Chapter 2

Machine Translation

In the early 1960s the philosopher Bar-Hillel published a famous attack on work in *machine translation* or *MT*. He made two main points: first, MT required a machine to understand the sentence to be translated, and second, we were so far from designing programs that could understand human language that we should put off MT into the indefinite future.

On the first point, Bar-Hillel's argument was conclusive. He simply pointed out that even a very simple example, like translating "The baby picked up a pen", is difficult because '*pen*' has two meanings (or, in NLP terminology, *word senses*): "writing instrument" and "animal/baby pen". If the target language (the language into which we are translating) does not have a word with exactly these two senses, it is necessary to *disambiguate* '*pen*'. (To "disambiguate" is to make unambiguous, in this case to decide which word sense was intended.) Word-sense disambiguation is an ongoing research issue. Furthermore, once we see one example like this it is not hard to think of many more.

Since we are still a long way from programs that understand what they read, perfect MT is still in the future. But perhaps the really difficult MT problems do not occur very often. That seems to be the case: readable MT, MT with only a few debilitating mistakes, now seems within our grasp. Certainly the progress over the last ten-fifteen years has been phenomenal. That this progress has been largely fueled by the new statistical approaches is one of the best selling points for statistical NLP. In this chapter we look at (elementary) MT from the statistical perspective. In doing so we also motivate some of the mathematical techniques used throughout this book, particularly the *expectation maximization algorithm* (*EM* for short).

The key idea behind statistical MT is quite simple. If we want to trans-

late between, say, French and English we first obtain a French-English *parallel corpus* — a text in which each sentence expressed in French is *aligned* with an English sentence meaning the same thing. The first such corpus to be used for MT was the so-called *Canadian Hansard's* — the proceedings of the Canadian parliament, which by Canadian law must be published in both English and French no matter which language was used by the speakers in parliament. (It is called *Hansard's* after the British printer who first published the proceedings of the British parliament.)

Now suppose we want to know which English word (or words) are the translations of the French word '*pain*'. (The most common translation is '*bread*'.) To make this concrete, let us further suppose that our "corpus" consisted of the following French/English pairs:

J 'ai acheté du pain	I bought some bread
J 'ai acheté du beurre	I bought some butter
Nous devons manger le pain blanc	We must eat the white bread

(In French '*j'ai*' is a single word. However when tokenizing French is it useful to split it in two, much as on page 15 we discussed splitting '*doesn't*' into '*does*' and '*n't*'.)

As we go through the corpus looking for French sentences with '*pain*' in them, we check the words in the corresponding English sentence. Here we arranged it so that the word '*bread*' is the only word common between the first and last sentences. In general things will not be so easy, but it is not hard to believe that you will find '*bread*' occurring with great regularity as a possible translation of '*pain*', and that at the very least it would become one of your top candidates.

Statistical MT follows from this simple observation.

2.1 The fundamental theorem of MT

Now let us start at the beginning and develop this idea more precisely.

Notation: Here $\boldsymbol{F}$ is a random variable denoting a French (or foreign) sentence, with $\boldsymbol{f}$ being a possible value, and $\boldsymbol{E}$ is a random variable denoting an English sentence. We use M for the length of $\boldsymbol{F}$, so $\boldsymbol{F} =< F_1, \ldots F_m >$. Similarly, L is the length of $\boldsymbol{E} =< E_1 \ldots E_l >$. We also typically use j to index over English sentences and k over French.

1. We must eat the white bread
2. We must eat the bread white
3. We eat must the bread white.

Figure 2.1: Some possible translations of "Nous devons manger le pain blanc"

From a probabilistic point of view, MT can be formalized as finding the most probable translation $\boldsymbol{e}$ of a foreign language string $\boldsymbol{f}$, which is

$$\arg\max_{\boldsymbol{e}} \mathrm{P}(\boldsymbol{e} \mid \boldsymbol{f}).$$

As noted in 1.6, the *noisy-channel model* is often a good way to approach "argmax"-type problems, and we do this here:

$$\arg\max_{\boldsymbol{e}} \mathrm{P}(\boldsymbol{e} \mid \boldsymbol{f}) = \arg\max_{\boldsymbol{e}} \mathrm{P}(\boldsymbol{e})\mathrm{P}(\boldsymbol{f} \mid \boldsymbol{e}). \tag{2.1}$$

This equation is often called the *fundamental theorem of machine translation*. The first term on the right is a *language model*, as discussed in Chapter 1. The second term is the *translation model*. It encodes the procedure for turning English strings into French ones.

At first glance, this second term looks counterintuitive. On the left we have the term $P(\boldsymbol{e} \mid \boldsymbol{f})$ and we turn this into a problem requiring that we estimate $P(\boldsymbol{f} \mid \boldsymbol{e})$. Given that it is just as hard to translate from English into French as the other way around, it is not obvious that the noisy-channel model has gained us much.

Nevertheless, this factorization is useful because the translation and language models capture different kinds of dependencies, and (2.1) tells us how these should be combined. To see this, let us consider the third sentence in our fake "corpus" in the introduction: "Nous devons manger le pain blanc". Consider the several possible English translations in Figure 2.1. The first is the correct translation, the second is the word-by-word translation, and the last permutes the second two words instead of the last two.

In our current state of knowledge, our translation models are very poor at ordering the words in the translation and at picking the best words for a particular context. Thus it would not be surprising if the translation model picked the incorrect literal translation in Figure 2.1 as the best and the other two as equally likely variants. On the other hand, the overlapping windows

of even a simple trigram language model should have no problem assigning the first a comparatively high probability and the others dramatically lower ones. Thus by multiplying the two probabilities, our program has a much better chance of getting the correct result.

A second important advantage of the noisy-channel formulation is that, while the translation model P($\boldsymbol{F}|\boldsymbol{E}$) needs to be trained from parallel data (which always is in short supply), the language model P($\boldsymbol{E}$) can be trained from monolingual data, which is plentiful. Thus it permits us to train our translation system from a wider range of data, and simply adding more training data usually results in more accurate translations (all other factors equal).

In the last chapter we covered language modeling. Here we start with the translation model.

2.2 The IBM Model 1 noisy-channel model

We now turn to a very simple model for P($\boldsymbol{F}|\boldsymbol{E}$) known as IBM model 1, so called because it was the first of five ever more complex MT models defined by a group at IBM in the early 1990s.

This model makes a number of simplifying assumptions that more complex models remove. These assumptions mean that model 1 is not a particularly accurate channel model, but it is very simple and easy to train. (We show how to relax one of these assumptions when we explain IBM model 2 in 2.3.)

IBM model 1 assumes that each French word f_k is the translation of exactly one English word in $\boldsymbol{e}$, say e_j. That is, we assume f_k is independent of all the other words in $\boldsymbol{e}$ given the word e_j.

This assumption is less restrictive than it may seem at first. We don't insist that $k = j$, so the French words don't need to be in the same order as the English words they correspond to. We also don't require a one-to-one mapping between English and French words, so each English word can correspond to zero, one, or several French words. We can also give French words some of the same flexibility by assuming that each English sentence $\boldsymbol{e}$ contains an additional invisible *null word* (also called a *spurious word*) '$*$N$*$' that generates words in $\boldsymbol{f}$ that aren't translations of any actual word in $\boldsymbol{e}$. (The $*$N$*$ is assumed to be e_0, the "zeroth" word of the English sentence.)

None of the IBM models give up the "one English word" assumption, but more recent work does. This work, to be discussed at the end of this chapter, assumes that multiword phrases, rather than individual words, are

the basic atomic units of translation.

We formalize this word-to-word translation idea in terms of word alignments. A *word alignment* $\boldsymbol{a}$ from a French sentence $\boldsymbol{f}$ of length m to an English sentence $\boldsymbol{e}$ of length l is a vector of length m where e_{a_k} is the English word that translates to f_k. That is, a_k is a position in $\boldsymbol{e}$, i.e., an integer between 0 and l.

Example 2.1: In our toy corpus in the introduction,

J 'ai acheté du pain	I bought some bread
J 'ai acheté du beurre	I bought some butter
Nous devons manger le pain blanc	We must eat the white bread

the alignment for the first sentence pair is

$< 1, 0, 2, 3, 4 >$

Note how the second French word aligns with the zeroth English word, the spurious word. The '*ai*' is needed to express past tense in French, and there is no word corresponding to it in the English. Otherwise the translation is word-for-word.

The correct alignment for the third sentence is:

$< 1, 2, 3, 4, 6, 5 >$

Again the alignment is close to word-for-word, except that it switches the order of between '*pain blanc*' and '*white bread*'.

To give another example, suppose the person who translated this into French had deleted '*devons*', the translation of '*must*'. Then we would have an alignment

$< 1, 3, 4, 6, 5 >$

Notice the alignment vector is now of length five rather than six, and none of the French words align to '*must*'.

We introduce alignments into our probability equations using marginalization. That is:

$$\mathrm{P}(\boldsymbol{f} \mid \boldsymbol{e}) \;=\; \sum_{\boldsymbol{a}} \mathrm{P}(\boldsymbol{f}, \boldsymbol{a} \mid \boldsymbol{e}) \tag{2.2}$$

As explained in Chapter 1, if you have joint probability $\mathrm{P}(C, D)$ you can sum over all possible values of D to get the probability of C. Here, we obtain the probability $\mathrm{P}(\boldsymbol{f}|\boldsymbol{e})$ by marginalizing over A. We then separate $\boldsymbol{f}$ and $\boldsymbol{a}$ using the chain rule:

$$\mathrm{P}(\boldsymbol{f} \mid \boldsymbol{e}) \;=\; \sum_{\boldsymbol{a}} \mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e}) \mathrm{P}(\boldsymbol{f} \mid \boldsymbol{a}, \boldsymbol{e}) \tag{2.3}$$

It is useful to introduce a bit of terminology. Our parallel corpus gives us just the English and French words, E, F. We say that these random variables are *visible variables*, since we see their values in our data. On the other hand, the alignment random variable $\boldsymbol{A}$ is not visible: it is a *hidden variable* because our parallel corpus is aligned only with respect to sentences, not words, so we do not know the correct value of $\boldsymbol{A}$. Hidden variables are useful when they give us a useful way to think about our data. We see in ensuing sections how $\boldsymbol{A}$ does this.

Example 2.2: Some examples of $\mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e})$ are in order. So suppose we are aligning an $\boldsymbol{e}/\boldsymbol{f}$ pair in which both are of length six. English and French have very similar word order so, all else being equal, the most probable alignment might be $< 1, 2, 3, 4, 5, 6 >$. In our "the white bread" example we came close to this except the order of the last two words was exchanged, $< 1, 2, 3, 4, 6, 5 >$. This should have a probability lower than 1 to 6 in order, but still relatively high.

At the other extreme, it is easy to make up alignments that should have very low probability when we are aligning two six-word sentences. Consider $< 6, 5, 4, 3, 2, 1 >$ — the French sentence is ordered in the reverse of the English. An even sillier alignment would be $< 1, 1, 1, 1, 1, 1 >$. This says that the entire French translation is based upon the first word of the English, and the rest of the English was completely ignored. There is nothing in the definition of an alignment function that forbids this.

Even though many silly alignments should be assigned *very* low probabilities, IBM model 1 assumes that all the ℓ^m possible word alignments of $\boldsymbol{e}$ and $\boldsymbol{f}$ are equally likely! Thus removing this assumption is one of the major improvements of more sophisticated noisy-channel models for machine translation. In particular, this is done in *IBM model 2* discussed in section 2.3.

Nevertheless, IBM model 1 does remarkably well, so let's see how the equal-probability assumption plays out. We now plug the IBM model 1 assumption into (2.3) and get our "model 1 equation".

First. note that once we condition on a, we can break apart the probability of $\boldsymbol{f}$ into the probabilities for each word in f, f_k. That is, we can now say

$$\mathrm{P}(\boldsymbol{f} \mid \boldsymbol{e}, \boldsymbol{a}) \;=\; \prod_{k=1}^{m} \mathrm{P}(f_k \mid e_{a(k)}).$$

Substituting this into equation 2.3 gives us:

$$\begin{aligned} \mathrm{P}(\boldsymbol{f} \mid \boldsymbol{e}) &= \sum_{\boldsymbol{a}} \mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e}) \prod_{k=1}^{m} \mathrm{P}(f_k \mid e_{a(k)}) \\ &= \mathrm{P}(m \mid \ell)\ell^{-m} \sum_{\boldsymbol{a}} \prod_{k=1}^{m} \mathrm{P}(f_k \mid e_{a(k)}). \end{aligned} \tag{2.4}$$

In (2.4) we replaced $\mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e})$ (the probability for alignment a) by $\mathrm{P}(m \mid \ell)\ell^{-m}$. First. note that there are ℓ^m possible alignments, and they all have the same probability. This probability is ℓ^{-m}. However, initially we are only "given" (condition on) $\boldsymbol{e}$ from which we can get l, its length. Thus we must "guess" (assign a probability to) m, the length of $\boldsymbol{f}$. Hence the term $\mathrm{P}(m \mid l)$.

Example 2.3: Suppose $\boldsymbol{f}$ = '*Pas fumer*' and $\boldsymbol{e}$ = '*No smoking*'. Here $l = m = 2$, so the probability of any one alignment is $\frac{1}{4}\mathrm{P}(M = 2 \mid L = 2)$. Next consider the length term. A plausible value for $\mathrm{P}(2 \mid 2)$ would be 0.4. That is, a substantial fraction of French two-word sentences get translated into two-word English ones. So the product of the terms outside the summation in 2.4 would be 0.1.

Equation 2.4 describes a generative model in that, given an English sentence $\boldsymbol{e}$, it tells us how we can generate a French sentence $\boldsymbol{f}$ along with its associated probability. As you may remember, in Chapter 1 we talked about a generative "story" as a good way to explain equations that describe generative models. Here is the associated generative story .

Given a sentence $\boldsymbol{e}$ of length ℓ, we can generate a French sentence $\boldsymbol{f}$ of length m using the following steps and their associated probabilities.

1. Choose the length m of the French sentence $\boldsymbol{f}$. This gives us the $\mathrm{P}(m \mid l)$ in Equation 2.4.

2. Choose a word alignment $\boldsymbol{a}$ at random from the ℓ^m possible alignments. This give us the ℓ^{-m} portion.

3. Generate each French word $F_k, k = 1, \ldots, m$ from the English word e_{a_k} it is aligned to with probability $\mathrm{P}(f_k \mid e_{a(k)})$. (This gives us $\prod_{k=1}^{m} \mathrm{P}(f_k \mid e_{a(k)})$.)

This gives one way to generate $\boldsymbol{f}$ from $\boldsymbol{e}$ — the way corresponding to the chosen alignment $\boldsymbol{a}$. We can generate $\boldsymbol{f}$ in lots of ways, each one corresponding to a different alignment. In Equation 2.4 we sum over all of these to get the total probability of $\mathrm{P}(\boldsymbol{f}|\boldsymbol{e})$.

From this story we see immediately that the IBM model 1 channel model has two sets of parameters. One set is the estimate of the conditional probability $\mathrm{P}(m|\ell)$ of generating a French sentence of length m from an English sentence of length ℓ. The most straightforward way to get this is to use the maximum likelihood estimator. First go through the parallel corpus and count how often an English sentence of length ℓ is paired with a French sentence of length m. In our usual way, we call this $n_{\ell,m}$. Then to get the maximum likelihood estimation we divide this by how often an English length ℓ appears in the corpus, $n_{\ell,\circ}$. We call these parameters $\eta_{\ell,m} = n_{\ell,m}/n_{\ell,\circ}$.

The second set of parameters is the conditional probabilities $\mathrm{P}(f|e)$ of generating the French word f given that it is aligned to the English word e, henceforth referred to as $\tau_{e,f}$.

After substituting the model parameter estimations into our IBM model 1 equation, we get this:

$$\mathrm{P}(\boldsymbol{f} \mid \boldsymbol{e}) \quad = \quad \eta_{\ell,m}\ell^{-m} \sum_{\boldsymbol{a}} \prod_{k=1}^{m} \tau_{e_{a(k)},f_k}. \tag{2.5}$$

All the terms of this equation are either known or easily estimated except for the τ's. We turn to this next.

Example 2.4: If $\boldsymbol{f}$ is '*J'ai acheté du pain*' then $\mathrm{P}(\boldsymbol{f} \mid \boldsymbol{e})$ will be the same if $\boldsymbol{e}$ is '*I bought some bread*' or '*Bought I some bread*'. Basically, this is a consequence of the assumption that all alignments are equally likely. More immediately, we can see this from Equation 2.5. Both English versions have $\ell = 4$ so $\eta_{\ell,m}\ell^{-m}$ is the same for both translations. Then in the first case $\boldsymbol{a}_1 =< 1, 0, 2, 3, 4 >$ while $\boldsymbol{a}_2 =< 2, 0, 1, 3, 4 >$. With these alignments the product of $\tau_{e_{a(k)},f_k}$ will be the same, differing only in the order of multiplication.

On the other hand, consider the two possible translations '*I bought some bread*' and '*I bought bread*'. (In most cases the second of these would be the preferred translation because the '*du*' in the French version is grammatically obligatory. It does not have any meaning of its own and thus need not be translated.) Now the translation model probabilities will differ. First, for comparable sentences French tends to use more words than English by about 10%, so while $\eta_{4,4}$ will probably be larger than $\eta_{3,4}$ the difference will be small. The next term, however, ℓ^{-m}, will be larger for the shorter English sentence (4^{-4} vs. 3^{-4}). The difference in word probabilities will be due to the difference in alignments of '*du*' in the French to '*some*' in the first English version and $*\mathrm{N}*$ in the second. Thus the first has the term $\tau_{some,du}$ and the second $\tau_{*\mathrm{N}*,du}$. Both these translation pairs are reasonably likely, but in our model $\tau_{*\mathrm{N}*,du}$ will typically be smaller because the null word has quite a few translations but '*some*' has fewer, and thus each one will be more probable. Thus we would typically expect the translation model to slightly prefer the more wordy English. On the other hand, a good English language model would

J 'ai acheté du pain	$< 1, 0, 2, 3, 4 >$	I bought some bread
J 'ai acheté du beurre	$< 1, 0, 2, 3, 4 >$	I bought some butter
Nous devons manger le pain blanc	$< 1, 2, 3, 4, 6, 5 >$	We must eat the white bread

Figure 2.2: A word-aligned version of our toy corpus

distinctly prefer the shorter English, and this would most likely tip the balance in favor of '*I bought bread*'.

2.2.1 Estimating IBM model 1 parameters with EM

We attack the τ problem by noting that it would be easy to estimate $\boldsymbol{\tau}$ from a *word-aligned parallel corpus*, i.e., a corpus that specifies the word alignment $\boldsymbol{a}$ for the English-French sentences. Figure 2.2 shows such an alignment for our toy corpus.

Note how the alignment allows us to read off how often each French word is paired with each English, and from that it is easy to get a maximum likelihood estimate of all the $\tau_{e,f}$. The maximum likelihood estimator for $\tau_{e,f}$ is just the number of times e aligns to f divided by the number of times e aligns with anything. Or, more formally:

$$n_{e,f}(\boldsymbol{a}) \quad = \quad \sum_{k:f_k=f} [\![e_{a_k} = e]\!] \,. \tag{2.6}$$

where $[\![condition]\!]$ is the *indicator function* for *condition*, i.e., it is 1 if *condition* is true and 0 otherwise. Then the maximum likelihood estimator $\boldsymbol{\tau}$ is just the relative frequency, i.e.:

$$\hat{\tau}_{e,f} \quad = \quad \frac{n_{e,f}(\boldsymbol{a})}{n_{e,\circ}(\boldsymbol{a})} \tag{2.7}$$

where $n_{e,\circ} = \sum_f n_{e,f}(\boldsymbol{a})$ is the number of times that e is aligned to any French word in $\boldsymbol{a}$ (this is *not* necessarily the same as the number of times e appears in the training corpus). These two equations combine to say, in effect: go through the corpus counting how often e aligns with f and then take the maximum likelihood estimate to get the corresponding probability.

Example 2.5: In Figure 2.2 all the τ's are 1, because each English word is aligned to a unique French word. Suppose that we made a mistake and aligned '*bread*' with '*must*' in the third sentence. Then $\tau_{\text{'}bread\text{'},\text{'}pain\text{'}}$ would be $\frac{1}{2}$ because bread is aligned with pain once, and bread is paired with anything two times.

English	French	Sentences			$\tau^2_{e,f}$
		1	2	3	
bought	pain	1/2			1/4
bought	acheté	1/2	1/2		1/2
bread	pain	1/2		1/2	1/2
bread	acheté	1/2			1/4
bought	beurre		1/2		1/4
butter	acheté		1/2		1/2
butter	beurre		1/2		1/2
eat	pain			1/2	1/2
eat	manger			1/2	1/2
bread	manger			1/2	1/4

Figure 2.3: Partial counts for English-French word pairs in three sentences

Now, let us suppose that the folks who created our word-aligned corpus were not always confident that they could align the words correctly and that when they were unsure they labeled alignments with probabilities: e.g., they think e_2 aligns with f_2, but it might be e_1, so they assign a probability of 0.9 to the first and 0.1 to the second. What do we do when the alignments are not yes or no, but more or less confident?

There are several possibilities. One is simply to ignore any alignment with confidence less than some threshold, say 0.8. If we had more word-aligned data than we could use, this would be a reasonable thing to do. However, this is never the case, so if there is useful information in our data we should try to extract it. And there is a lot of useful information in alignments even when they are quite uncertain. Suppose the aligner knows that in some ten-word sentence f_1 aligns with either e_1 or e_2, but is completely unsure which, so they both get probability 0.5. Note that without this information we were much more uncertain. In IBM model 1, since all alignments are equally likely, those two alignments would have had probability 0.1 each (since there were ten words in the sentence). So even a 50-50 split is useful.

We are now poised to introduce the first of two key insights of the so-called *expectation maximization algorithm* (EM). In a case like the above we split the alignment counts $N_{e,f}$ according to their probabilities. In the 50-50 split case both n_{e_1,f_1} and n_{e_2,f_1} get a 0.5 "count". When we split a count like this it is a *fractional count* or *partial count*.

Example 2.6: Suppose we have some annotators align our three-sentence par-

allel corpus. For some reason they align '*acheté*' with equal probability to both '*bought*' and '*bread*' in the first sentence. Similarly, '*pain*' is also aligned with equal probability to these two. The same sort of confusion occurs in the second sentence, except now it is '*acheté/beurre*' and '*bought/butter*'. In the third sentences it is '*manger/pain*' and '*eat/bread*'. Figure 2.3 shows the partial counts we get for these word pairs. For example, the first line starting with '*bought*' indicates that it is aligned with '*pain*' 0.5 times in sentence one, and not at all in the other two sentences. Looking down the columns, we see that each sentence is responsible for a total of two counts. (Because we are assuming all the other words are unambiguous we are ignoring them. In effect assuming that the sentences only include the ambiguous words.)

The last column gives the second iteration $\tau^2_{e,f}$ that is computed from these partial counts. (We occasionally use superscripts on parameter values to indicate the iteration in which they apply. In the first iteration all of the τs were equal and the new τ^2's are used in the second iteration.) For example,

$$\begin{aligned}\tau^2_{\text{'bought','pain'}} &= \frac{n_{\text{'bought','pain'}}}{n_{\text{'bought',}\circ}} \\ &= \frac{1/2}{1/2+1/2+1/2+1/2} = 1/4.\end{aligned}$$

Again, note that by this point EM is preferring '*bought/acheté*' over any of the other translations of '*bought*', and similarly for '*bread*' and '*pain*'. On the other hand, the translations for '*butter*' and '*eat*' have not been clarified. This is because each of these words appears only in a single sentence, so there is no way to get disambiguation from other sentences.

In the EM algorithm we take this to an extreme and pretend that initially our annotaters gave each ***f*-*e*** alignment an equal probability. So for a ten-word English sentence each fractional count would be 0.1. (Actually, it need not necessarily be uniform, but when you don't know anything, uniform is most often the best thing to try.) Then after going through the corpus summing up the fractional counts, we set the τ parameters to the maximum likelihood estimates from the counts, just as if they were "real".

Now we introduce the second key idea. The parameters we have just obtained should be much better than our original assumption that everything was equally likely, so we repeat the process, but now using the new probabilities. And so on. That is to say, EM is an iterative algorithm in which we start with a very bad (e.g., uniform) distribution of our τs and at each iteration replace them with the ML estimate from the previous iteration. In Section 2.2.3, after seeing that EM actually works, we give an analysis of what EM is doing from a mathematical point of view.

To make this complete we just need the equation for computing fractional counts when we have just probabilistic information about who is aligned with

English	French	Sentences			$\tau^3_{e,f}$
		1	2	3	
bought	pain	1/3			2/11
bought	acheté	2/3	1/2		7/11
bread	pain	2/3		1/2	7/11
bread	acheté	1/3			2/11
bought	beurre		1/3		2/11
butter	acheté		1/2		3/7
butter	beurre		2/3		4/7
eat	pain			1/2	3/7
eat	manger			2/3	4/7
bread	manger			1/3	2/11

Figure 2.4: Partial counts for English-French word pairs on the second iteration

whom. For the moment we give it without mathematical justification:

$$n_{e_j,f_k} += \frac{\tau_{e_j,f_k}}{p_k} \tag{2.8}$$

where

$$p_k = \sum_j \tau_{e_j,f_k} \tag{2.9}$$

That is, within each sentence and for each French word f_k, we add to our running total of n_{e_j,f_k} the term on the right-hand side of Equation 2.8. (We implicitly assume that j runs over only the English words in the sentence aligned with $\boldsymbol{f}$.) This term is our estimate for the probability that f_k is the translation of our English word e_j divided by the total probability that it translates any of the English words in the corresponding sentence (p_k).

Actually, what we are computing here is the expected number of times our generative model aligns f_k with e_j given our data, that is,

$$E[n_{e,f} \mid \boldsymbol{e}, \boldsymbol{f}].$$

We come back to this point in Section 2.2.3 where we derive Equation 2.8 from first principles.

Example 2.7: The previous example (see Figure 2.3) followed the EM algorithm though its first iteration, culminating in a new $\boldsymbol{\tau}$. Here we go though the second iteration (see Figure 2.4). As before, we go though the sentences computing $n_{e,f}$

1. Pick positive initial values for $\tau_{e,f}$ for all English words e and all French words f (equal is best).

2. For $i = 1, 2, \ldots$ until convergence (see below) do:

   (a) *E-step:*
   Set $n_{e,f} = 0$ for all English words e and French words f.
   For each E/F sentence pair and for each French word position $k = 1, \ldots, m$ do:

      i. Set $p_k = \sum_{j=0}^{l} \tau_{e_j,f_k}$, where j are the positions of the English words in the same sentence pair as f_k.

      ii. For each $0 \leq j \leq l$, increment $n_{e_j,f_k} += \tau_{e_j,f_k}/p_k$

   ($n_{e,f}$ now contains the expected number of times e aligns with f)

   (b) *M-step:*
   Set $\tau_{e,f} = n_{e,f}/n_{e,\circ}$, where $n_{e,\circ} = \sum_f n_{e,f}$.

Figure 2.5: EM algorithm for IBM model 1 inference

for each word pair. Again, to take a single example, consider $N_{bought,acheté}$ for the first sentence.

$$\begin{aligned} p_{\text{`acheté'}} &= \sum_j \tau_{\text{`}j\text{'},\text{`acheté'}} \\ &= \tau_{\text{`bread'},\text{`acheté'}} + \tau_{\text{`bought'},\text{`acheté'}} \\ &= 1/4 + 1/2 \\ &= 3/4 \end{aligned}$$

$$\begin{aligned} n_{\text{`bought'},\text{`achete'}} &= \frac{\tau_{\text{`bought'},\text{`acheté'}}}{p_{\text{`acheté'}}} \\ &= \frac{1/2}{3/4} \\ &= 2/3 \end{aligned}$$

Thus we arrive at the EM algorithm for estimating the τ parameters of IBM model 1 shown in Figure 2.5 A few points about this algorithm. First, we can now understand why this is called the *Expectation Maximization* algorithm. Each iteration has two steps. First in the *e-step* we use the previous estimate $\boldsymbol{\tau}^{(i-1)}$ of the parameters to compute the *expected value* of the statistics $n_{e,f}$. Then in the *m-step* we set $\boldsymbol{\tau}^{(i)}$ to the maximum likelihood estimate using those expected values.

Second, the algorithm says to iterate until "convergence." The EM algorithm finds values of the hidden parameters that correspond to a local maximum of the likelihood.

$$L_{\boldsymbol{d}}(\boldsymbol{\Phi}) \quad \propto \quad \prod_{k=1}^{m} p_k. \tag{2.10}$$

That is the likelihood of the data is a constant times the product of the p_ks where the constant is dependent only on the English strings. After a few iterations neither the likelihood nor the $\boldsymbol{\tau}$s change much from one iteration to the next. Since we need to compute the p_k's in the inner loop anyway, keeping track of the likelihood at each iteration is a good way to measure this "convergence". We set a threshold, say 1%, and when the likelihood changes less than this threshold from one iteration to the next we stop. (As you might expect, multiplying all of these probabilities together produces a *very* small number. Thus it is better to compute the log of the likelihood by adding the $\log p_k$'s.)

Also, while in general EM is guaranteed only to find a local maximum, in our present application the likelihood of Equation 2.10 is indxunimodal — it only has one maximum. Therefore in this case we find a global maximum.

Example 2.8: Figures 2.3 and 2.4 followed EM through two iterations on a simple example, where all the alignments were certain except those for '*pain*', '*acheté*', '*buerre*',and '*manger*'. Let us compute the likelihood of our data with the τ we computed after the first and second iterations to check that the likelihood of our training corpus is indeed increasing. Equation 2.10 tells us to take the product of the p_k for the position k of every French word in the corpus. First note that $p_k = 1$ for all French words k that we assumed were correctly and uniquely aligned, e.g. '*j*', '*ai*' '*le*', etc. So these can be ignored. All we need to do is look at the p_ks for all occurrences of the four French words that were originally labeled ambiguously. However, rather than do them all, let's just consider the $p_{\text{`}acheté\text{'}}$ in the first sentence. First for the $\boldsymbol{\tau}$ after the first iteration:

$$\begin{aligned} p_{\text{`}acheté\text{'}} &= \tau_{\text{`}acheté\text{'},\text{`}bought\text{'}} + \tau_{\text{`}acheté\text{'},\text{`}bread\text{'}} \\ &= 1/2 + 1/4 \\ &= 3/4. \end{aligned}$$

For the $\boldsymbol{\tau}$ we get after the second iteration

$$\begin{aligned} p_{\text{`}acheté\text{'}} &= 2/3 + 1/3 \\ &= 1. \end{aligned}$$

So this p_k increases. As it turns out, they all either increase or remain the same.

English word	Iteration 1	Iteration 2	Iteration 19	Iteration 20
bread	0.042	0.138	0.3712	0.3710
drudgery	0.048	0.055	0.0	0.0
enslaved	0.048	0.055	0.0	0.0
loaf	0.038	0.100	0.17561	0.17571
spirit	0.001	0.0	0.0	0.0
mouths	0.017	0.055	0.13292	0.13298

Figure 2.6: Probabilities for English words translating as '*pain*'

2.2.2 An extended example

Figure 2.6 shows the estimated IBM model 1 parameters $\tau_{e,pain}$ for several English words e, trained on some sections of the Canadian Hansard's. We show these values after several different iterations of EM.

In the first iteration there is good news and bad news. The good news is that $\tau_{bread,pain}$ comes out with a relatively high value. For example, '*bread*' at 0.04 is considerably higher than '*spirit*' at 0.001. However, the program does not do too well distinguishing '*bread*' from related words that appear along with bread, such as '*baked*', not to mention some seemingly random words like '*drudgery*', which at 0.048 seems way too high. And, of course, the probability for '*bread*' in Figure 2.6 is much too low. The actual probability of translating '*bread*' as '*pain*' is close to 1, say 0.9.

We now move on to a second iteration. Now the probability of '*pain*' given '*bread*' has gone up by better than a factor of three (from 0.042 to 0.138). Spirit fell off the map, but '*drudgery*' and '*enslaved*' went up, though only slightly.

Having done this twice, there is nothing stopping us from repeating this process ad infinitum, except after a while there is not much change from one iteration to the next. After twenty iterations of EM '*pain*' is the translation of '*bread*' 37% of the time. This is very good compared to the first iteration, but it is still too low. '*Drudgery*' and '*enslaved*' have properly gone to zero. '*Loaf*' can be translated as '*pain*', but the probability of 15% is probably too high. This is because '*loaf of bread*' would usually be translated as just '*pain.*' '*Mouths*' is definitely an artifact of some funny sentences somewhere.

We added an extra level of precision to the numbers for iterations 19 and 20 to show that the changes from one iteration are becoming quite small.

2.2.3 The mathematics of IBM 1 EM

The EM algorithm follows from a theorem of statistics which goes roughly as follows. Suppose you have a generative model that depends on hidden parameters (in our case, the τs). Initially make nonzero guesses about the values of these parameters. Imagine actually generating all of the data according to these parameters. In so doing, each of the τs will be used an estimated number of times E_τ. After going through our data we reset the τ's to their maximum likelihood value according to these estimates. This can be iterated. The theorem states that as you iterate the τs will monotonically approach a set that gives a local maximum for the likelihood of the data. (It is also possible to get stuck in a saddle-point, but we ignore that here.)

A quick comparison of the above description with the EM algorithm as shown on Page 49 should convince you that it is correct, *provided that what we there labeled the e-step indeed computes expectation for the τ's.* That is what we prove in this section.

We first start with a clear formulation of what the expectation is that we are computing at each iteration i:

$$n_{e,f}^{(i)} \;=\; E_{\boldsymbol{\tau}^{(i-1)}}[n_{e,f} \mid \boldsymbol{e}, \boldsymbol{f}] \;=\; \sum_{\boldsymbol{a}} n_{e,f}(\boldsymbol{a})\, \mathrm{P}_{\boldsymbol{\tau}^{(i-1)}}(\boldsymbol{a} \mid \boldsymbol{e}, \boldsymbol{f}). \quad (2.11)$$

The first equality makes it clear that at iteration i we compute the expectation according to the $\boldsymbol{\tau}$ we obtained at the previous iteration. (For the first iteration we just use the initial, all equal τs.) The second equality comes from the definition of expectation. Here $n_{e,f}(\boldsymbol{a})$ is the number of times f is aligned with e in the alignment $\boldsymbol{a}$ defined in Equation 2.6 and repeated here:

$$n_{e,f}(\boldsymbol{a}) \;=\; \sum_{k:f_k=f} [\![e_{a_k} = e]\!] \,.$$

So the right hand side says what we would expect (excuse the pun), that the expectation is the number of times we align e and f in an alignment times the probability of that alignment.

Unfortunately, the computation is intractable as expressed above because it sums over all alignments. This number grows exponentially in the length of $\boldsymbol{a}$. We now show how the non-exponential computation we gave in our EM algorithm is equivalent.

We do this in two steps. First we show how the expectation can be reformulated as follows:

$$E_{\boldsymbol{\tau}}[n_{e,f} \mid \boldsymbol{e}, \boldsymbol{f}] \;=\; \sum_{k:f_k=f} \sum_{j:e_j=e} \mathrm{P}_{\boldsymbol{\tau}}(A_k = j \mid \boldsymbol{e}, \boldsymbol{f}). \quad (2.12)$$

Note that in this equation there is no iteration over all $\boldsymbol{a}$. Instead, it tells us to march through each French word position k computing the sum on the right. This is what we do when the EM algorithm says "For each French word position k ...". Also, note that it is intuitively quite reasonable: it says that the expected number of times the English word type e is aligned with French word type f is the sum of the probabilities of any English token e_j of type e being aligned with any French token f_k of type f.

Then we show that for IBM model 1:

$$\mathrm{P}_{\boldsymbol{\tau}}(A_k = j \mid \boldsymbol{e}, \boldsymbol{f}) \;=\; \frac{\tau_{e_j, f_k}}{\sum_{j'=0}^{l} \tau_{e_{j'}, f_k}}. \tag{2.13}$$

The j' in the denominator ranges over the word positions in the corresponding English sentence.

In many applications EM is quite sensitive to its initial parameter values. But, as remarked above, the likelihood function (2.10) for IBM model 1 is unimodal, which means that in this particular case it is not that important how $\boldsymbol{\tau}$ is initialized. You can initialize it with random positive values, but setting them all equal is typically best.

So now we turn our attention to the two equations (2.12) and (2.13) used in the derivation of the EM algorithm above. Combining (2.6) and (2.11) and reordering the summation, we have:

$$\begin{aligned} E[n_{e,f}|\boldsymbol{e}, \boldsymbol{f}] &= \sum_{\boldsymbol{a}} \sum_{k:f_k=f} [\![e_{a_k} = e]\!]\ \mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e}, \boldsymbol{f}) \\ &= \sum_{k:f_k=f} \sum_{\boldsymbol{a}} [\![e_{a_k} = e]\!]\ \mathrm{P}(a_k \mid \boldsymbol{e}, \boldsymbol{f})\ \mathrm{P}(\boldsymbol{a}_{-k} \mid \boldsymbol{e}, \boldsymbol{f}, a_k). \end{aligned}$$

where $\boldsymbol{a}_{-k}$ is the vector of word alignments for all words *except* f_k (i.e., $\boldsymbol{a}_{-k}$ is $\boldsymbol{a}$ with a_k removed). Splitting the sum over $\boldsymbol{a}$ into a sum over a_k and $\boldsymbol{a}_{-k}$ and rearranging the terms, we have:

$$\begin{aligned} E[n_{e,f}|\boldsymbol{e}, \boldsymbol{f}] &= \sum_{k:f_k=f} \sum_{a_k} \sum_{\boldsymbol{a}_{-k}} [\![e_{a_k} = e]\!]\ \mathrm{P}(a_k \mid \boldsymbol{e}, \boldsymbol{f})\ \mathrm{P}(\boldsymbol{a}_{-k} \mid \boldsymbol{e}, \boldsymbol{f}, a_k) \\ &= \sum_{k:f_k=f} \sum_{a_k} \Big([\![e_{a_k} = e]\!]\ \mathrm{P}(a_k \mid \boldsymbol{e}, \boldsymbol{f}) \sum_{\boldsymbol{a}_{-k}} \mathrm{P}(\boldsymbol{a}_{-k} \mid \boldsymbol{e}, \boldsymbol{f}, a_k) \Big) \\ &= \sum_{k:f_k=f} \sum_{a_k} [\![e_{a_k} = e]\!]\ \mathrm{P}(a_k \mid \boldsymbol{e}, \boldsymbol{f}) \\ &= \sum_{k:f_k=f} \sum_{j:e_j=e} \mathrm{P}(A_k{=}j \mid \boldsymbol{e}, \boldsymbol{f}). \end{aligned}$$

We get the third line here because $\sum_{\boldsymbol{a}_{-k}} \mathrm{P}(\boldsymbol{a}_{-k} \mid \boldsymbol{e}, \boldsymbol{f}, a_k) = 1$. All the terms we are conditioning on are constants as far as the summation is concerned and the sum of any probability distribution over all possible outcomes is 1. Next, notice that nothing in this version depends on the entire $\boldsymbol{a}$, just a_k. From this the final line follows, and this is equation 2.12.

When we turn to IBM model 2 we reuse this result, so we need to point out several aspects of the equation that will become important. First, the words that we are translating appear only in the summation. What we actually compute is the probability of the alignment. The words tell us then which $n_{e,f}$ bin gets incremented.

Second, we note that only one IBM model 1 assumption was used to get here: that each French word is generated by zero or one English words. It is this assumption that allows us to characterize all the important hidden information in the alignment vector random variable $\boldsymbol{A}$. Or, to put it another way, alignments would not make much sense if several English words were allowed to combine to form a French one. On the other hand, we did not use the second major assumption, that all alignments have equal probability. Thus Equation 2.12's reuse is allowed when the latter assumption is relaxed.

Turning to Equation 2.13, we first use the postulate that IBM model 1 alignments of French word f_k do not depend on the alignments of any other French words, so:

$$\mathrm{P}(A_k{=}j \mid \boldsymbol{e}, \boldsymbol{f}) \;=\; \mathrm{P}(A_k{=}j \mid \boldsymbol{e}, f_k).$$

By Bayes' law we have:

$$\mathrm{P}(A_k{=}j \mid \boldsymbol{e}, f_k) \;=\; \frac{\mathrm{P}(f_k \mid A_k{=}j, \boldsymbol{e})\,\mathrm{P}(A_k{=}j \mid \boldsymbol{e})}{\mathrm{P}(f_k \mid \boldsymbol{e})} \tag{2.14}$$

$$\;=\; \frac{\mathrm{P}(f_k \mid A_k{=}j, e_j)\,\mathrm{P}(A_k{=}j \mid \boldsymbol{e})}{\sum_{j'} \mathrm{P}(f_k \mid A_k{=}j', e_{j'})\,\mathrm{P}(A_k{=}j' \mid \boldsymbol{e})}. \tag{2.15}$$

In IBM model 1 all alignments are equally likely, i.e., $\mathrm{P}(A_k{=}j|\boldsymbol{e}) = \mathrm{P}(A_k{=}j'|\boldsymbol{e})$, so:

$$\mathrm{P}(A_k{=}j \mid \boldsymbol{e}, \boldsymbol{f}) \;=\; \frac{\mathrm{P}(f_k \mid A_k{=}j, e_j)}{\sum_{j'} \mathrm{P}(f_k \mid A_k{=}j', e_{j'})} \;=\; \frac{\tau_{e_j, f_k}}{\sum_{j'} \tau_{e_{j'}, f_k}}. \tag{2.16}$$

which is Equation 2.8. Note that we have just used the all-alignments-are-equiprobable assumption.

2.3 IBM model 2

So far we have assumed that the probabilities of all alignments $\boldsymbol{a}$ are the same. As this is not a very good assumption, IBM model 2 replaces it with the assumption that a word at position k in the source language (French) will be moved to position j in the target with probability $\mathrm{P}(A_i = j|k, l, m)$, where as before l and m are the lengths of the English and French sentences respectively. Once we add these probabilities to our equations, we let EM estimate them, just like the word translation probabilities.

To see precisely how this works, let us go back and remember where the offending assumption (all alignments are equally likely) was introduced into our equations:

$$\begin{aligned} P(\boldsymbol{f} \mid \boldsymbol{e}) &= \sum_{\boldsymbol{a}} P(\boldsymbol{f}, \boldsymbol{a} \mid \boldsymbol{e}) \\ &= \sum_{\boldsymbol{a}} P(\boldsymbol{a} \mid \boldsymbol{e})\ P(\boldsymbol{f} \mid \boldsymbol{a}, \boldsymbol{e}) \\ &= \mathrm{P}(m \mid l) l^{-m} \sum_{\boldsymbol{a}} P(\boldsymbol{f} \mid \boldsymbol{a}, \boldsymbol{e}). \end{aligned} \tag{2.17}$$

It is in the last equation that we replaced $P(\boldsymbol{a} \mid \boldsymbol{e})$ with $\mathrm{P}(m \mid l) l^{-m}$. We back out of this assumption by reverting to the previous equation, so we now need to compute $\mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e})$. We get our formula for this as follows:

$$\begin{aligned} P(\boldsymbol{a} \mid \boldsymbol{e}) &= \mathrm{P}(m \mid l)\ P(\boldsymbol{a} \mid l, m) \\ &= \mathrm{P}(m \mid l) \prod_{k=1}^{m} \mathrm{P}(A_k = j \mid i, l, m). \end{aligned}$$

The first line assumes that the only thing the alignment probability takes from the English sentence is its length. The second line assumes that each alignment probability is independent of the others, and thus the probability of the total alignment is just the product of the individual alignment probabilities.

So IBM model 2 needs a new set of parameters $\delta_{j,k,l,m}$ that are estimates of $P(A_k = j \mid k, l, m)$, the probability that the French word in position k is aligned with the English word in position j given the position of the French word (k) and the lengths of the two sentences (l, m). These parameters are often called the *distortion probabilities*, whence the use of the Greek letter δ:

$$P(\boldsymbol{a} \mid \boldsymbol{e}) = \eta_{l,m} \prod_{k=1}^{m} \delta_{j,k,l,m}. \tag{2.18}$$

Example 2.9: Suppose we have the two sentences '*The white bread*' and '*Le pain blanc*' with the alignment $a =< 1, 3, 2 >$. Then

$$P(\boldsymbol{a} \mid \boldsymbol{e}) \quad = \quad \eta_{3,3} \cdot \delta_{1,1,3,3} \cdot \delta_{2,3,3,3} \cdot \delta_{3,2,3,3}. \tag{2.19}$$

Example 2.10: Suppose we have a French sentence of length 10 and we are looking at the second word. Our English sentence is of length 9. The distortion parameter for the case that $A_2 = 2$ is $\delta_{2,2,9,10}$. A reasonable value of this parameter would be around 0.2. We would expect this to be much larger than say, $\delta_{7,2,10,9}$, which might be, say, 0.01.

Example 2.11: Some combinations of δ parameters cannot correspond to any real situation and thus have probability zero, e.g. $\delta_{20,2,10,9}$. The second French word cannot be aligned to the twentieth English one when the English sentence has only nine words. We condition on the sentence lengths so as not to assign probability mass to such parameters.

In practice it is often convenient to index distortion probabilities on both word positions and length of the French sentence, but not the English. When we actually use them to translate we will know the former but not the latter, at least not until the translation is completed.

Example 2.12: The assumption that the alignment probabilities are independent of each other is not a very good one. To give a specific example, both French and English allow prepositional phrases like '*last Tuesday*' to migrate to the beginning of a sentence. In English we have:

> I will write the letter next week.
> Next week I will write the letter.

Think about the "alignment" of these two sentences (i.e., think of the first one as the French sentence). The correct alignment is $< 3, 4, 5, 6, 1, 2 >$. In general, the probability that $A_6 = 2$, as here, is low. However, if we knew that $A_5 = 1$, then it is much more likely. So distortion probabilities are not really independent of each other. In section 3.7 we show a clever way to handle these dependencies. For now we simply note that, although assuming that alignment probabilities are mutually independent is not great, it is certainly better than assuming they are all equal.

If we now substitute our alignment probability from Equation 2.18 into Equation 2.18, we get:

$$\begin{aligned} P(\boldsymbol{f} \mid \boldsymbol{e}) \quad &= \quad \sum_{\boldsymbol{a}} P(\boldsymbol{a} \mid \boldsymbol{e})\ P(\boldsymbol{f} \mid \boldsymbol{a}, \boldsymbol{e}) \\ &= \quad \eta_{l,m} \sum_{\boldsymbol{a}} \prod_{k=1}^{m} \delta_{a_k,k,l,m}\ \tau_{e_{a_k},f_k}. \end{aligned} \tag{2.20}$$

Note how similar this new equation is to our IBM model 1 equation, Equation 2.5. Before we multiplied all of the translation probabilities together. Now we do the same thing, but multiply in one alignment probability for each translation probability. Furthermore, this similarity carries over to our fractional count equation. The new equation replacing Equation 2.9 makes the identical transformation:

$$n_{e_{a_k},f_k} += \frac{\tau_{e_{a_k},f_k}\ \delta_{a_k,k,l,m}}{\sum_{j'=0}^{\ell} \tau_{e_{j'},f_k}\ \delta_{j',k,l,m}}. \tag{2.21}$$

Thus to learn IBM model 2 parameters we do the same thing we did for IBM 1, with some small differences. First, we need to pick initial values for δ as well (all equal as before). Second, to get the fractional counts we multiply in the δs as in Equation 2.21. Last, the fractional counts thus obtained are used to get new counts not only for the τ but for the δ as well.

Once we have the proof that we have correctly computed model 1 expected counts, only a small change will show that model 2 is correct as well. Note that the assumption that all alignments were equal came in only at the very last step, going from Equation 2.15 to Equation 2.16. But for model 2 we have:

$$\begin{aligned} \mathrm{P}(A_k{=}j|\boldsymbol{e},m) &\neq \mathrm{P}(A_k{=}j'|\boldsymbol{e},m) \\ &= \delta_{j,k,l,m}. \end{aligned}$$

Substituting the δs into Equation 2.16, we get Equation 2.21 and we are done.

As we have already noted, in general EM is not guaranteed to find estimates of the parameters that maximize the probability of the data. It may find a local maximum but not the global one. As remarked in Section 2.2.3, IBM 1 has only has one maximum and thus we are guaranteed to find it as long as none of the initial estimates are zero. This is not true for IBM 2 — the initial estimates matter.

In practice, we start EM ignoring the δs. That is, we set them all equal, but do not update them for several iterations — long enough for the τs to be "burned in" with reasonable values. Only then do we start collecting fractional counts for the δs and resetting them in the E-step. In practice this tends to find very good estimates.

2.4 Phrasal machine translation

Beyond IBM model 2 lie 3, 4, and 5. While they are important, certainly the most important difference between the MT we have presented and MT as

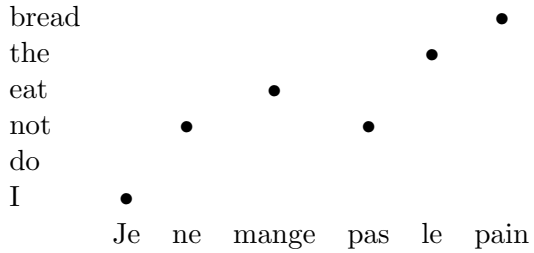

Figure 2.7: A graph of the alignment for the $(f)/(e)$ pair '*Je ne mange pas le pain*' and '*I do not eat the bread*'

it is practiced today is the introduction of phrases into machine translation — something not found in any of the IBM models.

You remember that IBM models 1 and 2 share the assumption that each French word is triggered by exactly one English word. This is captured by the alignments we introduced early in our discussion. Each alignment ties a French word to one, and only one English word (though it may be the null word).

Unfortunately, this assumption is far from correct. Consider the following $(f)/(e)$ pair:

Je ne mange pas le pain
I do not eat the bread

The best alignment is:

$< 1, 3, 4, 3, 5, 6 >$

'*Je*', '*mange*', '*le*', and '*pain*' align with '*I*', '*eat*', '*the*' and '*bread*' respectively. Both '*ne*' and '*pas*' align with '*not*'. Figure 2.7 shows a graph of this alignment, with the French words along the x axis and the English along the y axis. We have put a dot at the (x, y) position when $a_x = y$.

Notice that nothing aligns with the English word '*do*'. This is problematic when we want to translate, and here is where phrasal alignments kick in. We start the same way, but then we also do the reverse alignment — align each English word to exactly one French word. We do this by creating a second noisy-channel model, this time for computing $\mathrm{P}(\boldsymbol{e} \mid \boldsymbol{f})$ rather than the $\mathrm{P}(\boldsymbol{f} \mid \boldsymbol{e})$ we have been computing so far. Nothing in the IBM models cares about which language is which, so in principle this should be as simple as flipping the sentence pairs in the parallel corpus.

Going back to our '*do*' example, an English-to-French alignment might be:

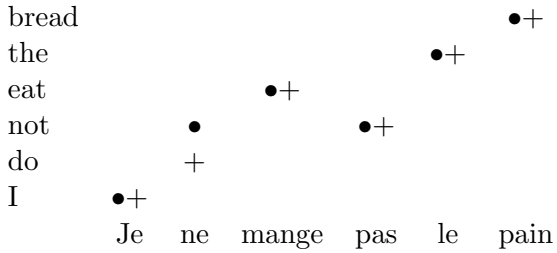

Figure 2.8: Both English to French and French to English alignments

$< 1, 2, 4, 3, 5, 6 >$

This is similar to the French-to-English, except this time we are forced to align '*do*' with something. In our example we have chosen to align it with '*ne*'. (Another possibility would be to align it to the null French word. This choice is not critical to our discussion.)

In Figure 2.8 we superpose the two alignments, representing the second alignment with the +'s. The figure clarifies a few things. First, '*Je/I*', '*le/the*', and '*pain/bread*' are in good shape. In each case there is a one-for-one alignment, and furthermore the alignment matches up with the words on at least one side. By this we mean that '*Je*' and '*I*' are both preceded by the start of sentence, '*le*' and '*the*' are both followed by words that are themselves aligned ('*pain*' and '*bread*'), and these last two words are preceded by aligned words and followed by the end of the sentence. This suggests that these words are reasonably translated in a one-to-one fashion.

None of the other words have this property. In particular, the relation between '*ne mange pas*' and '*do not eat*' is messy, and the words at their ends do not match up nicely with the words that precede/follow them. On the other hand, suppose we treat '*ne mange pas*' as a single phrase that should be translated into '*do not eat*' as a whole, not word for word. While the words within it do not match up very well, the phrases '*ne mange pas*' and '*do not eat*' match well in exactly the same way that the '*good*' words do: they are preceded and followed by good matches. That is the second key idea of phrasal MT — use graphs like that in Figure 2.8 as clues that some sequence of words should be aligned not individually but as phrases. We then keep alignment counts for these phrases, just as we did for individual words. Finally, at translation time we treat such phrases as if they were single words.

Unfortunately, this simple version of phrasal MT does not work very well. We show here two ways in which we have grossly oversimplified things.

First, the example in Figure 2.8 was deliberately chosen to be simple. In real examples, the graph of alignment pairs can get very messy and our vague description of how to find phrases would end up, say, aligning entire sentences as a single phrase. Second, even if we are lucky and never get phrases longer than, say, four words, the number of four-word phrases is very, very large, and there are not sufficient parallel corpora for us to collect statistics on even a fraction of what can occur. So we do not see merely '*ne mange pas*' and '*do not eat*'. Virtually any French verb and its English translation could be substituted here. What we really need to do is recognize that there is a pattern of the form '*ne* `french_verb` *pas*' that becomes '*do not* `English_verb`'. This is exactly what a real implementation does, but exactly how it works differs from system to system.

Actually, what we ought to do is to consider all possible phrasal translations and then, say, use EM to pick out good ones, just as we did for single words. Unfortunately this is not possible. Suppose we have a FE pair with each sentence thirty words long. How many phrasal pairs are there? Well, if we assume that phrases are always made up of contiguous words, there are 2^{29} possible phrases for the English, and the same number for the French. (Show this!) If each English phrase can align with any of the French ones, we get 2^{29} squared possibilities, or 2^{58} combinations. This is a seriously big number. So any method, such as EM, that requires simultaneously storing all expectations is out of the question. We come back to this point near the end of the book.

2.5 Decoding

There is, of course, one major problem with our MT efforts so far. We can gather probabilities, but we don't know how to use them to translate anything.

In this section we rectify this situation. However, rather than calling this section "translation," we call it "decoding," as this is what it is called in the MT community. The reason for this odd nomenclature is the origin of our fundamental theorem of MT, Equation 2.1, which is a particular instance of the noisy-channel model. As its name implies, the noisy-channel model was invented to overcome problems of noise on a communication channel. The key to all solutions to this problem is to encode the message in some way so as to make it possible to reconstruct the message even if noise obliterates portions of it. The process of finding the message from the coded version sent over the channel is, naturally enough, called "decoding." Since when

viewed as a noisy-channel problem MT imagines that a message originally written in our target, say English, arrived written in the source, say French, the problem of getting the English from the French is thus a "decoding" problem.

The MT decoding problem is, in its most general form, very difficult. Consider again the MT fundamental equation (2.1) repeated here:

$$\arg\max_{\boldsymbol{e}} P(\boldsymbol{e} \mid \boldsymbol{f}) = \arg\max_{\boldsymbol{e}} P(\boldsymbol{e})P(\boldsymbol{f} \mid \boldsymbol{e}).$$

We have now shown how to get the required language model $P(\boldsymbol{e})$ and translation model $P(\boldsymbol{f} \mid \boldsymbol{e})$, but solving the equation requires iterating over all possible English sentences — a daunting task. (For those of you who have taken a computational theory class, the problem is NP-hard.)

2.5.1 Really dumb decoding

As a general rule it is almost always worth doing something really simple before trying to do something very hard. Sometimes the dumb method works and you save a lot of work. More often it doesn't, but you gain some insight into the problem.

Here is our first feeble attempt. All this time we have been using EM to estimate $P(\boldsymbol{f} \mid \boldsymbol{e})$. Let's redo the programs to compute $P(\boldsymbol{e} \mid \boldsymbol{f})$. We then take our French sentence and for each French word substitute the English word $t(f_k)$ which we define as

$$t(f_k) = \arg\max_{e} P(e \mid f_k).$$

So $t(f_i)$ is a function from a French word f_i to the English word e that is its most likely translation out of context. Then the translation of the sentence $\boldsymbol{f}$ is the concatenation of $t(f_k)$ for $1 \leq i \leq m$. Now the τ's estimate $P(\boldsymbol{e} \mid \boldsymbol{f})$ rather than $P(\boldsymbol{f} \mid \boldsymbol{e})$. Finding these is *really* easy. Just take the program you had earlier, and switch the input files! That's it.

Unfortunately the results are pretty bad.

Example 2.13: Looking at the output of such a very simple decoder is a good way to appreciate the complexities of MT. We start with an example where things go relatively well, and then descend rapidly into the realm of unfunny machine-translation jokes. (The classic, which is surely apocryphal, is that the Russian equivalent of "The spirit is willing but the flesh is weak" came out as "The vodka is good but the meat is rotten".)

English:	That is quite substantial .
French:	Ce est une somme considerable .
MT output:	That is a amount considerable .

Here the program got it right except for the reversal of '*amount considerable*'.

English:	I disagree with the argument advanced by the minister .
French:	Je ne partage pas le avis de le ministre .
MT output:	I not share not the think of the Minister .

'*Je ne partage pas*' means '*I do not share.*' However, the split '*ne*' and '*pas*' both get translated as '*not*', and the '*do*' in the English version, which from the French point of view would be spurious, is not included. '*Avis*' means '*opinion*', but one can easily imagine that it gets translated into a phrase like '*I think*'.

English:	I believe he is looking into the matter .
French:	Je crois que il est en train de etudier la question .
MT output:	I think that he is in doing of look the question .

'*En train de*' means '*in the process of*' but comes out as '*in doing of*'.

English:	My question very briefly is this .
French:	Voici tres brièvement de quoi il se agit .
MT output:	: very briefly of what he is point .

We thought ending with '*voici*' ('*this*') being translated as a colon would be a nice touch.

2.5.2 IBM model 2 decoding

So really simple techniques do not work and we must turn to complicated ones. To make things more concrete, we consider a decoder for IBM model 2. Unfortunately, while there are standard IBM models 1 to 5, there is no such thing as a standard decoder for any of them. What we present is as typical as any, but we cannot say anything stronger.

In one sense, decoding is simple. If we substitute Equation 2.20 into the MT fundamental theorem, we get:

$$\arg\max_{\boldsymbol{e}} P(e) \cdot \eta_{l,m} \sum_{\boldsymbol{a}} \prod_{k=1}^{m} \delta_{a_k,k,l,m}\ \tau_{e_{a_k},f_k}. \tag{2.22}$$

Thus decoding "reduces" to testing every possible English sentence and seeing which one is the arg max required in Equation 2.22.

Of course, this is not possible. Even if we restrict consideration to English sentences no longer than the French one, the number of such sentences grows exponentially with the lengths of the sentences involved. Furthermore, the problem is "NP hard" — it is (almost certainly) inherently exponential.

We have then a search problem — finding the needle in the haystack. Fortunately, we computational linguists are not the only people with search

1. add the empty partial solution (no words translated) to H_0
2. set the probably of the current best answer ($P(A)$) to zero.
3. for $i = 1$ to M
   (a) pick the next ranked partial solutions s from H_{i-1}
      i. extend s by adding an English word to its end, creating s',
      ii. if s' is a complete sentence then if $P(s') = h(s')\eta_{l,m} > P(A)$, $A = s', P(A) = P(s')$
      iii. else, if H_i has fewer than W entires, or if $h(s') > h(H_W)$ add s to H_i. (Removing the one with the smallest heuristic value if necessary to keep the size of H_i to W.)
      iv. else, discard s'.
4. return A.

Figure 2.9: Beam search algorithm for IBM model 2 decoding

problems. A whole branch of artificial intelligence is devoted to them — *heuristic search*. More formally, we have a problem (constructing a sentence) that requires many steps to complete (selecting English words). There may be many possible solutions, in which case we have an evaluation metric for the quality of the solution. We need to search the space of partial solutions ($\boldsymbol{S}$) until we find the optimal total solution (the $\boldsymbol{e}$ with highest $\mathrm{P}(\boldsymbol{e} \mid \boldsymbol{f})$). In heuristic search we also have a *heuristic*, an indication of the relative quality of partial solutions. In our case we use equation 2.22 on the string of English words so far, except we ignore $\eta_{l,m}$. We call this quantity $h(s)$.

There is one major problem with this heuristic however — as we add more English words the probability of the string according to Equation 2.22 tends to decrease rapidly. After all, most any long sentence is going to be less probable than shorter sentences. Thus we are going to use a *beam search*. This will look something like that in Figure 2.9. Here we have a vector of *priority queues priority queue* (also known as *heaps*) H, one for each possible length of the English sentence e. Priority queues are data structures with the property that they efficiently store a set of entities s, each with a priority ($h(s)$), so we can efficiently find the s with the next highest (or smallest) priority, and pop it off the queue. We start by putting the empty English sentence into the queue for zero word sentences, H_0. Then, given the queue

for H_{i-1} we pop off the states, extend them by adding a new possible English word, and put them on H_i. Note, however, that we only put them on H_i, if their heuristic value $h(s)$ is sufficiently high, namely greater that $h(H_i[W])$. Here W is an integer $1 < W$ known as the *beam width*. The idea is that we only save the W highest states to pass on to the $i+1$'th iteration. Also notice that we only compare a partial solution of length l against other solutions of the same length, thus overcoming the problem that partial solutions have rappedly decreasing probabilities as their lenghts increase.

We also need to recognize if s' is a complete sentence. We do this by assuming that our sentences are padded with $\triangleleft$ characters, and $\triangleleft$ is one of the English words we can add to the end of s. If s' ends in $\triangleleft$ we do not put it on the next queue, but rather compare its value to the best solution we have seen so far (now multiplying in $\eta_{l,m}$ the probability of a French sentence of lenght m given that our English sentence is of length l).

Next, note that the outermost iteration was for $i = 1$ to M, the length of the French sentence. This assumes that we will finding a good English translation that is no longer than the French sentence. This could be relaxed by making the upper bound some multiple of M, at the cost of increased processing time. Also, line i in Figure 2.9 has us adding "an English word" to the end of s. Note that although any particular English word might appear several times as translations of different f's we only will want to consider it once here. To keep this simple we make a list of all the word types we have recorded as translations of the words in $\boldsymbol{f}$

Lastly, Equation 2.22 has us summing over all possible alignments: This is exponential, but fortunately we can reverse the order of the sum and product as we did with EM training to get a polynomial algorithm.

$$\begin{aligned} h(e) &= P(e) \cdot \eta_{l,m} \sum_{\boldsymbol{a}} \prod_{k=1}^{m} \delta_{a_k,k,l,m}\ \tau_{e_{a_k},f_k} \\ &= P(e) \cdot \eta_{l,m} \prod_{k=1}^{m} \sum_{j=1}^{l} \delta_{a_k,k,l,m}\ \tau_{e_j,f_k} \end{aligned}$$

2.6 Exercises

Exercise 2.1: Consider the following English sentences:

The dog drank the coffee.
The man drank the soup.
The coffee pleased the man.

We have a parallel corpus in Fthishr, a language that loves its coronal fricatives and word-final alveolar approximants.

Thir o favezh or shath.
Sirzh o zhiidh or shath.
Favezh or sirzh o vozir.

Give the correct alignment for the sentences

The soup pleased the dog.
Zhiidh or thir o vozir.

Exercise 2.2: Compute the values of $n_{\text{`}eat\text{'},\text{`}mange}$ and $\tau_{\text{`}eat\text{'},\text{`}mange}$ after the first and second iterations of EM given the following training data:

Elle mange du pain	*She eats bread*
Il mange du boef	*He eats beef*

Exercise 2.3: In Section 2.2.1 we noted that $n_{e,\circ}$ in general will not be the same as the number of times the word e appears in the corpus. Explain.

Exercise 2.4: Would $< 0, 0, 0 >$ be a legal alignment for a French sentence of length three? (Extra credit: Mention an interesting implication of this)

Exercise 2.5: We suggest starting EM with all $\tau_{e,f}$'s the same. However, as long as they are the same (and non-zero) any one value works as well as any other. That is, after the first M-step the τs you get will not be a function of the τs you stated with. For example, you could initialize all τs to one. Prove this. However, although the τs at the end of the M-step would not vary, there would be some other variable inside the EM al rithm we suggest you compute that *would*. What is it?

Exercise 2.6: On the first iteration of IBM model 1 training, the word that would align the most often with '*pain*' is almost certainly going to be '*the.*' Why? Yet in Figure 2.6 we did not mention '*the*' as a translation of '*pain.*' Why is that?

Exercise 2.7: In our derivation of Equation 2.13 we stated that since all alignments are equally probable in IBM model 1 it follows that

$$\mathrm{P}(A_k{=}j|\boldsymbol{e}) = \mathrm{P}(A_k{=}j'|\boldsymbol{e}). \tag{2.23}$$

Actually, this deserves a bit more thought. Note that it is *not* the case that

$$\mathrm{P}(A_k{=}j|\boldsymbol{e}, \boldsymbol{f}) = \mathrm{P}(A_k{=}j'|\boldsymbol{e}, \boldsymbol{f}). \tag{2.24}$$

First, explain why Equation 2.24 is false. Then explain why Equation 2.23 is nevertheless true.

2.7 Programming problems

Problem 2.1: **Machine Translation with Very Dumb Decoding**

Write an MT program based upon IBM model 1 and our very dumb decoder, which simply goes through the incoming French sentence and for each word f_i outputs $\arg\max_{e_j} P(e_j \mid f_i)$.

Start by building an IBM model 1 parameter estimation program. It takes as arguments two input files, one for the French half of the parallel corpus, one for the English half. Make sure that your program does not care which is which, so that by switching the order of the files your program will switch between computing $P(e_j \mid f_i)$ and $P(f_i \mid e_j)$.

To initialize EM we assign all translation probabilities the same value. As noted in Exercise 2.5 you can simply set them to 1. Equation 2.9 tells us how to compute the fractional counts for each word (the e-step), and then at the end of each iteration Equation 2.7 does the m-step. Ten iterations or so should be sufficient for all the parameter values to settle down.

Before writing the decoder part of this assignment, print out some probabilities for French words with reasonably obvious English translations. Foremost should be punctuation. If you do not know any French, you can also look at the French for words that look like English words and occur several times. Many French words have been adopted into English.

Finally, the decoder should be just a few lines of code. Two details. Given the (low) quality of our decoder, only very simple sentences have any chance of being comprehensible after translation. Only translate French sentences of length ten or less. Also, when your program encounters French words it has never seen before, just pass them through to the English output without change.

Use `English-senate-0.txt` and `french-senate-0.txt` as the training data. We are not tuning any smoothing parameters, so there is no particular need for held-out data. Translate `french-senate-2.txt` and save your translation to a new file.

Problem 2.2: **MT with a Simple Noisy-Channel Decoder**

Now we'll try to do better. For this assignment use the IBM model 1 decoder from the last programming assignment, but now use it to compute the reverse translation probabilities, $P(f_i \mid e_j)$. In addition, use the English bigram language model from the programing assignment in Chapter 1. The only thing that changes is our decoder.

In this version we again translate the French one word at a time. Now, however, rather than maximizing $P(e_j \mid f_i)$, we maximize $P(e_j \mid e_{j-1})P(f_i \mid$

e_j). As we are going left to right one word at a time, we will know the previous English word at each step. (Set the zeroth word to ▷.)

Use the same training and test data as in Problem 1.

Problem 2.3: **Evaluation**

Evaluate the two different translators according to the *F-score* of their output. The F-score is a standard measure of accuracy defined as the *harmonic mean* of *precision* and *recall*:

$$F = 2\frac{precision \cdot recall}{precision + recall}. \tag{2.25}$$

Precision is the number of correct results divided by the number of all returned results, and recall is the number of correct results divided by the number of results that should have been returned. In this case, we are counting the number of individual word tokens translated correctly. You may consider a translation of the word "pain" in `french-senate-2.txt` to "bread" as correct if "bread" occurs anywhere in the corresponding sentence in `english-senate-2.txt`. The total number of returned results is, of course, the word-token count of your translated output file (which will be exactly the same as that of `french-senate-2.txt`), and the number of results that should have been returned is the word-token count of `english-senate-2.txt`.

Which translation looks better to you (this is a rather subjective question)? Which gives the better F-score? Why? What does this tell us about our two different decoders? What about our chosen method of evaluation?

Include the answers to these questions in your README.

2.8 Further reading

For the most part, the best key words for MT are the obvious ones used in this chapter. However, if you want to download the *Canadian Hansard Corpus*, use this as the search key in a typical search engine; Google Scholar will list only papers citing the corpus. Another corpus is the *Europarl Corpus*.

You might want to try out an *open-source machine translation* system.

A recent trend is *grammar-based machine translation*, also called *hierarchical phrase-based* MT. The most frequent out-of-vocabulary items in MT are names of people and places. The best way to handle them is to learn how to *transliterate* them to get English equivalents.

One big problem is translation of a *low-resource language* indexlow-reseource languages — a language for which there are few parallel corpora.

One way to do this is to use a *bridging language* — a language similar to the low-resource target for which we have much more data. For example, to help translate Portuguese, use data from Spanish, or for Slovak, use Czech.

Finally, we would be remiss if we did not mention one of the best tutorials on MT, Kevin Knight's *MT Workbook*. The material covered is much the same as that in this chapter, but from a slightly different point of view. It is also enjoyable for Kevin's wacky sense of humor.