sentence because it is negated) or it may be unaligned.

To create aligned corpora, different methods have been developed:

- Length based
- Dictionary based
- And many more recently

Unknown Words: When there are words which do not appear in the train data but comes in the test data, in these situations SMT gives wrong output. For such cases not to happen, we have to train model on large training data sets.

Different word orders:
Word order in languages differ. Some classification can be done by naming the typical order of subject (S), verb (V) and object (O) in a sentence and one can talk, for instance, of SVO or VSO languages. There are also additional differences in word orders, for instance, where modifiers for nouns are located, or where the same words are used as a question or a statement.
In speech recognition, the speech signal and the corresponding textual representation can be mapped to each other in blocks in order. This is not always the case with the same text in two languages.

For SMT, the machine translator can only manage small sequences of words, and word order has to be thought of by the program designer. Attempts at solutions have included re-ordering models, where a distribution of location changes for each item of translation is guessed from aligned bi-text. Different location changes can be ranked with the help of the language model and the best can be selected.

Recently, Skype voice communicator started testing speech translation. However, machine translation is following technological trends in speech at a slower rate than speech recognition. In fact, some ideas from speech recognition research have been adopted by statistical machine translation.

Chapter 2

LITERATURE SURVEY

2.1 BACKGROUND

The idea behind statistical machine translation comes from information theory. A document is translated according to the probability distribution that a string in the target language (for example, English) is the translation of a string in the source language (for example, French). The problem of modeling the probability distribution has been approached in a number of ways.
One approach which lends itself well to computer implementation is to apply Bayes Theorem, that is, where the translation model is the probability that the source string is the translation of the target string, and the language model is the probability of seeing that target language string.
This decomposition is attractive as it splits the problem into two subproblems.

Finding the best translation is done by picking up the one that gives the highest probability: For a rigorous implementation of this one would have to perform an exhaustive search by going through all strings in the native language. Performing the search efficiently is the work of a machine translation decoder that uses the foreign string, heuristics and other methods to limit the search space and at the same time keeping acceptable quality. This trade-off between quality and time usage can also be found in speech recognition. As the translation systems are not able to store all native strings and their translations, a document is typically translated sentence by sentence, but even this is not enough.

Language models are typically approximated by smoothed n-gram models, and similar approaches have been applied to translation models, but there is additional complexity due to different sentence lengths and word orders in the languages. The statistical translation models were initially word based (Model 1 from IBM Hidden Markov model from Stephan Vogel), but significant advances were made with the introduction of phrase based models. Recent work has incorporated syntax or quasi-syntactic structures.

2.1.1 Basic Probability

We're going to consider that an English sentence e may translate into any French sentence f. Some translations are just more likely than others. Here are the basic notations we'll use:

$p(e)$ -- a priori probability. The chance that e happens. For example, if e is the English string “I like snakes,” then $p(e)$ is the chance that a certain person at a certain time will say “I like snakes” as opposed to saying something else.

$p(f/e)$ -- conditional probability. The chance of f given e. For example, if e is the English string “I like snakes,” and if f is the French string “maison bleue,” then $p(f/e)$ is the chance that upon seeing e, a translator will produce f.

$p(e, f)$ -- joint probability. The chance of e and f both happening. If e and f don't influence each other, then we can write $p(e, f) = p(e) * p(f/e)$

For example, if e stands for "the first roll of the die comes up 5" and f stands for "the second roll of the die comes up 3," then $p(e, f) = p(e) * p(f/e)$ = 1/6 * 1/6 = 1/36. If e and f do influence each other, then we had better write $p(e, f) = p(e) * p(f/e)$. That means: the chance that "e happens" times the chance that "if e happens, then f happens." If e and f are strings that are mutual translations, then there's definitely some influence.

2.1.2 Bayes rule applications in statistical machine translation

Given a French sentence f, we seek English sentence e that maximizes P (e |f). We write this as:
$\underset{e}{argmax}\ p(e/f)$

Read this argmax as follows: "the English sentence e, out of all such sentences, which yields the highest value for $p(e/f)$.

Bayes Rule:

$$p(e/f) = \frac{p(e) * p(f/e)}{p(f)}$$

That means the most likely translation **e** maximizes the product of two terms, (1) the chance that someone would say e in the first place, and (2) if he did say e, the chance that someone else would translate it into f. The noisy channel works like this. We imagine that someone has e in his head, but by the time it gets on to the printed page it is corrupted by "noise" and becomes f.

To recover the most likely e, we reason about
(1) what kinds of things people say any English, and
(2) how English gets turned into French. These are sometimes called "source modeling" and "channel modeling."

People use the noisy channel metaphor for a lot of engineering problems, like actual noise on telephone transmissions.
The noisy channel model using Bayes Rule, we can rewrite the expression for translation:
argmax $p(e/f)$ = argmax $p(e) * p(f/e)$,
e e

where $p(f/e)$ is the translation model and $p(e)$ is the language model

What happened to $\boldsymbol{p(f)}$Since $\boldsymbol{p(f)}$ is constant for a given sentence, we can ignore it.
why 'turn it around' this way? Because then you can use $\boldsymbol{p(e)}$, instead of $\boldsymbol{p(f)}$ and there is lot more data available online for English than for French (or some other language you want to translate to)

Our main goal: Maximize $\boldsymbol{p(e/f)}$ – given an input French sentence, maximize arguments (the words, ordering etc.) to get the best English equivalent sentence.
In other words,
1) maximize $\boldsymbol{p(f/e)}$– probability that the speaker wanted to say it that way, given what I know of the way English Sentences are and how they go to French) (translation model) **and**
2) maximize $\boldsymbol{p(e)}$– probability governing how the English words are properly composed, for which there is plenty of training material (Language model).

2.1.3 Approach to statistical machine translation Architecture

General approach is to build two models and then, given a sentence, 'decode' it by finding the most probable words and rules in two decoding steps:

1. Translation model: Find the best target language word or phrase for each input unit. Result: you will have a bag of words or phrases.

2. Language model: Pick one word or phrase at a time from the bag of words or phrases and build up a sentence.

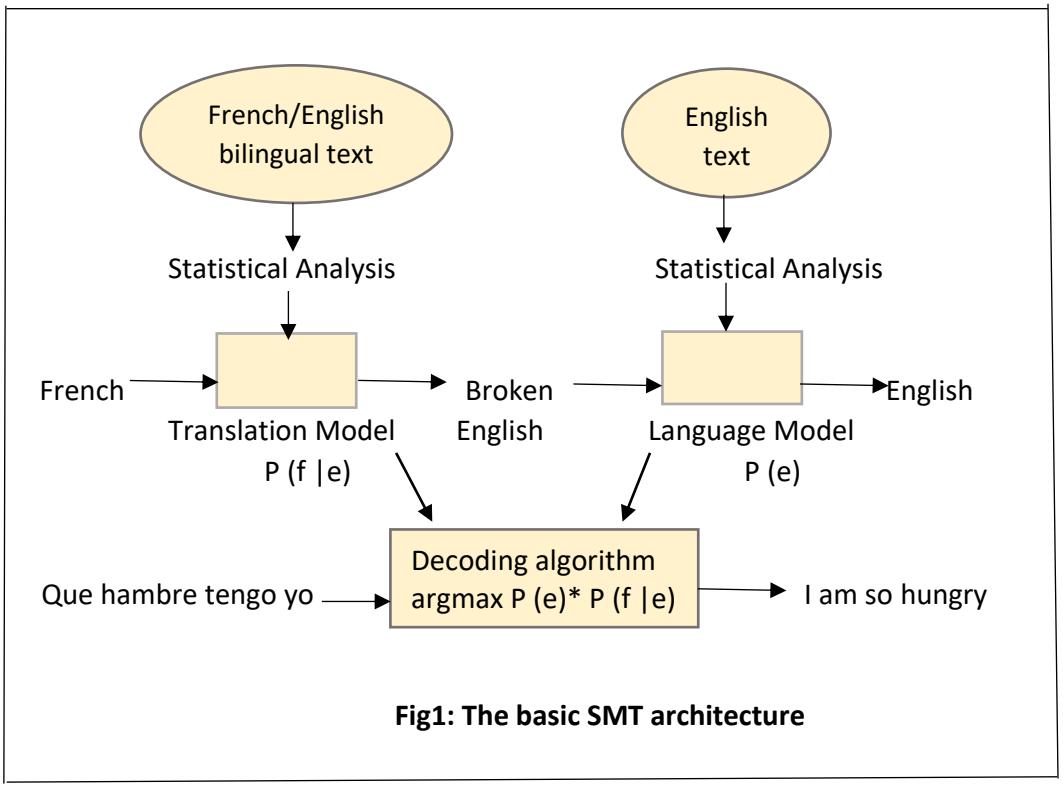


Fig1: The basic SMT architecture

Language model: Use bigrams (and later tri-grams, …., n-grams) of English, with frequency counts normalized to probabilities.

Transition Models: try a series of increasingly complex transition models:

Model 1: word/ phrase alignment: models at first assume simple word-word and phrase-phrase correspondence

Model 2: fertility: how many word(s) a single word translates into (for French and English it is mostly one-to-one)

Model 3: Distortion: changes in absolute word position (French and English have more or less same word order)

Model 4: Relative word group distortion

2.2 LANGUAGE MODEL

Where do all these probability numbers come from?
Later, we'll worry about $\boldsymbol{p}(\boldsymbol{f}/\boldsymbol{e})$. First, we need to build a machine that assigns a probability $\boldsymbol{p}(\boldsymbol{e})$ to each English sentence e. This is called a **language model.** We could build a program that knows a lot about the world, about the types of things people like to discuss, about the grammatical structures that people use when describing certain events and objects, etc. We could type in lots of numbers by hand at various points in the program, and they might get multiplied or added together, at other points.

Implementing this idea is simpler -- just record every sentence that anyone ever says in English. Suppose you record a database of one billion utterances. If the sentence "how's it going?" appears 76,413 times in that database, then we say P (how's it going?) = 76,413/1,000,000,000 = 0.000076413. If enough data are available, to work out a complete table of frequencies of occurrence of each string of characters or words. This probability table defines a stochastic model for strings of arbitrary length.

We can use the Web or the online Wall Street Journal for the data without actual recording since its already been done and that data is readily available for us.

Exercise. What is the probability of: "I like snakes that are not poisonous" and "I like snakes" or "I hate snakes"?

One big problem is that many perfectly good sentences will be assigned a $p(e)$ of zero, because we have never seen them before. This is bad. A great $p(f/e)$ module may give us a nice bag of English words like "not that like are snakes poisonous I." But no matter what order we assign to these words, $p(e)$ always equal zero. That means "like poisonous I are that not snakes" will be considered just as likely a translation as "I like snakes that are not poisonous."

What should the length of words be (n-gram)? What should be done to get rid of zero probabilities? By smoothening, this we will discuss later.

2.2.1 Word Re-ordering in translation

If we reason directly about translation using $p(e/f)$, then our probability estimates had better be very good. On the other hand, if we break things apart using Bayes Rule, then we can theoretically get good translations even if the probability numbers aren't that accurate. For example, suppose we assign a high value to $p(f/e)$ only if the words in f are generally translations of words in e. The words in f may be in any order: we don't care. Well, that's not a very accurate model of how English gets turned into French. Maybe it's an accurate model of how English gets turned into really bad French.

Now let's talk about $p(e)$. Suppose that we assign high value to $p(e)$ only if e is grammatical. That's pretty reasonable, though difficult to do in practice. An interesting thing happens when we observe f and try to come up with the most likely translation e. Every e gets the score $p(e)$ * $p(f/e)$. The factor $p(f/e)$ will ensure that a good e will have words that generally translate to words in f. Various "English" sentences will pass this test.

For example, if the string "the boy runs" passes, then "runs boy the" will also pass. Some word orders will be grammatical and some will not. However, the factor $p(e)$ will lower the score of ungrammatical sentences. In effect, $p(e)$ worries only about English word order so that $p(f/e)$ doesn't have to. That makes $p(f/e)$ easier to build than you might have thought. It only needs to say whether or not a bag of English words corresponds to a bag of French words. This might be done with some sort of bilingual dictionary. In algorithmic terms, this module needs to be able to turn a bag of French words into a bag of English words, and assign a score of $p(f/e)$ to the bag-pair.

Exercise: Put these words in order: "have programming a seen never I language better". This task is called **bag generation**.

What kind of knowledge is being applied here? Do you think machine could do this job? Can you think of a way to automatically test how well a machine is doing, without human checking?

Another Exercise: Put these words in order: "loves John Mary". It can either be "John loves Mary" or "Mary loves John". It seems like $p(f/e)$ needs to know something about word order after all. It can't simply suggest a bag of English words and be done with it. It needs to know a little bit about word order.

2.2.2 Word choice in translation

The $p(e)$ model can also be useful for selecting English translations of French words. For example, suppose there is a French word that either translates as "in" or "on."

Then there may be two English strings with equally good $p(f/e)$ scores:
(1) she is in the end zone,
(2) she is on the end zone.

(Let's ignore other strings like “in zone end the is she” which also get good $p(f/e)$ scores).

Well, the first sentence is much better English than the second, so it should get a better $p(e)$ score, and therefore a better $p(e) * p(f/e)$ score. Remember that the most likely translation is the one with the best $p(e) * p(f/e)$ score.

2.2.3 N-Gram model

we assume the **Markov property** that, conditional on fixing any string of n − 1 consecutive characters, the two characters immediately before and after an occurrence of this string in English text are independent. This is called the nth-order Markov approximation to the full language of strings. As Shannon showed very convincingly, these models get increasingly better as n increases, and the convergence of these models to reasonable-sounding English can be measured by entropy.

For computers, the easiest way to break a string down into components is to consider substrings. An n-word substring is called an n-gram. If n=2, we say bigram. If n=3, we say trigram. If n=1, it is called unigram (it is word of course). If a string has a lot of reasonable n-grams, then maybe it is a reasonable string.
Let B (y | x) be the probability that word y follows word x. We can estimate this probability from online text. We simply divide the number of times we see the phrase “xy” by the number of times we see the word “x”. That's called a conditional bigram probability. Each distinct B (y | x) is called a parameter.

A commonly used n-gram estimator looks like this, (where n = 2 here):

$$B(y/x) = \frac{\text{number} - \text{of} - \text{occurrences}(\text{“xy”})}{\text{number} - \text{of} - \text{occurrences}(\text{“x”})}$$

P (I like snakes that are not poisonous) ~
B (I | start-of-sentence) *
B (like | I) *
B (snakes | like) *
...
B (poisonous | not) *
B (end-of-sentence | poisonous)

In other words, what's the chance that you'll start a sentence with the word “I”? If you did say “I”, what's the chance that you would say the word “like” immediately after? And if you did say “like”, is “snakes” a reasonable next word? And so on.

Actually, this is another case of generative model. This model says that people keep spitting out words one after another, but they can't remember anything except the last word they said. It's called a bigram language model. If we're nice, we might allow for the possibility that people remember the last two words they said. That's called a trigram language model:

$$B(z/xy) = \frac{\text{number} - \text{of} - \text{occurrences}(\text{“xyz”})}{\text{number} - \text{of} - \text{occurrences}(\text{“xy”})}, (\text{where n } = \text{ 3 here.})$$

P (I like snakes that are not poisonous) ~
B (I | start-of-sentence start-of-sentence) *
B (like | start-of-sentence I) *
B (snakes | I like) *
...
B (poisonous | are not) *
B (end-of-sentence | not poisonous) *
B (poisonous | end-of-sentence end-of-sentence)

2.2.4 Smoothing

N-gram models can assign non-zero probabilities to sentences they have never seen before. It's a good thing, like Martha Stewart says. The only way you'll get a zero probability is, if the sentence contains a previously unseen bigram or trigram. That can happen.

In that case, all we can do is smoothing. If “z” never followed “xy” in our text, we might further wonder whether “z” at least followed “y”. If it did, then maybe “xyz” isn't so bad. If it didn't, we might further wonder whether “z” is even a common word or not. If it's not even a common word, then “xyz” should probably get a low probability. Instead of

$$B(z/xy) = \frac{\text{number} - \text{of} - \text{occurrences}(\text{“xyz”})}{\text{number} - \text{of} - \text{occurrences}(\text{“xy”})}$$

we can use

$$B\ (z \mid x\ y)\ =\ 0.95\ *\ \frac{number - of\ -\ occurrences(\text{“}xyz\text{”})}{number - of\ -\ occurrences(\text{“}xy\text{”})}\ +$$
$$0.04\ *\ \frac{number - of\ -\ occurrences\ (\text{“}yz\text{”})}{number - of\ -\ occurrences\ (\text{“}z\text{”})}\ +$$
$$0.008\ *\ \frac{number - of\ -\ occurrences(\text{“}z\text{”})}{total - words - seen}\ +\ \mathbf{0.002}$$

It's handy to use different smoothing coefficients in different situations. You might want 0.95 in the case of xy(z), but 0.85 in another case like ab(c). For example, if “ab” doesn't occur very much, then the counts of “ab” and “abc” might not be very reliable.

Notice that as long as we have that “0.002” in there, then no conditional trigram probability will ever be zero, so **no $\boldsymbol{p(e)}$ will ever be zero**. That means we will assign some positive probability to any string of words, even if it's totally ungrammatical.

We'll have more to say about estimating those smoothing coefficients. This n-gram business is not set in stone. You can come up with all sorts of different generative models for language.

2.2.5 Evaluating models

A model often consists of a generative "story" (e.g., people produce words based on the last two words they said) and a set of parameter values (e.g., B (z | x y) = 0.02). It's usually too hard to set the parameter values by hand, so we look at training data. N-gram models are easy to train -- basically, we just count n-gram frequencies and divide. For one thing, you need to be able to identify the main verb of a training sentence.

How do you know if one model "works better" than another? One way to pit language models against each other is to gather up a bunch of previously unseen English test data, then ask: What is the probability of a certain model (generative story plus particular parameter values), given the test data that we observe? We can write this symbolically as:
P (model | test-data)

Using Bayes rule:

$$\boldsymbol{P}\ (\boldsymbol{model} \mid \boldsymbol{test-data})\ = \frac{\boldsymbol{P}(\boldsymbol{model})\ *\ \boldsymbol{P}\ (\boldsymbol{test-data} \mid \boldsymbol{model})}{\boldsymbol{P}(\boldsymbol{data})}$$

(P(data) is neglected)

Let's suppose that P(model) is the same for all models. That is, without looking at the test data, we have no idea whether "0.95" is a better number than "0.07" deep inside some model. Then, the best model is the one that maximizes P (test-data | model). Fortunately, P (test-data | model) is something that is easy to compute. It's just the same thing as $p(e)$, where e = test-data.

Now, anyone can come up with any crackpot computer program that produces P(e) values, and we can compare it to any other crackpot computer program. It's a bit like gambling. A model assigns bets on all kinds of strings, by assigning them high or low P(e) scores. When the test data is revealed, we see how much the model bet on that string. The more it bet, the better the model.

A trigram model will have higher P(test-set) than a bigram model. Why?
A bigram model will assign reasonably high probability to a string like "I hire men who is good pilots". This string contains good word pairs. The model will lay some significant "bet" on this string. A trigram model won't bet much on this string, though, because B (is | men who) is very small. That means the trigram model has more money to bet on good strings which tend to show up in unseen test data (like "I hire men who are good pilots").

Any model that assigns zero probability to the test data is going to get killed! But we're going to do smoothing anyway, right? About those smoothing coefficients -- there are methods for choosing values that will optimize P (test-data | model). That's not really fair, though. To keep a level playing field, we shouldn't do any kind of training on test data. It's better to divide the original training data into two sets. The first can be used for collecting n-gram frequencies. The second can be used for setting smoothing coefficients. Then we can test all models on previously unseen test data.

2.2.6 PERPLEXITY (ENTROPY CONCEPTS CAN BE USED AS WELL)

If the test data is very long, then an n-gram model will assign a P(e) value that is the product of many small numbers, each less than one. Some n-gram conditional probabilities may be very small themselves. So, $p(e)$ will be tiny and the numbers will be hard to read. A more common way to compare language models is to compute:

$$-\frac{\log_2 p(e)}{N}$$

which is called the perplexity of a model. N is the number of words in the test data. Dividing by N helps to normalize things, so that a given model will have roughly the same perplexity no matter how big the test set is. The logarithm is base two.

As $p(e)$ increases, perplexity decreases. A good model will have a relatively large $p(e)$ and a relatively small perplexity. The lower the perplexity, the better.

Exercise: Suppose a language model assigns the following conditional n-gram probabilities to 3-word test set: 1/4, 1/2, 1/4. Then P(test-set) = 1/4 * 1/2 * 1/4 = 0.03125. Its perplexity is 1.666667. Suppose another language model assigns the following conditional n-gram probabilities to a different 6-word test set: 1/4, 1/2, 1/4, 1/4, 1/2, 1/4. Then its P(test-set) = 0.0009765625. Its perplexity is 1.6666667.

So much for $p(e)$ -- now we can assign a probability estimate to any English string e.

2.3 TRANSLATION MODEL

Next, we need to worry about $p(f/e)$, the probability of a French string f given an English string e. This is called a translation model. Here is a generative "story" about how English strings become French strings. Remember that this $p(f/e)$ will be a module in overall French-to-English machine translation system.

When we see an actual French string f, we want to reason backwards:

(1) what English string e is likely to be uttered, and
(2) likely to subsequently translate to f?

Indirectly, we are looking for the e that maximizes $p(e) * p(f/e)$.

How does English become French? One view is that an English sentence gets converted into predicate logic, or perhaps a conjunction of atomic logical assertions. These assertions clear away all of the "illogical" features of language.

Example: consider two syntactic similar sentences *"John must not go" and "John may not go."*

"John must not go" gets converted to OBLIGATORY(NOT(GO(JOHN))) whereas

"John may not go" gets converted to NOT(PERMITTED(GO(JOHN))).

Notice how the NOT operates differently despite the syntactic similarity of the two sentences. The other half of this view is that predicate logic then gets converted to French. Another view is that an English sentence gets parsed syntactically.

What is meant by parse?
In linguistics, parse means to break down a sentence into its component parts so that the meaning of the sentence can be understood. When parsing a sentence, the reader takes note of the sentence elements and their parts of speech (whether a word is a noun, verb, adjective etc..).

That is, a binary tree diagram is placed on top of the sentence, showing the syntactic relationships between heads and modifiers, for example, subject/verb, adjective/noun, prepositional-phrase/verb-phrase, etc. This tree diagram is then transformed into a French tree ... phrases are swapped around, and English words are replaced by French translations. This view is called "syntactic transfer."

2.3.1 Word - Word Alignments

Now, we turn to the problem of modeling the conditional probability $p(f/e)$ for any French sentence $f = f_1 \dots \dots f_m$ where m is number of French words paired with an English sentence $e = e_1 \dots \dots e_l$ where l is the number of words in English.

Recall that $p(f/e)$ involves two choices: first, a choice of the length m of the French sentence; second, a choice of the words $f_1 \dots \dots f_m$.

We will assume that there is some distribution $p(m/l)$ that models the conditional distribution of French sentence length conditioned on the English sentence length. From now on, we take the length m to be fixed, and our focus will be on modeling the distribution

$$\mathrm{P}\,(f_1 \dots \dots f_m | e_1 \dots \dots e_l, m)$$

i.e., the conditional probability of the words $f_1 \dots \dots f_m$, conditioned on the English string $e_1 \dots \dots e_l$, and the French length m. It is very difficult to model $\mathrm{P}\,(f_1 \dots \dots f_m | e_1 \dots \dots e_l, m)$ directly.

A central idea in the IBM models was to introduce additional alignment variables to the problem. We will have alignment variables $a_1 \dots \dots a_m$—that is, one alignment variable for each French word in the sentence—where each alignment variable can take any value in {0, 1, ……, l}. The alignment variables will specify an alignment for each French word to some word in the English sentence.

2.3.2 Alignment Probabilities

Rather than attempting to define $\mathrm{P}\,(f_1 \dots \dots f_m | e_1 \dots \dots e_l, m)$ directly, we will instead define a conditional distribution:

$$\mathrm{P}\,(f_1 \dots \dots f_m | e_1 \dots \dots e_l, m)$$

over French sequences $f_1 \ldots \ldots f_m$ together with alignment variables $a_1 \ldots \ldots a_m$.
Having defined this model, we can then calculate $\mathrm{P}\ (f_1 \ldots \ldots f_m | e_1 \ldots \ldots e_l, m)$ by summing over the alignment variables ("marginalizing out" the alignment variables):

$$\mathrm{P}\ (f_1 \ldots \ldots f_m | e_1 \ldots \ldots e_l) = \sum_{a_1=0}^{l} \sum_{a_2=0}^{l} \ldots \ldots \ldots \sum_{a_m=0}^{l} \mathrm{P}\ (f_1 \ldots \ldots f_m, a_1 \ldots \ldots a_m | e_1 \ldots \ldots e_l)$$

We now describe the alignment variables in detail. Each alignment variable a_j specifies that the French word fj is aligned to the English word e_{a_j}: we will see soon that intuitively, in the probabilistic model, word fj will be generated from English word e_{a_j}. We define $e_0$to be a special NULL word; so, $a_j = 0$ specifies that word f_j is generated from the NULL word. We will see the role that the NULL symbol plays when we describe the probabilistic model.

As one example, consider a case where l = 6, m = 7, and

e = And the programme has been implemented

f = Le programme a ete mis en application

In this case the length of the French sentence, m, is equal to 7; hence we have alignment variables $a_1, a_2, a_3, \ldots, a_7$. As one alignment (which is quite plausible),

we could have

$$a_1, a_2, a_3, \ldots, a_7 = <2, 3, 4, 5, 6, 6, 6>$$

specifying the following alignment:

Le ⇒ the
Programme ⇒ program
a ⇒ has
ete ⇒ been
mis ⇒ implemented
en ⇒ implemented
application ⇒ implemented

Note that each French word is aligned to exactly one English word.

The alignment is many-to-one: more than one French word can be aligned to a single English word (e.g., mis, en, and application are all aligned to implemented). Some English words may be aligned to zero French words: for example, the word "and" is not aligned to any French word in this example.

Note also that the model is asymmetric, in that there is no constraint that each English word is aligned to exactly one French word: each English word can be aligned to any number (zero or more) French words. We will return to this point.

Another example alignment, we have

$$a_1, a_2, a_3, \ldots, a_7 = <1, 1, 1, 1, 1, 1, 1>$$

specifying the following alignment:

Le ⇒ And
Programme ⇒ And
a ⇒ An
ete ⇒ And
mis ⇒ And
en ⇒ And
application ⇒ And

This is clearly not a good alignment for this example.

We will now discuss about IBM models which directly use the idea of alignment probabilities.

CHAPTER 3
WORKING ON IBM TRANSLATION MODELS

The IBM models were an early approach to SMT, and are not widely used for translation: improved models have been derived in recent work.

However, they will be useful for us, for the following reasons:

1. The models make direct use of the idea of alignments, and as a consequence allows us to recover alignments between French and English words in the training data. The resulting alignment models are of central importance in modern SMT systems.
2. The parameters of the IBM models will be estimated using the Expectation Maximization (EM) algorithm. The EM is widely used in statistical models for NLP and other problem domains.

3.1 IBM Alignment MODEL 1

We now describe a model for the conditional probability

$$\mathrm{P}\,(f_1 \ldots \ldots f_m, a_1 \ldots \ldots a_m | e_1 \ldots \ldots e_l, \mathrm{m})$$

Where English sentence e has l words $e_1 \ldots \ldots e_l$, French sentence F has m words , $f_1 \ldots \ldots f_m$. An alignment a identifies which English word each French word is originated from. Formally, an alignment a is $\{a_1 \ldots \ldots a_m\}$, where each $a_j \in \{0,1,..,l\}$. There are $((l+1)^m)$ possible alignments.

Our goal is to build a model of

$$\mathrm{P}\,(F_1 = f_1 \ldots \ldots f_m,\ A_1 = a_1 \ldots \ldots a_m |\ E_1 = e_1 \ldots \ldots e_l, \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m})$$

As a first step, we can use the chain rule of probabilities to decompose this into two terms:

$$\mathrm{P}\,(F_1 = f_1 \ldots \ldots f_m,\ A_1 = a_1 \ldots \ldots a_m |\ E_1 = e_1 \ldots \ldots e_l, \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m})$$

$$= \mathrm{P}\,(A_1 = a_1 \ldots \ldots a_m |\ E_1 = e_1 \ldots \ldots e_l, \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m}) * \mathrm{P}\,(F_1 = f_1 \ldots \ldots f_m \mid A_1 = a_1 \ldots \ldots a_m, E_1 = e_1 \ldots \ldots e_l, \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m})$$

The first equality is exact, by the chain rule of probabilities.
The second equality corresponds to a very strong independence assumption: namely, that the distribution of the random variable A_i depends only on the values for the random variables L and M (it is independent of the English words $E_1 \ldots \ldots E_l$, and of the other alignment variables).
Also,

$$\mathrm{P}\,(F_1 = f_1 \ldots \ldots f_m |\ E_1 = e_1 \ldots \ldots e_l, \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m})$$

$$= \sum_{a \in A} \mathrm{P}\,(A_1 = a_1 \ldots \ldots a_m |\ E_1 = e_1 \ldots \ldots e_l, \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m}) * \mathrm{P}\,(F_1 = f_1 \ldots \ldots f_m \mid A_1 = a_1 \ldots \ldots a_m, E_1 = e_1 \ldots \ldots e_l, \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m})$$

Where A is the set of all possible alignments

Once we have a model $\mathrm{P}(\mathrm{f}, \mathrm{a}|\mathrm{e}, \mathrm{m}) = \mathrm{P}(\mathrm{a}|\mathrm{e}) * \mathrm{P}(\mathrm{f}|\mathrm{a}, \mathrm{e}, \mathrm{m})$

We can also calculate

$$p(a|f, e, m) = \frac{p(f, a\,|e, m)}{\sum_{a \in A} p(f, a\,|e, m)}$$

For a given f, e pair, we can also compute the most likely alignment,

$$\tilde{a} = \arg\arg max_a\, p(a|f, e, m)$$

We'll now consider these two terms separately.

First, we make the following independence assumptions:

In IBM model 1 all alignments a are equally likely and equal to

$$\frac{1}{(l+1)}$$

$$\mathrm{P}(A_1 = a_1 \ldots \ldots a_m | E_1 = e_1 \ldots \ldots e_l, \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m})$$
$$= \prod_{i=1}^{m} \mathrm{P}(A_i = a_i \,|\, A_1 = a_1, \ldots \ldots A_{i-1} = a_{i-1} | E_1 = e_1, \ldots \ldots E_l = e_l, \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m})$$
$$= \prod_{i=1}^{m} (A_{i-1} = a_{i-1} | \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m})$$
$$= \frac{1}{(l+1)^m}$$

Now, we assume some transitional probabilities i.e, come up with an estimator for $p(f|a, e, m)$

Next, we make the following assumption:

$$p(F_1 = f_1, \ldots \ldots F_m = f_m \mid A_1 = a_1 \ldots \ldots A_m = a_m, E_1 = e_1 \ldots \ldots E_l = e_l, L = l, M = m)$$
$$= \prod_{i=1}^{m} p\left(F_i = f_i \,\middle|\, F_1 = f_1, \ldots F_{i-1} = f_{i-1}, A_1 = a_1 \ldots A_m = a_m, E_1 = e_1 \ldots E_l = e_l, L = l, M = m\right)$$
$$= \prod_{i=1}^{m} p\,(F_i = f_i | E_{a_i} = e_{a_i}) = \prod_{i=1}^{m} \mathrm{t}\,(f_i | e_{a_i})$$

where $\mathrm{t}\,(f_i | e_{a_i})$ is the translation probability for each French string f.

3.1.1 Alignment probabilities for this model

To illustrate this definition, consider the previous example where l = 6, m = 7,

e = And the programme has been implemented

f = Le programme a ete mis en application

and the alignment variables are

$$a_1, a_2, a_3, \dots, a_7 = <2, 3, 4, 5, 6, 6, 6>$$

specifying the following alignment:

Le ⇒ the
Programme ⇒ program
a ⇒ has
ete ⇒ been
mis ⇒ implemented
en ⇒ implemented
application ⇒ implemented

In this case we have

$$\mathrm{P}\,(f_1 \dots \dots f_m, a_1 \dots \dots a_m | e_1 \dots \dots e_l, \mathrm{m}) = \mathrm{t}\,(\mathrm{Le}|\ \mathrm{the}) \times \mathrm{t}\,(\mathrm{Programme}|\ \mathrm{program}) \times \mathrm{t}\,(\mathrm{a}|\ \mathrm{has}) \times \mathrm{t}\,(\mathrm{ete}|\ \mathrm{been}) \times \mathrm{t}\,(\mathrm{mis}|\ \mathrm{implemented}) \times \mathrm{t}\,(\mathrm{en}|\ \mathrm{implemented}) \times \mathrm{t}\,(\mathrm{application}|\ \mathrm{implemented})$$

3.1.2 Generative process for this model 1

To generate a French string f from an English string e:

Step 1:

Pick an alignment a with probability

$$\frac{1}{(l+1)^m}$$

Step 2:

Pick the French words with probability

$$p(f|a, e, m) = \prod_{i=1}^{m} \mathrm{t}\,(f_i|\ e_{a_i})$$

The final result from model 1

$$p(f, a|e, m) = p(a|e, m) * p(f|a, e, m) = \frac{1}{(l+1)^m} \prod_{i=1}^{m} \mathrm{t}\,(f_i|\ e_{a_i})$$

Disadvantages of IBM Model 1:

Model 1 is weak in terms of conducting reordering or adding and dropping words. In most cases, words that follow each other in one language would have a different order after translation, but IBM model 1 treats all kinds of reordering as equally as possible.

Another problem while aligning is the fertility (the notion that input words produce a specific number of output words after translation). In most cases one input word will be translated into one single word, but some words will produce multiple words or even get dropped (produce no words at all). The fertility of the word models addresses this aspect of translation. While additional complexity of models, the main principles of IBM model 1 are constant.

3.2 IBM Alignment MODEL 2

We now describe a model for the conditional probability

$$\mathrm{P}\ (f_1 \ldots \ldots f_m, a_1 \ldots \ldots a_m | e_1 \ldots \ldots e_l, \mathrm{m})$$

The model we describe is usually referred to as IBM model 2:

we will use IBM-M2 as shorthand for this model. Later we will describe how IBM model 1 is a special case of IBM model 2.

The definition is as follows:
Definition 1 (IBM Model 2): An IBM-M2 model consists of a finite set E of English words, a set F of French words, and integers M and L specifying the maximum length of French and English sentences respectively.

The parameters of the model are as follows:

- $t(f/e)$ for any $\mathrm{f} \in \mathrm{F}, \mathrm{e} \in \mathrm{E} \cup \{\mathrm{NULL}\}$. The parameter $t(f/e)$ can be interpreted as the conditional probability of generating French word f from English word e.
- $\mathrm{q}\ (\mathrm{j} \mid \mathrm{i}, \mathrm{l}, \mathrm{m})$ for any $\mathrm{l} \in \{1 \ldots \mathrm{L}\},\ \mathrm{m} \in \{1 \ldots \mathrm{M}\},\ \mathrm{i} \in \{1 \ldots \mathrm{m}\},\ \mathrm{j} \in \{0 \ldots \mathrm{l}\}$. The parameter $\mathrm{q}\ (\mathrm{j} \mid \mathrm{i}, \mathrm{l}, \mathrm{m})$ can be interpreted as the probability of alignment variable a_i taking the value j, conditioned on the lengths l and m of the English and French sentences (distortion parameter).

The IBM model 2 has an additional distortion parameter and translational probability.

3.2.1 Alignment probabilities for this model

Given these definitions, for any English sentence $e_1 \ldots \ldots e_l$ where each $e_j \in \mathrm{E}$, for each length m, we define the conditional distribution over French sentences $f_1 \ldots \ldots f_m$ and alignments $a_1 \ldots \ldots a_m$ as

$$p\ (f_1 \ldots \ldots f_m, a_1 \ldots \ldots a_m | e_1 \ldots \ldots e_l, \mathrm{m}) = \prod_{i=1}^{m} \mathrm{q}\ (a_i\ |\mathrm{i}, \mathrm{l}, \mathrm{m}) * \mathrm{t}\ (f_i |\ e_{a_i})$$

Here we define e_0 to be the NULL word.

To illustrate this definition, consider the previous example where l = 6, m = 7,

e = And the programme has been implemented

f = Le programme a ete mis en application

and the alignment variables are

$$a_1, a_2, a_3, \ldots, a_7 = <2, 3, 4, 5, 6, 6, 6>$$

specifying the following alignment:

Le ⇒ the
Programme ⇒ program
a ⇒ has
ete ⇒ been
mis ⇒ implemented
en ⇒ implemented
application ⇒ implemented

In this case we have

$$P(f_1 \ldots \ldots f_m, a_1 \ldots \ldots a_m | e_1 \ldots \ldots e_l, m) =$$

q (2|1, 6, 7) × t (Le| the)
× q (3|2, 6, 7) × t (Programme| program)
× q (4|3, 6, 7) × t (a| has)
× q (5|4, 6, 7) × t (ete| been)
× q (6|5, 6, 7) × t (mis| implemented)
× q (6|6, 6, 7) × t (en| implemented)
× q (6|7, 6, 7) × t (application| implemented)

Thus, each French word has two associated terms:
first, a choice of alignment variable, specifying which English word the word is aligned to; second, a choice of the French word itself, based on the English word that was chosen in step 1.

For example, for f_5 = mis we first choose a_5 = 6, with probability q (6|5, 6, 7), and then choose the word mis, based on the English word e_6= implemented,

with probability t (mis| implemented).

Note that the alignment parameters, q (j| i, l, m) specify a different distribution
<q (0|i, l, m), q (1|i, l, m), . . ., q (l| i, l, m)>
for each possible value of the tuple i, l, m, where i is the position in the French sentence, l is the length of the English sentence, and m is the length of the French sentence.

This will allow us, for example, to capture the tendency for words close to the beginning of the French sentence to be translations of words close to the beginning of the English sentence. The model is certainly rather simple and naive. However, it captures some important aspects of the data.

3.2.2 Independence Assumptions in IBM Model 2

We now consider the independence assumptions underlying IBM Model 2. Take L to be a random variable corresponding to the length of the English sentence; $E_1 \ldots E_l$ to be a sequence of random variables corresponding to the words in the English sentence; M to be a random variable corresponding to the length of the French sentence; and $F_1 \ldots F_m$ and $A_1 \ldots A_m$ to be sequences of French words, and alignment variables.

Our goal is to build a model of

$$\mathrm{P}\,(F_1 = f_1 \ldots \ldots f_m,\ A_1 = a_1 \ldots \ldots a_m \mid E_1 = e_1 \ldots \ldots e_l, \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m})$$

As a first step, we can use the chain rule of probabilities to decompose this into two terms:

$$\mathrm{P}\,(F_1 = f_1 \ldots \ldots f_m,\ A_1 = a_1 \ldots \ldots a_m \mid E_1 = e_1 \ldots \ldots e_l, \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m})$$

$$= \mathrm{P}\,(A_1 = a_1 \ldots \ldots a_m \mid E_1 = e_1 \ldots \ldots e_l, \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m}) \quad *$$

$$\mathrm{P}\,(F_1 = f_1 \ldots \ldots f_m \mid A_1 = a_1 \ldots \ldots a_m, E_1 = e_1 \ldots \ldots e_l, \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m})$$

We'll now consider these two terms separately. First, we make the following independence assumptions:

$$\mathrm{P}\,(A_1 = a_1 \ldots \ldots a_m \mid E_1 = e_1 \ldots \ldots e_l, \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m})$$
$$= \prod_{i=1}^{m} \mathrm{P}\,(A_i = a_i \mid A_1 = a_1, \ldots \ldots A_{i-1} = a_{i-1} \mid E_1 = e_1, \ldots \ldots E_l = e_l, \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m})$$
$$= \prod_{i=1}^{m} (A_{i-1} = a_{i-1} \mid \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m})$$

The first equality is exact, by the chain rule of probabilities.
The second equality corresponds to a very strong independence assumption: namely, that the distribution of the random variable A_i depends only on the values for the random variables L and M (it is independent of the English words $E_1 \ldots \ldots E_l$, and of the other alignment variables). Finally, we make the assumption that

$$p(A_i = a_i \mid \mathrm{L} = \mathrm{l}, \mathrm{M} = \mathrm{m}) = q(a_i \mid i, l, m)$$

where each $q(a_i \mid i, l, m)$ is a parameter of our model.
Next, we make the following assumption:

$$p\,(\,F_1 = f_1, \ldots\ldots\, F_m = f_m \mid\, A_1 = a_1 \ldots\ldots\, A_m = a_m, E_1 = e_1 \ldots\ldots E_l = \, e_l, L = l, M = m)$$
$$= \prod_{i=1}^{m} p\left(\, F_i = f_{\,i} \,\middle|\, F_1 = f_{\,1}, \ldots\, F_{i-1} = f_{i-1}, A_1 = a_1 \ldots\, A_m = a_m, E_1 = e_{\,1} \ldots E_l = \, e_l, L = l, M = m\right)$$
$$= \prod_{i=1}^{m} p\,(F_i = f_i |\, E_{a_i} = \, e_{a_i})$$

The first step is again exact, by the chain rule. In the second step, we assume that the value for F_i depends only on E_{ai}: i.e., on the identity of the English word to which F_i is aligned. Finally, we make the assumption that for all i,

$p\,(F_i = f_i \big|\, E_{a_i} = \, e_{a_i}) = t\big(f_{\,i} \big|\, e_{a_i}\big)$ where each $t\big(f_{\,i} \big|\, e_{a_i}\big)$ is a parameter of our model.

3.2.3 Generative process for this model 2

To generate a French string f from an English string e:

Step 1:

Pick an alignment a = $\{a_1, \ldots\ldots a_m\}$ with probability

$$\prod_{i=1}^{m} \mathrm{q}(a_i \mid \mathrm{i}, \mathrm{l}, \mathrm{m})$$

Step 2:

Pick the French words with probability

$$p(f|a, e, m) = \prod_{i=1}^{m} \mathrm{t}\,(f_i |\, e_{a_i})$$

The final result from model 1

$$p(f, a|e, m) = p(a|e, m) * p(f|a, e, m) = \prod_{i=1}^{m} \mathrm{q}(a_i \mid \mathrm{i}, \mathrm{l}, \mathrm{m})\ \mathrm{t}\,(f_i |\, e_{a_i})$$

3.2.4 Applying IBM Model 2

The next section describes a parameter estimation algorithm for IBM Model 2. Before getting to this, we first consider an important question: what is IBM Model 2 useful for?

The original motivation was the full machine translation problem. Once we have estimated parameters $\mathrm{q}\ (\mathrm{j} \mid \mathrm{i}, \mathrm{l}, \mathrm{m})$ and $\mathrm{t}\ (\mathrm{f}|\ \mathrm{e})$ from data, we have a distribution: $p(f, a \mid e)$

for any French sentence f, alignment sequence a, and English sentence e; from this we can derive a distribution

$$p(f|\, e) = \sum_{a} p(f, a \mid e)$$

Finally, assuming we have a language model P(e), we can define the translation of any French sentence f to be

$$\underset{e}{argmax}\ p(e)\ *\ p\ (f|\ e)$$

where the argmax is taken over all possible English sentences. The problem of finding the argmax in Equation is often referred to as the decoding problem. Solving the decoding problem is a computationally very hard problem, but various approximate methods have been derived.

In reality, however, IBM Model 2 is not a particularly good translation model. In later lectures we'll see alternative, state-of-the-art, models that are far more effective.

The IBM models are, however, still crucial in modern translation systems, for two reasons: 1. The lexical probabilities $t(f|e)$ are directly used in various translation systems.
2. Most importantly, the alignments derived using IBM models are of direct use in building modern translation systems.

Recovering Alignments:
Let's consider the second point in more detail. Assume that we have estimated our parameters $t(f|e)$ and $q(|i,l,m)$ from a training corpus (using the parameters estimation algorithm described in the next section). Given any training example consisting of an English sentence e paired with a French sentence f, we can then find the most probably alignment under the model:

$$\underset{a_1 \ldots\ldots a_m}{argmax}\ \mathrm{P}\ (a_1 \ldots \ldots a_m|f_1 \ldots \ldots f_m, e_1 \ldots \ldots e_l, \mathrm{m})$$

Because the model takes such as simple form, finding the solution to above Equation is straightforward. In fact, a simple derivation shows that we simply define

$$a_i = \underset{\mathrm{j}\epsilon\{0\ldots\mathrm{l}\}}{argmax}\mathrm{q}(\mathrm{j}\mid \mathrm{i}, \mathrm{l}, \mathrm{m})\ \times\ \mathrm{t}\ (f_i \mid e_j)$$

for i = 1 . . . m. So, for each French word i, we simply align it to the English position j which maximizes the product of two terms: first, the alignment probability $\mathrm{q}(\mathrm{j}\mid \mathrm{i}, \mathrm{l}, \mathrm{m})$; and second, the translation probability $\mathrm{t}\ (f_i \mid e_j)$.

CHAPTER 4

PARAMETER ESTIMATION

4.1 ESTIMATING PARAMETERS

This section describes methods for estimating the $t(f|e)$ parameters and q (j| i, l, m) parameters from translation data. We consider two scenarios: first, estimation with fully observed data; and second, estimation with partially observed data. The first scenario is unrealistic, but will be a useful warm-up before we get to the second, more realistic case.

4.1.1 Parameter estimation with fully observed data

We now turn to the following problem:

how do we estimate the parameters $t(f|e)$ and $\mathrm{q(j \mid i, l, m)}$ of the model? We will assume that we have a training corpus $\{f^{(k)}, e^{(k)}\}_{k=1}^{n}$ of translations.

Note however, that a crucial piece of information is missing in this data: *we do not know the underlying alignment for each training example.*

In this sense we will refer to the data being only partially observed, because some information—i.e., the alignment for each sentence—is missing. Because of this, we will often refer to the alignment variables as being hidden variables. In spite of the presence of hidden variables, we will see that we can in fact estimate the parameters of the model.

Note that we could presumably employ humans to annotate data with underlying alignments (in a similar way to employing humans to annotate underlying parse trees, to form a treebank resource). However, we wish to avoid this because manual annotation of alignments would be an expensive task, taking a great deal of time for reasonable size translation corpora—moreover, each time we collect a new corpus, we would have to annotate it in this way.

In this section, as a warm-up for the case of partially-observed data, we will consider the case of fully-observed data, where each training example does in fact consist of a triple

$$\left(f^{(k)}, e^{(k)}, a^{(k)}\right), \text{ where}$$

$$f^{(k)} = f_1^{(k)}, \ldots\ldots, f_{m_k}^{(k)} \text{ is a French sentence,}$$

$$e^{(k)} = e_1^{(k)}, \ldots\ldots, e_{l_k}^{(k)} \text{ is an English sentence,}$$

$$a^{(k)} = a_1^{(k)}, \ldots\ldots, a_{m_k}^{(k)} \text{ is a sequence of alignment variables.}$$

Solving this case will be be useful in developing the algorithm for partially-observed data.

The estimates for fully-observed data are simple to derive. Define $c(e, f)$ to be the number of times word e is aligned to word f in the training data, and c(e) to be the number of times that e is aligned to any French word. In addition, define $c(j|i, l, m)$ to be the number of times we see an English sentence of length l, and a French sentence of length m, where word i in French is aligned to word j in English. Finally, define $c(i, l, m)$ to be the number of times we see an English sentence of length l together with a French sentence of length m. Then the maximum-likelihood estimates are

$$t_{ML}(f/e) = \frac{c(e,f)}{c(e)}$$

$$q_{ML}(j/i,l,m) = \frac{c(j\,|i,l\,,m)}{c(i,l,m)}$$

Figure 2 shows an algorithm for parameter estimation with fully observed data.

Input: A training corpus $(f^{(k)}, e^{(k)}, a^{(k)})$ for all $k = 1,2, \ldots \ldots, n$ where $f^{(k)} = f_1^{(k)}, \ldots \ldots, f_{m_k}^{(k)}$ is a French sentence, $e^{(k)} = e_1^{(k)}, \ldots \ldots, e_{l_k}^{(k)}$ is an English sentence, $a^{(k)} = a_1^{(k)}, \ldots \ldots, a_{m_k}^{(k)}$ is a sequence of alignment variables.

Algorithm:

- Set of all counts $c(\ldots) = 0$
- For $k = 1 \ldots . n$
  - For $i = 1 \ldots \ldots m_k$
    - For $j = 1 \ldots \ldots l_k$

$$c(e_j^{(k)}, f_i^{(k)}) \leftarrow c\left(e_j^{(k)}, f_i^{(k)}\right) + \delta(k,i,j)$$

$$c\left(e_j^{(k)}\right) \leftarrow c\left(e_j^{(k)}\right) + \delta(k,i,j)$$

$$c(j|i,l,m) \leftarrow c(j|i,l,m) + \delta(k,i,j)$$

$$c(i,l,m) \leftarrow c(i,l,m) + \delta(k,i,j)$$

$$where\ \delta(k,i,j) = 1\ if\ a_i^{(k)} = j, 0\ otherwise.$$

Output:

$$t_{ML}(f/e) = \frac{c(e,f)}{c(e)}$$

$$q_{ML}(j/i,l,m) = \frac{c(j\,|i,l\,,m)}{c(i,l,m)}$$

Figure 2: parameter estimation algorithm for fully observed data

Thus, to estimate parameters we simply compile counts from the training corpus, then take ratios of these counts.
The algorithm for partially-observed data will be a direct modification of this algorithm. The algorithm considers all possible French/English word pairs in the corpus, which could be aligned: i.e., all possible (k, i, j) tuples where $k \in \{1 \dots n\}, i \in \{1 \dots m_k\}, and\ j \in \{0 \dots l_k\}$.

For each such pair of words, we have $a_i^{(k)} = j$ if the two words are aligned. In this case we increment the relevant $c(e, f), c(e), c(j|i, l, m)$ and $c(i, l, m)$ counts.

If $a_i^{(k)} \neq j$ then the two words are not aligned, and no counts are incremented.

4.1.2 Parameter estimation with partially observed data

We now consider the case of partially-observed data, where the alignment variables $a^{(k)}$ are not observed in the training corpus. The algorithm for this case is shown in figure 2.

There are two important differences for this algorithm from the algorithm in figure 1:

- The algorithm is iterative. We begin with some initial value for the t and q parameters: for example, we might initialize them to random values. At each iteration we first compile some "counts" $c(e), c(e, f), c(j|i, l, m)$ and $c(i, l, m)$ based on the data together with our current estimates of the parameters. We then re-estimate the parameters using these counts, and iterate.
- The counts are calculated using a similar definition to that in figure 1, but with one crucial difference: rather than defining $\delta(k, i, j) = 1$ if $a_i^{(k)} = j$, 0 otherwise \

we use the definition

$$\delta(k, i, j) = \frac{\text{q(j | i, } l_k, m_k) \text{ t } (f_i^{(k)} | e_j^{(k)})}{\sum_{j=0}^{l_k} \text{q(j | i, } l_k, m_k) \text{ t } (f_i^{(k)} | e_j^{(k)})}$$

where the q and t values are our current parameter estimates.

Let's consider this last definition in more detail. We can in fact show the following identity:

$$p\ (A_i = j \mid e_1, \dots \dots e_l, f_1, \dots \dots f_m, \text{m}) = \frac{\text{q(j | i, } l_k, m_k) \text{ t } (f_i^{(k)} | e_j^{(k)})}{\sum_{j=0}^{l_k} \text{q(j | i, } l_k, m_k) \text{ t } (f_i^{(k)} | e_j^{(k)})}$$

where $p\ (A_i = j \mid e_1, \dots \dots e_l, f_1, \dots \dots f_m, \text{m})$ is the conditional probability of the alignment variable a_i taking the value j, under the current model parameters. Thus, we have effectively filled in the alignment variables probabilistically, using our current parameter estimates. This in contrast to the fully observed case, where we could simply define $\delta(k, i, j) = 1$ if $a_i^{(k)} = j$, and 0 otherwise.

Input: A training corpus $(f^{(k)}, e^{(k)})$ for all $k = 1,2, \ldots\ldots, n$ where $f^{(k)} = f_1^{(k)}, \ldots\ldots, f_{m_k}^{(k)}$ is a French sentence, $e^{(k)} = e_1^{(k)}, \ldots\ldots, e_{l_k}^{(k)}$ is an English sentence.

Initialization: Initialize $t(f|e)$ and $q(j|i,l,m)$ parameters (e.g., to random values).

Algorithm:

- Set of all counts $c(\ldots) = 0$
- For $k = 1\ldots.n$
  - For $i = 1\ldots\ldots m_k$
    * For $j = 1\ldots\ldots l_k$

$$c(e_j^{(k)}, f_i^{(k)}) \leftarrow c\left(e_j^{(k)}, f_i^{(k)}\right) + \delta(k,i,j)$$

$$c\left(e_j^{(k)}\right) \leftarrow c\left(e_j^{(k)}\right) + \delta(k,i,j)$$

$$c(j|i,l,m) \leftarrow c(j|i,l,m) + \delta(k,i,j)$$

$$c(i,l,m) \leftarrow c(i,l,m) + \delta(k,i,j)$$

$$where\ \delta(k,i,j) = \frac{\mathrm{q(j \mid i}, l_k, m_k)\ \mathrm{t}\ (f_i^{(k)} \mid e_j^{(k)})}{\sum_{j=0}^{l_k} \mathrm{q(j \mid i}, l_k, m_k)\ \mathrm{t}\ (f_i^{(k)} \mid e_j^{(k)})}$$

Set- --

$$t_{ML}(f/e) = \frac{c(e,f)}{c(e)}$$

$$q_{ML}(j/i,l,m) = \frac{c(j\ |i,l\ ,m)}{c(i,l,m)}$$

Output: parameters $t(f|e)$ and $q(j|i,l,m)$

Figure 3: parameter estimation algorithm for IBM model 2 for the case of partially observed data

As an example, consider our previous example where l = 6, m = 7, and

$e^{(k)}$ = And the programme has been implemented

$f^{(k)}$ = Le programme a ete mis en application

The value for $\delta(k, 5, 6)$ for this example would be the current model's estimate of the probability of word f_5 being aligned to word e6 in the data. It would be calculated as

$$\delta(k, 5,6) = \frac{\mathrm{q}(6|5, 6, 7) \times \mathrm{t}(\mathrm{mis}|\mathrm{implemented})}{\sum_{j=0}^{6} \mathrm{q}(\mathrm{j}|5, 6, 7) \times \mathrm{t}(\mathrm{mis}|e_j)}$$

Thus, the numerator takes into account the translation parameter t (mis| implemented) together with the alignment parameter $q(6|5, 6, 7)$; the denominator involves a sum over terms, where we consider each English word in turn.

The algorithm in figure 2 is an instance of the expectation-maximization (EM) algorithm. The EM algorithm is very widely used for parameter estimation in the case of partially-observed data. The counts $c(e), c(e, f)$ and so on are referred to as expected counts, because they are effectively expected counts under the distribution defined by the model.

$$p\ (a_1, \ldots \ldots a_m \mid f_1, \ldots \ldots f_m, e_1, \ldots \ldots e_l, \mathrm{m})$$

In the first step of each iteration, we calculate the expected counts under the model.
In the second step, we use these expected counts to re-estimate the t and q parameters.
We iterate this two-step procedure until the parameters converge (this often happens in just a few iterations).

4.1.3 EM Algorithm

Soon we'll trace an example run of the EM algorithm, on some simple data. But first, we'll consider the following question: *how can we justify the algorithm? What are its formal guarantees, and what function is it optimizing?*

In this section we'll describe **how the EM algorithm is attempting to find the maximum-likelihood estimates for the data**. For this we'll need to introduce some notation, and in particular, we'll need to carefully specify what exactly is meant by maximum-likelihood estimates for IBM model 2.

First, consider the parameters of the model.

There are two types of parameters:

the translation parameters $t(f|e)$,
and the alignment parameters $q(j|i, l, m)$.

We will use t to refer to the vector of translation parameters, and q to refer to the vector of alignment parameters,

$$t\ = \{t(f|e): f\ \in\ F, e\ \in\ E\ \cup\ \{NULL\}\}$$

$$q = \{q(j|i,l,m): l \in \{1 \ldots L\}, m \in \{1 \ldots M\}, j \in \{0 \ldots l\}, i \in \{1 \ldots m\}\}.$$

We will use T to refer to the parameter space for the translation parameters—that is, the set of valid settings for the translation parameters, defined as follows:

$$T = \{t : \forall e, f, t(f|e) \geq 0; \forall e \in E \cup \{NULL\}, \sum_{f \in F} t(f|e) = 1\}$$

and we will use Q to refer to the parameter space for the alignment parameters,

$$Q = \{q : \forall j, i, l, m, q(j|i,l,m) \geq 0; \forall i, l, m, X\ l\ j = 0\ q(j|i,l,m) = 1\}$$

Next, consider the probability distribution under the model. This depends on the parameter settings t and q. We will introduce notation that makes this dependence explicit. We write

$$p(f, a|e, m; t, q) = \prod_{i=1}^{m} \mathrm{q}(a_i|\ \mathrm{i}, \mathrm{l}, \mathrm{m}) \times \mathrm{t}\ (f_i \mid e_{a_i})$$

as the conditional probability of a French sentence $f_1, \ldots \ldots f_m$, with alignment variables $a_1, \ldots \ldots a_m$ conditioned on an English sentence $e_1 \ldots \ldots e_l$, and the French sentence length m.

The function $p(f, a|e, m; t, q)$ varies as the parameter vectors t and q vary, and we make this dependence explicit by including t and q after the ";" in this expression.
As we described before, we also have the following distribution:

$$p(f|e, m; t, q) = \sum_{a \in A(l,m)} p(f, a|e, m; t, q)$$

where $A(l, m)$ is the set of all possible settings for the alignment variables, given that the English sentence has length l, and the French sentence has length m:

$$A(l, m) = \{(a_1, \ldots \ldots a_m) : a_j \in \{0 \ldots l\}\ for\ j = 1 \ldots m\}$$

So $p(f|e, m; t, q)$ is the conditional probability of French sentence f, conditioned on e and m, under parameter settings t and q. Now consider the parameter estimation problem.

We have the following set-up:

- The input to the parameter estimation algorithm is a set of training examples, $(f^{(k)}, e^{(k)})$, for k = 1 . . . n.
- The output of the parameter estimation algorithm is a pair of parameter vectors $t \in T, q \in Q$.

So how should we choose the parameters t and q?

We first consider a single training example, $(f^{(k)}, e^{(k)})$, for some $k \in \{1 \ldots n\}$. For any parameter settings t and q, we can consider the probability under the model.

$$p(f^{(k)} \mid e^{(k)}, m_k;\ t, q)$$

As we vary the parameters t and q, this probability will vary. Intuitively, a good model would make this probability as high as possible.

Now consider the entire set of training examples. For any parameter settings t and q, we can evaluate the probability of the entire training sample, as follows:

$$\prod_{k=1}^{n} p(f^{(k)} \mid e^{(k)}, m_k;\ t, q)$$

Again, this probability varies as the parameters t and q vary; intuitively, we would like to choose parameter settings t and q which make this probability as high as possible.

4.1.4 Maximum-likelihood (ML) estimates

This leads to the following definition:

Definition 2 (Maximum-likelihood (ML) estimates for IBM model 2): The ML estimates for IBM model 2 are

$$(t_{ML}, q_{ML}) = argmax_{t \in T, q \in Q}\ L(t, q)$$

where

$$L(t, q) = \log \prod_{k=1}^{n} p(f^{(k)} \mid e^{(k)}, m_k;\ t, q)$$

$$= \sum_{k=1}^{n} \log p(f^{(k)} \mid e^{(k)}, m_k;\ t, q)$$

$$= \sum_{k=1}^{n} \log \sum_{\mathrm{a} \in \mathrm{A}(l_{k,}, m_k)} p(f^{(k)} \mid e^{(k)}, m_k;\ t, q)$$

We will refer to the function $L(t, q)$ as the log-likelihood function.

Under this definition, the ML estimates are defined as maximizing the function

$$\log \prod_{k=1}^{n} p(f^{(k)} \mid e^{(k)}, m_k;\ t, q)$$

It is important to realise that this is equivalent to maximizing

$$\prod_{k=1}^{n} p(f^{(k)} \mid e^{(k)}, m_k;\ t, q)$$

because log is a monotonically increasing function, hence maximizing a function $log\ f(t,q)$ is equivalent to maximizing $f(t,q)$. The log is often used because it makes some mathematical derivations more convenient.

We now consider the function $L(t,q)$ which is being optimized. This is actually a difficult function to deal with: for one thing, there is no analytical solution to the optimization problem

$$(t,q) \ = \ argmax_{t\in T, q\in Q}\ L(t,q) \quad \text{--- (3)}$$

By an "analytical" solution, we mean a simple, closed-form solution. As one example of an analytical solution, in language modeling, we found that the maximum-likelihood estimates of trigram parameters were

$$q_{ML}(w|u,v) = \frac{count(u,v,w)}{count(u,v)}$$

Unfortunately, there is no similar simple expression for parameter settings that maximize the expression in Eq. 3.

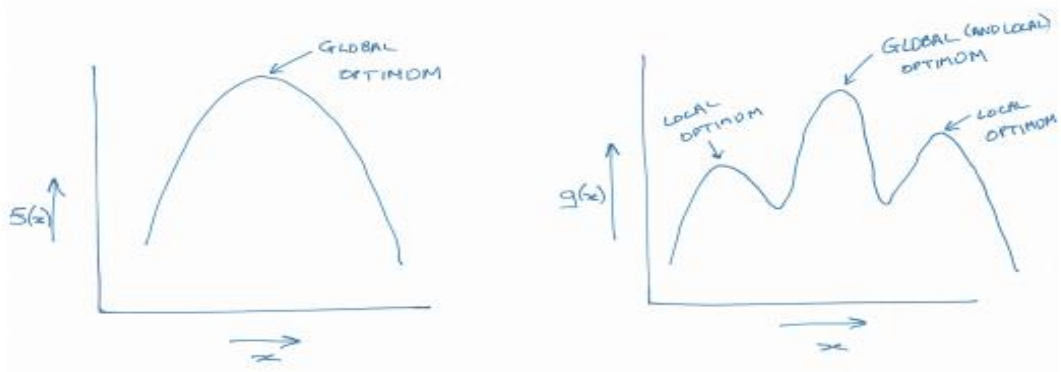


Figure 4: Examples of convex and non-convex functions in a single dimension. On the left, f(x) is convex. On the right, g(x) is non-convex

A second difficulty is that $L(t,q)$ is not a convex function. Figure 3 shows examples of convex and non-convex functions for the simple case of a function $f(x)$ where x is a scalar value (as opposed to a vector).

A convex function has a single global optimum, and intuitively, a simple hill-climbing algorithm will climb to this point. In contrast, the second function in figure 3 has multiple "local" optima, and

intuitively a hill-climbing procedure may get stuck in a local optimum which is not the global optimum.

The formal definitions of convex and non-convex functions are beyond the scope of this note. However, in brief, there are many results showing that convex functions are "easy" to optimize (i.e., we can design efficient algorithms that find the *argmax*), whereas non-convex functions are generally much harder to deal with (i.e., we can often show that finding the *argmax* is computationally hard, for example it is often NP-hard). In many cases, the best we can hope for is that the optimization method finds a local optimum of a non-convex function.

In fact, this is precisely the case for the EM algorithm for model 2. It has the following guarantees:

4.1.5 Properties of EM Algorithm

Theorem 1 (Convergence of the EM algorithm for IBM model 2):
We use $t(s)$ and $q(s)$ to refer to the parameter estimates after s iterations of the EM algorithm, $t(0)$ and $q(0)$ to refer to the initial parameter estimates. Then for any $s \geq 1$, we have

$$\mathrm{L(t\,(s)\,,q(s)\,)} \geq \mathrm{L(t\,(s-1),q(s-1))}$$

Furthermore, under mild conditions, in the limit as s → ∞, the parameter estimates $(t(s)\,, q(s))$ converge to a local optimum of the log-likelihood function.

Later in the discussion we will consider the EM algorithm in much more detail: we will show that it can be applied to a quite broad range of models in NLP, and we will describe its theoretical properties in more detail. Though, this convergence theorem is the most important property of the algorithm. Above Equation states that the log-likelihood is strictly non-decreasing: at each iteration of the EM algorithm, it cannot decrease. However, this does not rule out rather uninteresting cases, such as

$$L(t(s), q(s)) = L(t(s-1), q(s-1)) \; for\; all\; s.$$

The second condition states that the method does in fact converge to a local optimum of the log-likelihood function.

One important consequence of this result is the following: the EM algorithm for IBM model 2 may converge to different parameter estimates, depending on the initial parameter values $t(0)$ and $q(0)$. This is because the algorithm may converge to a different local optimum, depending on its starting point. In practice, this means that some care is often required in initialization (i.e., choice of the initial parameter values).

4.2 TRANSFERING PARAMETERS FROM ONE MODEL TO ANOTHER

Initialization using IBM Alignment Model 1 and using model 2 parameters. As described in the previous section, the EM algorithm for IBM model 2 may be sensitive to initialization: depending on the initial values, it may converge to different local optima of the log-likelihood function. Because of this, in practice the choice of a good heuristic for parameter initialization is important.

Any set of parameters for Model 1 can be re-interpreted as a set of parameters for Model 2. Model 2 does not have a single local maximum. We can use the parameters estimated from Model 1, to instantiate the estimation process for Model 2. We describe IBM Model 1, and the initialization method based on IBM Model 1, in this section. Recall that in IBM model 2, we had parameters

$$q(j \mid i, l, m)$$

which are interpreted as the conditional probability of French word fi being aligned to English word e $_j$, given the French length m and the English length l. In IBM Model 1, we simply assume that for all i, j, l, m,

$$q(j \mid i, l, m) = \frac{1}{l+1}$$

Thus, there is a uniform probability distribution over all $l + 1$ possible English word's (recall that the English sentence is $e_1 \ldots \ldots e_l$, and there is also the possibility that j = 0, indicating that the French word is aligned to e_0 = NULL so, each French word can be aligned to $l + 1$ English words).

This leads to the following definition:

Definition 3 (IBM Model 1) An IBM-M1 model consists of a finite set E of English words, a set F of French words, and integers M and L specifying the maximum length of French and English sentences respectively.

The parameters of the model are as follows:

- $t(f|e)$ for any f ∈ F, e ∈ E ∪ {NULL}. The parameter $t(f|e)$ can be interpreted as the conditional probability of generating French word f from English word e.

4.2.1 Working of the model 1 with model 2 parameters

Given these definitions, for any English sentence $e_1 \ldots \ldots e_l$ where each $e_j \in \mathrm{E}$, for each length m, we define the conditional distribution over French sentences $f_1 \ldots f_m$ and alignments $a_1 \ldots a_m$ as

$$p\,(f_1 \ldots \ldots f_m, a_1 \ldots \ldots a_m | e_1 \ldots \ldots e_l, \mathrm{m}) = \prod_{i=1}^{m} \frac{1}{l+1} * \mathrm{t}\,(f_i |\, e_{a_i})$$

Here we define e_0 to be the NULL word.

The parameters of IBM Model 1 can be estimated using the EM algorithm, which is very similar to the algorithm for IBM Model 2. The algorithm is shown in figure 4. The only change from the algorithm for IBM Model 2 comes from replacing

$$\delta(k, i, j) = \frac{\text{q(j | i, } l_k, m_k) \text{ t } (f_i^{(k)} | e_j^{(k)})}{\sum_{j=0}^{l_k} \text{q(j | i, } l_k, m_k) \text{ t } (f_i^{(k)} | e_j^{(k)})}$$

with

$$\delta(k, i, j) = \frac{\frac{1}{(l^{(k)}+1)} \text{ t } (f_i^{(k)} | e_j^{(k)})}{\sum_{j=0}^{l_k} \frac{1}{(l^{(k)}+1)} * \text{ t } (f_i^{(k)} | e_j^{(k)})} = \frac{\text{t } (f_i^{(k)} | e_j^{(k)})}{\sum_{j=0}^{l_k} \text{t } (f_i^{(k)} | e_j^{(k)})}$$

reflecting the fact that in Model 1 we have

$$\text{q(j | i, } l_k, m_k) = \frac{1}{(l^{(k)}+1)}$$

A key property of IBM Model 1 is the following:

Proposition 1 Under mild conditions, the EM algorithm in figure 4 converges to the global optimum of the log-likelihood function under IBM Model 1.

Thus, for IBM Model 1, we have a guarantee of convergence to the global optimum of the log-likelihood function. Because of this, the EM algorithm will converge to the same value, regardless of initialization.

4.2.2 Parameter estimation for model 2 using model 1 approximation

This suggests the following procedure for training the parameters of IBM Model 2:

1) Estimate the t parameters using the EM algorithm for IBM Model 1, using the algorithm in figure 4.
2) Estimate parameters of IBM Model 2 using the algorithm in figure 2.

To initialize this model, use:
1) the $t(f|e)$ parameters estimated under IBM Model 1, in step 1;
2) random values for the $q(j|i,l,m)$ parameters. Intuitively, if IBM Model 1 leads to reasonable estimates for the t parameters, this method should generally perform better for IBM Model 2.

This is often the case in practice. Parameter estimation for IBM Model 2, can also be done using this heuristic.

Input: A training corpus $(f^{(k)}, e^{(k)})$ for all $k = 1,2, \dots \dots, n$ where $f^{(k)} = f_1^{(k)}, \dots \dots, f_{m_k}^{(k)}$ is a French sentence, $e^{(k)} = e_1^{(k)}, \dots \dots, e_{l_k}^{(k)}$ is an English sentence.

Initialization: Initialize $t(f|e)$ parameters (e.g., to random values).

Algorithm:

- Set of all counts $c(\dots) = 0$
- For $k = 1 \dots . n$
  - For $i = 1 \dots \dots . m_k$
    - For $j = 1 \dots \dots . l_k$

$$c(e_j^{(k)}, f_i^{(k)}) \leftarrow c\left(e_j^{(k)}, f_i^{(k)}\right) + \delta(k, i, j)$$

$$c\left(e_j^{(k)}\right) \leftarrow c\left(e_j^{(k)}\right) + \delta(k, i, j)$$

$$c(j|i, l, m) \leftarrow c(j|i, l, m) + \delta(k, i, j)$$

$$c(i, l, m) \leftarrow c(i, l, m) + \delta(k, i, j)$$

$$where\ \delta(k, i, j) = \frac{\mathrm{t}\,(f_i^{(k)} |\, e_j^{(k)})}{\sum_{j=0}^{l_k} \mathrm{t}\,(f_i^{(k)} |\, e_j^{(k)})}$$

Set- --

$$t_{ML}(f/e) = \frac{c(e, f)}{c(e)}$$

$$q_{ML}(j/i, l, m) = \frac{c(j\,|i, l\,, m)}{c(i, l, m)}$$

Output: parameters $t(f|e)$

Figure 5: parameter estimation algorithm for IBM model 2 for the case of partially observed data

RESULTS ON TEST DATA FOR THE ENGLISH-FRENCH WORD ALIGNMENT TASK:

Data: The test set for English-French contains 447 sentences. These sentences were extracted from a set of word aligned sentences created by Prof.Hermann Ney and Franz Och.

Source and target language: For this task, English is the source language, and French is the target language. Therefore, in a word alignment line: it outputs sentence number position_L1 position_L2 [S|P] [confidence]

position_L1 represents the position of the word within the English sentence,
position_L2 represents the position of the word within the French sentence, [the first word in a sentence is at position 1]

An alignment has to be provided for each word in the test files. If a word does not appear in any alignment, we assume by default a P[Probable] NULL alignment for that word.

Output: Alignments between English sentences and French sentences is the output. We will calculate precision, Recall, F-measure for both Sure and Probable alignments for the test data set.

With IBM model 1, parameters iteration size = 0.05, and assuming uniform distribution following are the results:

Word Alignment Evaluation:

Evaluation of SURE alignments	Evaluation of PROBABLE alignments
Precision = 0.4316	Precision = 0.6559
Recall = 0.8098	Recall = 0.2850
F-measure = 0.5631	F-measure = 0.3973

AER = 0.2906

Precision quantifies the number of positive class predictions that actually belong to the positive class.
Precision = True Positives / (True Positives + False Positives)

Recall quantifies the number of positive class predictions made out of all positive examples in the dataset.
Recall = True Positives / (True Positives + False Negatives)

F-Measure provides a single score that balances both the concerns of precision and recall in one number.
F-Measure = (2 * Precision * Recall) / (Precision + Recall)

Average Error Rate (AER): test errors averaged over all splits and all datasets.

CHAPTER 5

IBM MODEL 3

5.1 IBM Alignment MODEL 3

Fertility problem is addressed in model 3. The fertility is modeled using probability distribution with parameter (phi). For each foreign word j, such distribution indicates to how many output words ∅ it usually translates. This model deals with dropping input words because it allows phi =0. But there is still an issue while adding words. For example, the English word do is often inserted when negating. This issue generates a special NULL token that can also have its fertility modeled using a conditional distribution.

For one thing, this view is simple. We can set things up so that any English sentence can be converted to any French sentence, and this will turn out to be important later. In the other views, it is not so clear how to get from here to there for any arbitrary pair of sentences. For another thing, remember that $p(f|e)$ doesn't necessarily have to turn English into good French. We saw previously that some of the slack will be taken up by the independently-trained $p(e)$ model.

5.1.1 Working of the model

Let's look at this view in more detail. We can't propose that English words are replaced by French words one for one, because then French translations would always be the same length as their English counterparts. This isn't what we observe in translation data. So, we have to allow that an English word may produce more than one French word, or perhaps no French words at all.

Here's the clear view:

1. For each word $\boldsymbol{e_i}$ in an English sentence (i = 1 ... l), we choose a fertility phi-i. The choice of fertility is dependent solely on the English word in question. It is not dependent on the other English words in the English sentence, or on their fertilities.

2. For each word $\boldsymbol{e_i}$, we generate $\emptyset_i$ French words. The choice of French word is dependent solely on the English word that generates it. It is not dependent on the English context around the English word. It is not dependent on other French words that have been generated from this or any other English word.

3. All those French words are permuted. Each French word is assigned an absolute target "position slot." For example, one word may be assigned position 3, and another word may be assigned position 2 -- the latter word would then precede the former in the final French sentence. The choice of position for a French word is dependent solely on the absolute position of the English word that generates it.

$$P(a, f \mid e) = \prod_{i=1}^{l} n(\emptyset_i \mid e_i) * \prod_{j=1}^{m} t(f_i \mid e_{a_j}) * \prod_{j=1}^{m} d(j \mid a_j, l, m)$$

5.1.2 Parameters for this model

Word translations: $t(f|e)$
Spurious words: $t(f_i|\ NULL)$
Fertilities: $n(number\ of\ words\ word\ will\ produce|e)$. for example, $n(1|house)$ means probability that "house" will produce 1 French word whenever it appears.

Distortions:
$q(position\ number\ of\ word\ in\ English\ sentence \mid position\ number\ of\ word\ in\ French\ sentence,$
$number\ of\ words\ in\ English\ sentence, number\ of\ words\ in\ French\ sentence)$.

The probability of producing a particular alignment and French string is basically the product of a bunch of smaller probabilities, such as for choosing a particular fertility for a particular word (n), choosing a particular word translation (t), moving a word from English position i to French position j (d), etc.

n, t, p, q are the model 3 parameters. Again, if we had complete data of English strings and step-by-step rewritings into French, we could:

- Compute $n(0|did)$ by locating every instance of "did", and seeing how many words it translates to
- $t(mansion|house)$ how many of all French words generated by "house" were mansion
- $q(5|2,4,6)$ out of all times some second word is in a translation, how many times did it come from the fifth word (in sentences of length 4 and 6 respectively).

5.1.3 Distortion probabilities for Spurious words

We could have $n(3|NULL)$ (probability of there being exactly 3 spurious words in a French translation). Instead of, $n(0|NULL), n(1|NULL), \ldots., n(25|NULL)$ have a single parameter p_1.

After assign fertilities to non-NULL English words we want to generate (say) z French words.
As we generate each of z words, we optionally toss in spurious French word with probability p_1.
Probability of not adding spurious word: $p_0 = 1 - p_1$

Since we don't have word- aligned data, we bootstrap alignments to incomplete data
From a sentence aligned bilingual corpus

1) Assume some startup values for n, t, p, q.
2) Use values for n, t, p, q in model 3 to work out chances of different possible alignments. Use these alignments to update values of n, t, p, q.
3) Go to step 2 until it converges

This is a more complicated case of the EM algorithm.

5.1.4 Generative Algorithm for Model 3

Algorithm:

1. For each English word e_i indexed by $i = 1, 2, \ldots, l$, choose fertility $\emptyset_i$ with probability $n(\emptyset_i | e_i)$.
2. Choose the number $\emptyset_0$ of "spurious" French words to be generated from e_0 = NULL, using probability p_1 and the sum of fertilities from step 1.
3. Let m be the sum of fertilities for all words, including NULL.
4. For each $i = 0, 1, 2, \ldots, l$, and each $k = 1, 2, \ldots, \emptyset_i$, choose a French word τ_i^k with probability $t(\tau_i^k | e_i)$.
5. For each $i = 1, 2, \ldots, l$, and each $k = 1, 2, \ldots, \emptyset_i$, choose target French position $\emptyset_i^k$ with probability $d(\emptyset_i^k | i, l, m)$.
6. For each k = 1, 2, ..., $\emptyset_i$, choose a position $\emptyset_0^k$ from the $\emptyset_i^{k+1}$ remaining vacant positions in 1, 2, ... m, for a total probability of $1/\emptyset_0!$.
7. Output the French sentence with words τ_i^k in positions
$$\emptyset_i^k \ (0 <= i <= l, 1 <= k <= \emptyset_i).$$

5.1.5 What is EM Algorithm doing here in this model 3?

The EM algorithm knows nothing about natural language translation, of course. It's simply trying to optimize some numerical deal. What is that deal? To answer that, let's go back to the quantitative evaluation of models that we discussed. In that section, we were discussing language models, and we decided that a good language model would be one that assigned a high $P(e)$ to monolingual test data. Likewise, a good translation
n model will be one that assigns a high $P(f|e)$ to bilingual test data.

Because Model 3 does not include any dependencies from one sentence to the next, we can take $P(f|e)$ to be the product of all the $P(f|e)$'s of all the sentence pairs in the bilingual corpus. We have a formula for $P(f|e)$, so this is a quantitative notion. One set of parameter values for Model 3 will be better or worse than another set, according to this measure.

Parameters set uniformly will produce a very low $P(f|e)$. As it turns out, each iteration of the EM algorithm is guaranteed to improve $P(f|e)$. That's the number that EM is optimizing in its computations. EM is not guaranteed to find a global optimum, but rather only a local optimum.

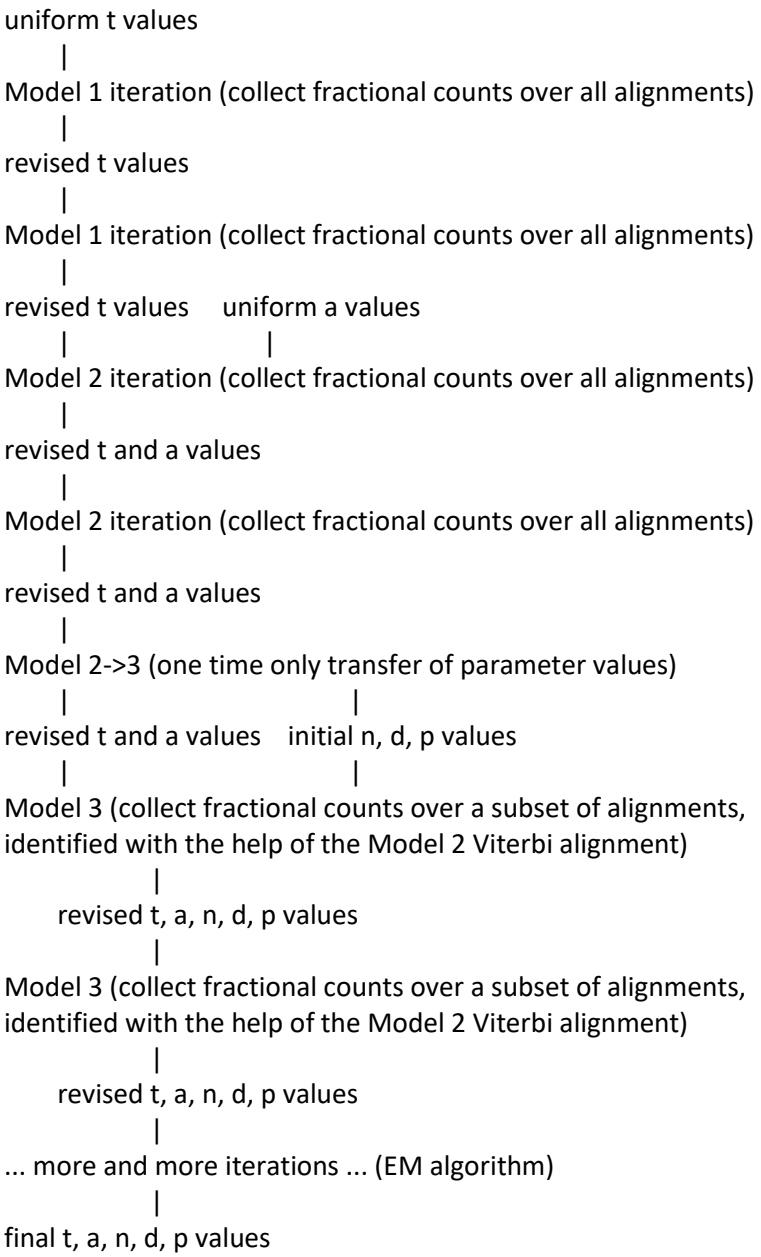
It is more convenient to measure the goodness of a model with perplexity:

$$- \frac{\log_2 p(f/e)}{N}$$

Each EM iteration lowers the perplexity.
Estimation is performed in a similar way to the previous two Models (initialization, counting, estimation). Model 3 can again be initialized using the estimations from running Model 2. At estimation time, summing over all alignments is not possible.

5.2 The final training Scheme and Decoding Algorithm for IBM Model 3

In this new scheme, we will insert Model 2 iterations as well as a special “one time only” transfer iteration between Models 2 and 3. To collect a subset of reasonable alignments for Model 3 training, we will start with the Model 2 Viterbi alignment, and use it to greedily search for the Model 3 “Viterbi” alignment (the scare quotes indicate that we are not really sure it's the best, but it's the best we can find quickly). Then we collect up a neighborhood of reasonable alignments. Here it is the practical view of proceeding to algorithm:



Notice that we continue to revise the “a” parameter values (“reverse distortion”) even during Model 3 iterations. That's because Model 3 needs to invoke Model 2 scoring as a subroutine.

REFERENCES

(1) Pattern Theory The Stochastic Analysis of Real-World Signals David Mumford Agnes Desolneux

(2) Statistical Machine Translation: The IBM Translation Models 1-3 Peter F. Brown Stephen A. Della Pietra Vincent J. Della Pietra Robert L. Mercer IBM T.J. Watson Research Center PGDOP Course, 2006

(3) A STATISTICAL APPROACH TO MACHINE TRANSLATION Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin IBM Thomas J. Watson Research Center Yorktown Heights, NY

(4) Statistical Machine Translation: IBM Models 1 and 2 Michael Collins

(5) HMM-Based Word Alignment in Statistical Translation Stephan Vogel Hermann Ney Christoph Tillmann Lehrstuhl f•ur Informatik VI, RWTH Aachen D-52056 Aachen, Germany fvogel,ney,tillmanng@informatik.rwth-aachen.de

(6) An Approach to N-Gram Language Model Evaluation in Phrase-Based Statistical Machine Translation Jinsong Su ; Qun Liu ; Huailin Dong ; Yidong Chen ; Xiaodong Shi

(7) wikipedia.org/wiki/Statistical_machine_translation

(8) Natural Language Processing: Machine translation ny Christopher Manning

(9) Computing optical Alignments for the IBM model 3 Translation Model Thomas Schoenemann Centre for Mathematical Sciences Lund University, Sweden

(10) A Statistical MT Tutorial Workbook Kevin Knight *prepared in connection with the JHU summer workshop*

Abbreviations

e = English sentence

f = French sentence

e_i = the i^{th}English word

f_j= the j^{th} French word

l = number of words in the English sentence

m = number of words in the French sentence

a = alignment (vector of integers a_1 ... a_m, where each a_j ranges from $0\ to\ l$)

a_j = the English position connected to by the j^{th} French word in alignment a

e_{a_j} = the actual English word connected to by the j^{th} French word in alignment a

$\emptyset_i$= fertility of English word i (where i ranges from 0 to l), given the alignment a

EM – Estimation Maximization

M-1 - MODEL 1

SMT – Statistical Machine Translation

ML – maximum Like-li-hood