
A Statistical MT Tutorial Workbook

Kevin Knight

prepared in connection with the JHU summer workshop

April 30, 1999

---

1. The Overall Plan

We want to automatically analyze existing human sentence translations, with an eye toward building general translation rules -- we will use these rules to translate new texts automatically.

I know this looks like a thick workbook, but if you take a day to work through it, you will know almost as much about statistical machine translation as anybody!

The basic text that this tutorial relies on is Brown et al, "The Mathematics of Statistical Machine Translation", Computational Linguistics, 1993. On top of this excellent presentation, I can only add some perspective and perhaps some sympathy for the poor reader, who has (after all) done nothing wrong.

Important terms are underlined throughout!

---

2. 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 to formalize "more likely":

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. Not bloody likely, in this case.

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). 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) = 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.

Exercise. P(e,f) = P(f) * ?

All these probabilities are between zero and one, inclusive. A probability of 0.5 means "there's a half a chance."

---

3. Sums and Products

To represent the addition of integers from 1 to n, we write:

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n$$

For the product of integers from 1 to n, we write:

$$\prod_{i=1}^{n} i = 1 * 2 * 3 * \ldots * n$$

If there's a factor inside a summation that does not depend on what's being summed over, it can be taken outside:

$$\sum_{i=1}^{n} i * k = k + 2k + 3k + \ldots + nk = k \sum_{i=1}^{n} i$$

Exercise.

$$\prod_{i=1}^{n} i * k = ?$$

Sometimes we'll sum over all strings e. Here are some useful things about probabilities:

$$\sum_{e} P(e) = 1$$

$$\sum_{e} P(e \mid f) = 1$$

$$P(f) = \sum_{e} P(e) * P(f \mid e)$$

You can read the last one like this: "Suppose f is influenced by some event. Then for each possible influencing event e, we calculate the chance that (1) e happened, and (2) if e happened, then f happened. To cover all possible influencing events, we add up all those chances.

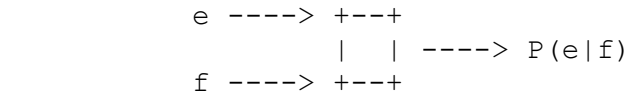
---

4. Statistical Machine Translation

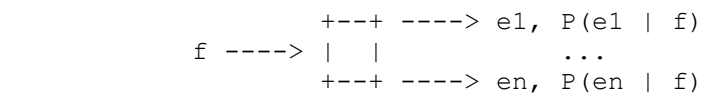
Given a French sentence f, we seek the English sentence e that maximizes P(e | f). (The "most likely" translation). Sometimes we write:

$$\operatorname*{argmax}_{e} P(e \mid f)$$

Read this argmax as follows: "the English sentence e, out of all such sentences, which yields the highest value for P(e | f). If you want to think of this in terms of computer programs, you could imagine one program that takes a pair of sentences e and f, and returns a probability P(e | f). We will look at such a program later on.



Or, you could imagine another program that takes a sentence f as input, and outputs every conceivable string ei along with its P(ei | f). This program would take a long time to run, even if you limit English translations some arbitrary length.



---

5. The Noisy Channel

Memorize Bayes Rule, it's very important!

$$P(e|f) = P(e) * P(f \mid e) / P(f)$$

Exercise: Now prove it, using the exercise in section 2.

Using Bayes Rule, we can rewrite the expression for the most likely translation:

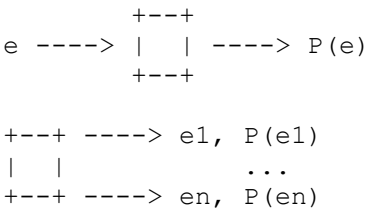
$$\operatorname*{argmax}_{e} P(e \mid f) = \operatorname*{argmax}_{e} P(e) * P(f \mid e)$$

Exercise: What happened to P(f)?

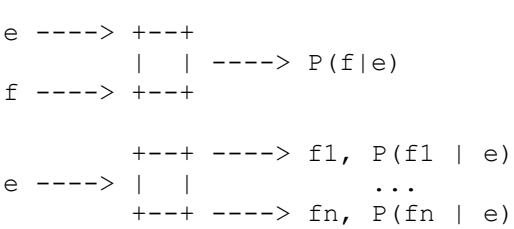
That means the most likely translation e maximizes the product of two terms, (1) the chance that someone would say e in the first place, and (2) if he did say e, the chance that someone else would translate it into f.

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.

If you want to think of P(e) in terms of computer programs, you can think of one program that takes any English string e and outputs a probability P(e). We'll see such a program pretty soon. Or, likewise, you can think of a program that produces a long list of all sentences ei with their associated probabilities P(ei).

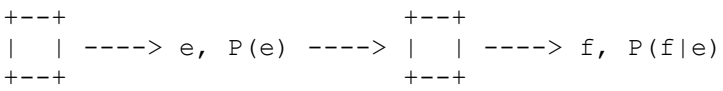


To think about the P(f | e) factor, imagine another program that takes a pair of sentences e and f, and outputs P(f | e). Or, likewise, a program that takes a sentence e and produces various sentences fi along with corresponding probabilities P(fi | e).



These last two programs are sort of like the ones in section 4, except P(f | e) is not the same thing as P(e | f).

You can put the source and channel modules together like this:



There are many ways to produce the same French sentence f. Each way corresponds to a different choice of "source" sentence e. Notice that the modules have arrows pointing to the right. This is called a generative model because it is a theory of how French sentences get generated. The theory is, first an English sentence is generated, then it gets turned into French. Kind of a weird theory.

---

6. Bayesian Reasoning

Even though the arrows point to the right, we will actually use this setup to translate French back into English.

Think of a sentence f like a crime scene. We want to reason about how this crime scene got to be. Our generative model might be something like: some person e decided to do the crime, and then that person actually did the crime. So we start reasoning about (1) who might have made the decision (P(e): motive, personality) and also (2) how they might have gone about it (P(f | e): transportation, weapons). In general, these two things may conflict. You might have someone with a good motive, but without the means; or you might have someone who could easily have done the crime, but has no motive.

Or, think of a sentence f like a set of medical symptoms. There are many diseases that could give rise to these symptoms. If we build a generative model, then we can reason about the probability of any disease e

occurring, as well as the probability that symptoms f will arise from any particular disease e. That's P(e) and P(f | e) again. They may conflict: you may have a common disease that often gives rise to symptoms f, and you may have a very rare disease that always gives rise to symptoms f. That's a tough call, right?

Since biologists know roughly how diseases cause symptoms, i.e. P(f | e), it's possible to build computer models of how this happens. It's not so obvious how to build a single model that reasons from symptoms to diseases, i.e. P(e | f). Furthermore, we may have independent sources of information about P(e) in isolation, such as old hospital records.

---

7. Word Reordering 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 a 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 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. Or to put it 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.

Exercise. Put these words in order: "actual the hashing is since not collision-free usually the is less perfectly the of somewhat capacity table"

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

Exercise. Put these words in order: "loves John Mary"

The last exercise is hard. 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. But, maybe it only needs to know a little bit about word order, not everything.

---

8. 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 "zone end the in is she" which probably 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.

Exercise. Write down a short foreign-language sentence. Rearrange the words so that they appear in "English word order." Use a bilingual dictionary to look up all possible translations of all the words. Write each word's translations in a column below the word. Erase the original foreign-language words. Ask a friend (or enemy) to construct an English sentence by choosing a single word from each column.

---

9. Language Modeling

Where do all these probability numbers come from?

Later, we'll worry about P(f | e). First we need to build a machine that assigns a probability P(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.

Another 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. We can use the Web or the online Wall Street Journal if we don't want to do a lot of actual recording.

Exercise. Which has a higher probability: "I like snakes" or "I hate snakes"? Type both in to www.altavista.com and find out (include quotation marks in your query to ensure that the words appear together and in order in the retrieved documents).

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

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 snakes poisonous I are." 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."

People seem to be able to judge whether or not a string is English without storing a database of utterances. (Have you ever heard the sentence "I like snakes that are not poisonous"? Are you sure?). We seem to be able to break the sentence down into components. If the components are good, and if they combine in

reasonable ways, then we say that the string is English.

---

10. N-grams

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, nerds say unigram, and normal people say word.

If a string has a lot of reasonable n-grams, then maybe it is a reasonable string. Not necessarily, but maybe.

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:

b(y | x) = number-of-occurrences("xy") / number-of-occurrences("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 a generative model (section 8). This model says that people keep spitting out words one after another, but they can't remember anything except the last word they said. That's a crazy model. 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 | x y) = number-of-occurrences("xyz") / number-of-occurrences("xy")

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)

---

11. 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, we can do 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 | x y) = number-of-occurrences("xyz") / number-of-occurrences("xy")

we can use

b(z | x y) = 0.95 * number-of-occurrences("xyz") / number-of-occurrences("xy") +
0.04 * number-of-occurrences ("yz") / number-of-occurrences ("z") +
0.008 * number-of-occurrences("z") / total-words-seen +
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 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. Sometimes people say ungrammatical things after all.

We'll have more to say about estimating those smoothing coefficients in a minute.

This n-gram business is not set in stone. You can come up with all sorts of different generative models for language. For example, imagine that people first mentally produce a verb. Then they produce a noun to the left of that verb. Then they produce adjectives to the left of that noun. Then they go back to the verb and produce a noun to the right. And so on. When the whole sentence is formed, they speak it out loud.

---

12. 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. The verb-noun-adjective model described at the end of the previous section is perhaps not so easy to train. For one thing, you need to be able to identify the main verb of a training sentence. And even if you could train it easily, it's not clear that it would "work better" than an n-gram model.

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:

P(model | test-data) = P(model) * P(test-data | model) / P(data)

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).

Exercise. What happened to P(data)?

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.

---

13. Perplexity

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

$$2^{-\log(P(e))/N}$$

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 a 3-word test set: 1/4, 1/2, 1/4. Then P(test-set) = 1/4 * 1/2 * 1/4 = 0.03125. What is the perplexity? 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. What is its P(test-set)? What is its perplexity?

Exercise. If P(test-set) = 0, what is the perplexity of the model?

Why do you think it is called "perplexity"?

---

14. Log Probability Arithmetic

Another problem with P(e) is that tiny numbers will easily underflow any floating point scheme. Suppose P(e) is the product of many factors f1, f2, f3 ... fn, each of which is less than one. Then there is a trick:

log(P(e)) = log(f1 * f2 * f3 * ... * fn) = log(f1) + log(f2) + log(f3) + ... + log(fn)

If we store and manipulate the log versions of probabilities, then we will avoid underflow. Instead of multiplying two probabilities together, we add the log versions. You can get used to log probabilities with this table:

p	log(p)
0.0	-infinity
0.1	-3.32
0.2	-2.32
0.3	-1.74
0.4	-1.32
0.5	-1.00
0.6	-0.74
0.7	-0.51
0.8	-0.32
0.9	-0.15
1.0	-0.00

So (0.5 * 0.5 * 0.5 * 0.5 * ... * 0.5 = 0.5^n) might get too small, but (- 1 - 1 - 1 - ... - 1 = -n) is manageable.

A useful thing to remember about logs is this: log-base-2(x) = log-base-10(x) / log-base-10(2).

So far, we have done nothing with probabilities except multiply them. Later, we will need to add them. Adding the log versions is tricky. We need a function g such that g(log(p1), log(p2)) = log(p1 + p2). We could simply turn the two log probabilities into regular probabilities, add them, and take the log. But the first step would defeat the whole purpose, because the regular probability may be too small to store as a floating point number. There is a good approximate way to do it, and I'll make it an optional exercise (hint: if there is a large distance between the two log probabilities, then one of them can be ignored in practice).

---

15. Translation Modeling

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

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. We need to tell 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 ... what English string e is (1) likely to be uttered, and (2) likely to subsequently translate to f? We're looking for the e that maximizes P(e) * P(f | e).

How does English become French? A good story 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. For example, "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 story is that predicate logic then gets converted to French. I like this story.

Another story is that an English sentence gets parsed syntactically. 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 story is called "syntactic transfer."

Yet another story is that the words in an English sentence are replaced by French words, which are then scrambled around. What kind of a crackpot story is that? That's the story we're going to use for the rest of this workbook. We will refer to this story in somewhat sinister fashion as IBM Model 3.

For one thing, this story 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 stories, 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 in sections 7 and 8 that some of the slack will be taken up by the independently-trained P(e) model.

Let's look at the story 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 story:

1. For each word ei 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 ei, we generate phi-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.

Is that a funny story, or what?

---

16. Translation as String Rewriting

You can think of this generative story as one of string rewriting. First you start with an English string like this:

Mary did not slap the green witch

Then you assign fertilities. For fertility 1, you simply copy the word over. For fertility 2, you copy the word twice. Etc. For fertility zero, you drop the word. For example, we might assign fertility zero to the word "did." This yields another string:

Mary not slap slap slap the the green witch

Notice that "slap" got fertility 3. Next you replace the English words with French words (let me use Spanish here, since I know Spanish). At this point, the replacement can be one-for-one:

Mary no daba una botefada a la verde bruja

Finally, we permute those words:

Mary no daba una botefada a la bruja verde

Exercise. Start with the sentence "my child, you must go home" and transform it into a foreign-language translation using the above generative story.

---

17. Model 3

In language modeling, we began with a story like "people produce words based on the last word they said" and wound up with a model with lots of parameters like b(snakes | like). We could obtain values for these parameters by looking at language data. We're going to do the same thing now.

In thinking about what parameters to have, we need only look at the dependencies in the generative story. For example, which French word to generate depends only on the English word that is doing generating. So it is easy to imagine a parameter like t(maison | house), which gives the probability of producing "maison" from "house." Likewise for fertilities. We can have parameters like n(1 | house), which gives the probability that "house" will produce exactly one French word whenever "house" appears. And likewise for what are called (in super-sinister fashion) distortion parameters. For example, d(5 | 2) may give the probability that an English word in position 2 (of an English sentence) will generate a French word that winds up in position 5 (of a French translation). Actually, Model 3 uses a slightly enhanced distortion scheme in which the target position also depends on the lengths of the English and French sentences. So the parameters look like d(5 | 2, 4, 6), which is just like d(5 | 2) except also given that the English sentence has four words and French sentence has six words. Remember that the distortion parameters do not need to

place each French word carefully into a grammatical whole. It might be okay if this model generates bad French.

Model 3 has one more twist. This is really sinister. We imagine that some French words are "spurious." That is, they mysteriously appear in a French translation although no English word can really be held responsible for them. For example, function words. Consider the Spanish word "a" in the example in the last section. I assigned a fertility of 2 to the English word "the", which ultimately generated the Spanish "a". Maybe it's better to assign a fertility of 1 to "the", and let "a" be generated spuriously.

In modeling this phenomenon, we pretend that every English sentence contains the invisible word NULL in the initial, zero-th position. We have parameters like t(maison | NULL) that give the probability of generating various words spuriously from NULL.

We could also have parameters like n(3 | NULL), which would give the probability of there being exactly 3 spurious words in a French translation. But, Model 3 worries that longer sentences will have more spurious words. I know, this is like a sudden attack of scruples. Weird. Anyway, instead of n(0 | NULL) ... n(25 | NULL), we're going to have a single floating-point parameter called p1. You can think of p1 like this. After we assign fertilities to all the "real" English words (excluding NULL), we will be ready to generate (say) z French words. As we generate each of these z words, we optionally toss in a spurious French word also, with probability p1. We'll refer to the probability of not tossing in (at each point) a spurious word as p0 = 1 - p1.

Exercise. If the maximum fertility for English words were 2, how long is the longest French sentence that can be generated from an English sentence of length l? (Don't forget about NULL-generated words).

Finally, we need to worry about distortion probabilities for NULL-generated words. I'm afraid it is too simple to have parameters like d(5 | 0, 4, 6), i.e., the chance that the NULL word will generate a French word that winds up in French position 5 as opposed to some other position. I mean, these words are spurious, man! They can appear anywhere! It's fruitless to try to predict it! We'll instead (1) use the normal-word distortion parameters to choose positions for normally-generated French words, and then (2) put the NULL-generated words into the empty slots that are left over. If there are three NULL-generated words, and three empty slots, then there are 3! ("three factorial"), or six, ways for slotting them all in. We'll just assign a probability of 1/6 for each way.

Now we are ready to see the real generative Model 3:

1. For each English word ei indexed by i = 1, 2, ..., l, choose fertility phi-i with probability n(phi-i | ei).
2. Choose the number phi-0 of "spurious" French words to be generated from e0 = NULL, using probability p1 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, ..., l, and each k = 1, 2, ..., phi-i, choose a French word tau-ik with probability t(tau-ik | ei).
5. For each i = 1, 2, ..., l, and each k = 1, 2, ..., phi-i, choose target French position pi-ik with probability d(pi-ik | i,l,m).
6. For each k = 1, 2, ..., phi-0, choose a position pi-0k from the phi-0 - k + 1 remaining vacant positions in 1, 2, ... m, for a total probability of 1/phi-0!.
7. Output the French sentence with words tau-ik in positions pi-ik (0 <= i <= l, 1 <= k <= phi-i).

If you want to think about this in terms of string rewriting, consider the following sequence:

Mary did not slap the green witch (input)

Mary not slap slap slap the green witch (choose fertilities)
Mary not slap slap slap NULL the green witch (choose number of spurious words)
Mary no daba una botefada a la verde bruja (choose translations)
Mary no daba una botefada a la bruja verde (choose target positions)

---

18. Model 3 Parameters

You can see that the model has four types of parameters: n, t, p, and d.

n and t are two-dimensional tables of floating-point numbers. d is a one-dimensional table of floating-point numbers. p is a single floating-point number.

There's nothing mysterious about a model: it's mostly just a bunch of numbers!

Exercise. If each floating-point number takes up four bytes of storage, how much space do you think we need for all these tables?

---

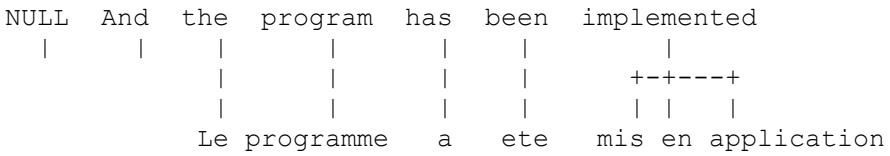
19. Word-for-Word Alignments

Now we need to think about two things. First, how can we automatically obtain parameter values from data? Second, armed with a set of parameter values, how can we compute P(f | e) for any pair of sentences?

First let's think about automatically obtaining values for the n, t, p, and d parameters from data. If we had a bunch of English strings and a bunch of step-by-step rewritings into French, then life would be easy. To compute n(0 | did), we would just locate every instance of the word "did" and see what happens to it during the first rewriting step. If "did" appeared 15,000 times and was deleted during the first rewriting step 13,000 times, then n(0 | did) = 13/15.

Exercise. Take 10,000 English sentences and rewrite them into French, storing all intermediate strings. No, make that a million English sentences! Ha, ha, just kidding. Don't do this exercise.

It will be very useful for us to consider a data structure that is simple, yet preserves most of the decisions made by the model. This data structure is called a word alignment. Here is a sample word alignment:



(Notice the invisible English NULL word in initial, zeroth position.)

If we had a lot of sentence pairs aligned like this, then again, life would be easy. To compute n(0 | did), we just see how many times "did" connected to zero French words. To compute t(maison | house), we count up all the French words generated by all the occurrences of "house," and see how many of those words are "maison."

The word alignments that best correspond to the Model 3 story are those in which every French word is connected to exactly one English word (either a regular word or NULL). So, it's never the case that two English words jointly generate some French word. That's too bad, because sometimes that's a very intuitive alignment.

Since the computer doesn't like drawings very much, it will be convenient to represent at alignment as a vector of integers. This vector has the same length as the French sentence, and each entry corresponds to a particular French word. The value we store for each entry is the position of the English word that the particular French word connects to in the alignment drawing. We can represent the sample word alignment above as [2, 3, 4, 5, 6, 6, 6].

Exercise. Draw a word-for-word alignment between the two sentences "Mary did not slap the green witch" and "Mary no daba una botefada a la bruja verde."

Exercise. What is the vector representation for the alignment you drew in the previous exercise?

Notice that the word-for-word alignment does not quite preserve all of the model's decisions in transforming an English sentence into a French one. For example, if a single English word x is connected to French words y and z, then we don't really know whether they were generated in that order, or whether they were generated as z and y, then permuted. This will have ramifications later, when we figure out how to calculate P(f | e).

---

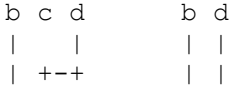
20. Estimating Parameter Values from Word-for-Word Alignments

We mentioned above how to estimate n and t parameter values from aligned sentence pairs. It is also easy to estimate d parameter values. Every connection in an alignment contributes to the count for a particular parameter such as d(5 | 2, 4, 6). Once we obtain all the counts, we normalize to get the probabilistic parameter values. For example, the count dc(5 | 2, 4, 6) would be 125 if we observed 125 times some English word in position 2 generating some French word in position 5, when the respective lengths of the sentences were 4 and 6. If "total" is the sum for all j of dc(j | 2, 4, 6), then d(5 | 2, 4, 6) = dc(5 | 2, 4, 6) / total. Or in the sum notation described in section 3:

$$d(5 \mid 2, 4, 6) = \frac{dc(5 \mid 2, 4, 6)}{\sum_{j=1}^{25} dc(j \mid 2, 4, 6)}$$

The last thing to estimate is p1. To do this, we can consider the entire French corpus of N words. Suppose that M of those words were generated by NULL, according to the alignments provided. (The other N-M were generated by normal English words). Then, we can imagine that after each normally generated French word, a spurious word will be generated in M of the N-M cases, so p1 = M / N-M.

Exercise. Writes down estimates for n, t, p, and d parameter values based on the following word-aligned corpus:



```
| | |       | |
x y z       x y
```

---

21. Bootstrapping

To get good parameter value estimates, we may need a very large corpus of translated sentences. Such large corpora do exist, sometimes, but they do not come with word-for-word alignments. However, it is possible to obtain estimates from non-aligned sentence pairs. It's like magic. I was so happy when I finally understood this concept. Once I understood the concept in this application, I could apply it to many other problems.

Exercise. To get an intuitive feel for this, try to manually build alignments for the set of unaligned sentence pairs on page 84 of the article "Automating Knowledge Acquisition for Machine Translation," Kevin Knight, AI Magazine, Winter 1997.

A slightly more formal intuition is that pairs of English and French words which co-occur in sentence translations may indeed be translations of each other. Then again they may not be. But, if we had even a rough idea about which English words corresponded with which French ones, then we could start to gather information about distortion probabilities. We might notice, for example, that the first word in an English sentence and the first word in a corresponding French sentence are frequently translations of each other, leading us to hypothesize that d(1 | 1, l, m) is very high. Moreover, when French sentences are much longer than their English versions, we might guess that some of the words in those English sentences tend to have large fertilities. But we're not sure which words exactly.

The idea is that we can gather information incrementally, with each new piece of information helping us to build the next. This is called bootstrapping.

---

22. All Possible Alignments

Alignments will be the key to our bootstrapping algorithm. If we had a single correct alignment for each sentence pair, then we could collect parameter counts from those alignments directly. Now suppose that for one sentence pair, we couldn't decide on the single correct alignment. Suppose that there were two equally good-looking alignments for that sentence pair. Then, for just that one sentence pair, we could decide to collect parameter counts from both alignments. But, since we aren't sure which alignment is correct, we might want to discount the things we collect from that sentence pair, as opposed to other sentence pairs that we are more certain about. For example, any count from one of the two equally good-looking alignments could be multiplied by 0.5. If the word "house" was connected with the word "maison" in only one of those two alignments, then we would pretend that we observed "house" connected to "maison" 0.5 times. That doesn't make much sense -- you can observe something happening once, or twice, or three times, but how can you observe something happening 0.5 times? Suspend your disbelief! Call these fractional counts.

In reality, we may not be able to narrow things down to just one or two possible alignments. In fact, at some point, all alignments may be deemed possible for a given sentence pair.

Exercise. In a sentence pair with l English words and m French words, how many alignments are possible? (Don't forget about the invisible NULL word).

Exercise. What are all the ways to align the two-word English sentence "b c" with the two-word French sentence "x y"? I have drawn some of them below -- fill in the others.

```
b c
| |       alignment1
x y

b c
 /|       alignment2
x y

b c
         alignment3
x y

b c
         alignment4
x y
```

Exercise. Suppose that b = blue, c = house, x = maison, y = bleue. Circle the alignment above that you intuitively prefer.

---

23. Collecting Fractional Counts

If we have two possible alignments for a given sentence pair, we might view them as equally good-looking. On the other hand, we might have cause to think that one is better-looking to the other. In that case, we may assign an alignment weight of 0.2 to one and 0.8 to the other. We can use these weights to build fractional counts. Because the second alignment has a higher weight, it "has more weight to throw around" and will therefore have a bigger voice in the ultimate estimated values for n, t, p, and d.

Here I have attached weights to the alignments we just saw:

```
b c
| |       alignment1 weight = 0.3
x y

b c
 /|       alignment2 weight = 0.2
x y

b c
|\        alignment3 weight = 0.4
x y

b c
 X       alignment4 weight = 0.1
x y
```

Consider estimating the value of n(1 | b), i.e., the chance that English word b generates exactly one French

word. How many times did this occur in the corpus? (This is a funny corpus -- it only has one sentence pair). Using fractional counts, we pretend that this occurred 0.3 + 0.1 times, instead of twice. So, nc(1 | b) = 0.4. Likewise, we observe nc(0 | b) = 0.2, and nc(2 | b) = 0.4. We can compute fertility estimates by normalizing the observed nc counts. In this case, n(1 | b) = 0.4 / 0.4 + 0.2 + 0.4 = 0.4.

Exercise. Use fractional counts to estimate the value of the word translation parameter t(y | b).

With the corpus of many sentences, we do the same thing. We collect fractional counts over all alignments of all sentence pairs. So, given alignment weights, we can estimate n, t, p, and d. I know, you are thinking: where the heck are we going to get alignment weights?

---

24. Alignment Probabilities

Let's ditch the idea of alignment weights in favor of alignment probabilities. That is, we ask for a given sentence pair: what is the probability of the words being aligned in such-and-such a way? For a given sentence pair, the probabilities of the various possible alignments should add to one, as they do in the example given in the previous section.

We'll write P(a | e,f) for the probability of a particular alignment given a particular sentence pair. Remember that an alignment a is just a vector of integers that identify the positions of English words that various French words connect to.

What would make one alignment more probable than other? Well, if someone magically gave us word translation (t) parameter values, then we might use them to judge alignment probabilities. An alignment would be probable to the extent that it connected English and French words that are already known to be high-probability translations of each other. A low probability alignment would be one that connected words with very low t values.

I will now execute a fancy manipulation:

P(a | e,f) = P(a,f | e) / P(f | e)

Exercise. Prove it. Hint: P(a | e,f) = P(a,e,f) / P(e,f) = ? = P(a,f | e) / P(f | e).

So to compute P(a | e,f), we need to be able to compute the ratio of two terms. The first is P(a,f | e). Fortunately, we can compute P(a,f | e) using the generative story of Model 3. Given an English string e, Model 3 marches us through various decision points (fertilities, word translations, permutations, etc.). These decisions determine both an alignment and a resulting French string. Remember that an alignment, by itself, is just a set of positional connectors. (You can have the same alignment and produce two completely different French strings).

The second term is P(f | e). Back at the beginning of section 19, we said we wanted to do two things. One was estimate translation model parameters from data, and the other was to be able to compute P(f | e) given some parameter values. We need the latter for our ultimate goal of French-to-English translation, i.e., choosing a translation e that maximizes P(e) * P(f | e).

We're about to kill two birds with one stone. Consider that there are many ways to produce the same French sentence f from the same English sentence e. Each way corresponds to an alignment that you might draw between them. (You can have two different alignments that produce the same French string). Therefore:

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

So we have reduced both of our problems to computing a formula for P(a,f | e).

---

25. P(a,f | e)

The Model 3 generative story doesn't mention anything about alignments, but an alignment is a way of summarizing the various choices that get made according to the story. 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.

Here's a first cut.

e = English sentence
f = French sentence
ei = the ith English word
fj = the jth French word
l = number of words in the English sentence
m = number of words in the French sentence
a = alignment (vector of integers a1 ... am, where each aj ranges from 0 to l)
aj = the English position connected to by the jth French word in alignment a
eaj = the actual English word connected to by the jth French word in alignment a
phi-i = fertility of English word i (where i ranges from 0 to l), given the alignment a

$$P(a,f \mid e) = \prod_{i=1}^{l} n(\text{phi-}i \mid ei) * \prod_{j=1}^{m} t(fj \mid eaj) * \prod_{j=1}^{m} d(j \mid aj, l, m)$$

This formula accounts for the basic fertility, word translation, and distortion values implied by the alignment and French string. There are a few problems, however.

The first problem is that we should only count distortions which involve French words that were generated by "real" English words, and not by NULL. We should eliminate any d value for which aj = 0.

A related problem is that we forgot to include costs for generating "spurious" French words. Any French word fj for which aj = 0 is spurious. There are phi-0 spurious French words, according to the definitions above. Observe that there are m - phi-0 non-spurious French words, i.e., ones that were generated by "real" English words. After each of those m - phi-0 words, we have the option of generating a spurious word with probability p1. How many ways are there to spit out phi-0 spurious words under this process? That's the same as asking how many ways there are to pull phi-0 balls out of a box that contains (m - phi-0) balls.

For example, there is only one way to generate exactly zero spurious words -- always fail to exercise the option. There are actually m - 1 ways to generate a single spurious word. You could exercise the option after the first "real" word, after the second word, etc., or after the mth word. So we have an additional factor for P(a,f | e), which is:

$$\binom{m - \phi_0}{\phi_0}$$

Let's not forget about the costs of taking each option. If we take the "add spurious" option phi-0 times, then we have an extra factor of p1^phi-0. And if we take the "don't add spurious" option ((m - phi-0) - phi-0) times, then we have an extra factor of p0^(m - 2 * phi-0).

Exercise. Why did I write ((m - phi-0) - phi-0) instead of just (m - phi-0)?

The third problem is that we forgot to include costs for permuting the spurious French words into their final target positions. Recall from section 17 that there are no distortion parameters of the form d(j | 0, l, m). Instead we slot the spurious words into empty French positions as the final step of generating the French string. Since there are phi-0! possible orderings, all of which are deemed equally probable, then any particular ordering we generate adds an additional factor of 1/phi-0! to the computation of P(a,f | e).

A fourth problem is comes up because the alignment loses a bit of information about the generative process that turned e into f. Recall from section 19 that "... if a single English word x is connected to French words y and z, then we don't really know whether they were generated in that order, or whether they were generated as z and y, then permuted." This distinction is lost in an alignment that simply connects x to y and z. It's like if I rolled two dice and reported a result of 4-3, without reporting which die came up 4 and which die came up 3. There are two ways to roll a "4-3." Consider three dice. How many ways are there to roll a "6-4-1"? There are six: two ways if the first die is 6, two ways if the first die is 4, and two ways in the first die is 1.

If English word x is connected to French words y, z, and w, then there are six ways this could have happened according to the Model 3 generative story. Notice that all six ways are equally probable, because the factors that get multiplied are always the same: n(3 | x), t(y | x), t(z | x), t(w | x), d(1 | 1,1,3), d(2 | 1,1,3), d(3 | 1,1,3), p0. Multiplication doesn't care which order these factors come in. So in this case, P(a,f | e) = 6 * all those factors.

This fourth problem arises exactly when any word has fertility greater than 1. So we will add a final factor to our P(a,f | e) formula, namely:

$$\prod_{i=0}^{l} \phi_i!$$

Here's our great new formula!

$$P(a,f \mid e) =$$

$$\binom{m - \phi_0}{\phi_0} * p_0^{(m - 2\phi_0)} * p_1^{\phi_0} *$$

$$\prod_{i=1}^{l} n(\phi_i \mid e_i) * \prod_{j=1}^{m} t(f_j \mid e_{a_j}) * \prod_{j:a_j \neq 0}^{m} d(j \mid a_j, l, m) *$$

$$\prod_{i=0}^{l} \phi_i! * (1 / \phi_0!)$$

Recall that we can express P(f | e) in terms of P(a,f | e):

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

Recall also that we can express P(a | f,e) in terms of P(a,f | e).

$$P(a \mid e,f) = P(a,f \mid e) \,/\, \sum_{a} P(a,f \mid e)$$

So that's the whole Model 3 story boiled down to a math formula. If you can come up with a better story and boil it down to a math formula, then maybe you can get better translations!

---

26. Chicken and Egg

The point of section 23 was this: if you have alignment probabilities, then you can compute parameter estimates (using fractional counts). The point of section 25 was this: if you have parameter values, then you can compute alignment probabilities. Uh oh, what went wrong?

Our big goal of being able to train P(f | e) from bilingual text seems just out of reach. To compute P(f | e), we need to compute P(a,f | e), and to do that we need parameter values. We know how to get parameter values -- we just need to be able to invoke P(a | f,e). To do that, we need P(a,f | e), and we're stuck in a circle.

Like they say, and egg is a chicken's way of making more chickens. (Or something like that?)

---

27. Now for the Magic

The EM algorithm can solve our problem. "EM" stands for "estimation-maximization." Some people claim it stands for "expectation-maximization." That reminds me of the time that I couldn't remember how to spell a certain word: was it "intractibility" or "intractability"? I queried AltaVista on both, and the latter won out 785-22. I'm not sure what my point is, but anyway, the EM algorithm can solve our problem!

We will begin with uniform parameter values. That is, suppose there are 40,000 words in the French vocabulary. Then we begin with the assumption that t(f | e) = 1/40,000 for every pair of English and French words. Maybe "house" translates to "maison," or maybe translates to "internationale." At this point, we have no idea.

Exercise. With uniform distortion probabilities, what is the chance that an English word in position 4 will produce a French word that winds up in position 7, assuming both sentences are 10 words long? I.e., d(7 | 4, 10, 10) = ?

We can pick a random value for p1, say 0.15, and we can assign every English word the same set of fertility probabilities.

Now that we have some values, we can compute alignment probabilities for each pair of sentences. That

means we can collect fractional counts. When we normalize the fractional counts, we have a revised set of parameter values. Hopefully these will be better, as they take into account correlation data in the bilingual corpus. Armed with these better parameter values, we can again compute (new) alignment probabilities. And from these, we can get a set of even-more-revised parameter values. If we do this over and over, we may converge on some good values.

Let's see how this works in practice. We'll illustrate things with a very simplified scenario. Our corpus will contain two sentence pairs. The first sentence pair is bc/xy (a two-word English sentence "b c" paired with a two-word French sentence "x y"). The second sentence pair is b/y (each sentence contains only one word).

Exercise. What would you say to me if I claimed that "every time the English word c appears, the French word y also appears, so they are translations of each other"?

We are also going to forget about the NULL word, require every word to have only fertility 1, and forget about distortion probabilities. In that case, the sentence pair bc/xy has only two alignments:

Exercise. Which two alignments are we ignoring for the purposes of this scenario?

The sentence pair b/y only has one alignment:

Given our simplifications, the only things that bear on the alignment probabilities are the word translation (t) parameter values. So we'll pare down our P(a,f | e) formula:

$$P(a,f \mid e) = \prod_{j=1}^{m} t(fj \mid eaj)$$

This is actually the P(a,f | e) formula for Model 1. (Although, unlike this scenario, Model 1 generally works with NULL and all possible alignments).

Okay, here goes EM:

Step 1. Set parameter values uniformly.

t(x | b) = 1/2
t(y | b) = 1/2
t(x | c) = 1/2
t(y | c) = 1/2

Step 2. Compute P(a,f | e) for all alignments.

```
| |    P(a,f | e) = 1/2 * 1/2 = 1/4
x y

b c
 X    P(a,f | e) = 1/2 * 1/2 = 1/4
x y

b
|     P(a,f | e) = 1/2
y
```

Step 3. Normalize P(a,f | e) values to yield P(a | e,f) values.

```
b c
| |    P(a | e,f) = 1/4 / 2/4 = 1/2
x y

b c
 X    P(a | e,f) = 1/4 / 2/4 = 1/2
x y

b
|     P(a | e,f) = 1/2 / 1/2 = 1       ;; there's only one alignment, so P(a | e,f) will be 1 always.
y
```

Step 4. Collect fractional counts.

```
tc(x | b) = 1/2
tc(y | b) = 1/2 + 1 = 3/2
tc(x | c) = 1/2
tc(y | c) = 1/2
```

Step 5. Normalize fractional counts to get revised parameter values.

```
t(x | b) = 1/2 / 4/2 = 1/4
t(y | b) = 3/2 / 4/2 = 3/4
t(x | c) = 1/2 / 1 = 1/2
t(y | c) = 1/2 / 1 = 1/2
```

Repeat Step 2. Compute P(a,f | e) for all alignments.

```
b c
| |    P(a,f | e) = 1/4 * 1/2 = 1/8
x y

b c
 X    P(a,f | e) = 3/4 * 1/2 = 3/8
x y

b
|     P(a,f | e) = 3/4
```

y

Repeat Step 3. Normalize P(a,f | e) values to yield P(a | e,f) values.

b c
| | P(a | e,f) = 1/4
x y

b c
X P(a | e,f) = 1/4 / 2/4 = 3/4
x y

b
| P(a | e,f) = 1
y

Repeat Step 4. Collect fractional counts.

tc(x | b) = 1/4
tc(y | b) = 3/4 + 1 = 7/4
tc(x | c) = 3/4
tc(y | c) = 1/4

Repeat Step 5. Normalize fractional counts to get revised parameter values.

t(x | b) = 1/8
t(y | b) = 7/8
t(x | c) = 3/4
t(y | c) = 1/4

Repeating steps 2-5 many times yields:

t(x | b) = 0.0001
t(y | b) = 0.9999
t(x | c) = 0.9999
t(y | c) = 0.0001

What happened? The alignment probability for the "crossing alignment" (where b connects to y) got a boost from the second sentence pair b/y. That further solidified t(y | b), but as a side effect also boosted t(x | c), because x connects to c in that same "crossing alignment." The effect of boosting t(x | c) necessarily means downgrading t(y | c) because they sum to one. So, even though y and c co-occur, analysis reveals that they are not translations of each other.

EM for Model 3 is just like this, except we use Model 3's formula for P(a | f,e), and we additionally collect fractional counts for n, p, and d parameters from the weighted alignments.

Exercise. Carry out two or three iterations of Model 1 EM training on the following sentence pairs:

the blue house <-> la maison bleue
the house <-> la maison

Just consider six possible alignments for the first sentence pair, and two possible alignments for the second sentence pair. Here is a start:

t(la | the) = 1/3
t(maison | the) = 1/3
t(bleue | the) = 1/3
t(la | blue) = 1/3
t(maison | blue) = 1/3
t(bleue | blue) = 1/3
t(la | house) = 1/3
t(maison | house) = 1/3
t(bleue | house) = 1/3

```
the blue house
|     |     |       P(a,f | e) = 1/3 * 1/3 * 1/3 = 1/27
la maison bleue

the   blue house
|       X           P(a,f | e) = 1/3 * 1/3 * 1/3 = 1/27
la maison bleue
```

...

---

28. What is EM doing?

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 section 12. 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 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. Where EM winds up is therefore partially a function of where it starts.

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

$$2^{-\log(P(f \mid e))/N}$$

Each EM iteration lowers the perplexity.

Imagine the following gambling game. I show you an English sentence e, and I sequester a certain French translation f in my pocket. Now I give you $100 to bet on various French sentences. You can spread the

$100 as you see fit. You can bet it all on one French sentence, or you can bet fractions of pennies on many French sentences. Spreading your bet is like spreading the unit mass of probability P(f | e) across many sentences f.

Then I reveal the French translation from my pocket. We see how much you bet on that sentence. If you bet $100, you'll get a nice payoff. Even if you bet a dime, you'll get a nice payoff. But if you bet nothing, then you lose all your money. Because in that case, you set P(f | e) = 0, and you made a big mistake.

At each iteration of the EM algorithm, the computer can play this gambling game better and better. That's what it learns to do. That's all it learns to do.

Exercise. If you were very good at this game, do you think you would be a good translator? If you were a good translator, would you be good at this game?

Model 3 has at least two problems with this game. You should know about them. The first one is simple. The distortion parameters are a very weak description of word-order change in translation. Words just fly around all over the place. So Model 3 will bet on lots of ungrammatical French sentences. We mentioned above that P(e) will take up some of the slack. The second problem is that Model 3 also lays bets on French sentences that you wouldn't even recognize as natural language at all. In the generative model, remember that we choose a target position slot for each French word (the permutation step). Nothing in the generative story prevents us from selecting target position 7 for every French word, in which case they will all pile up on top of each other, leaving the other target positions empty. If you saw a sentence like that, you wouldn't be able to read it, because it would look like the lot of white space surrounding a big blob of ink. Because Model 3 lays bets on such non-strings, better models will look down on it and called it deficient. Don't lose any sleep over this, though. We're not going to use Model 3 to translate English into French. Plus, if we happened to see a French sentence with words piled on top of each other, we would be able to deal with it -- most machine translation algorithms would just freak out, don't you think?

---

29. Decoding

We can learn parameters values for computing P(e) from monolingual English text. We can learn parameter values for computing P(f | e) from bilingual sentence pairs. To translate an observed French sentence f, we seek the English sentence e which maximizes the product of those two terms. This process is called decoding. It's impossible to search through all possible sentences, but it is possible to inspect a highly relevant subset of such sentences. We won't go into the details of this heuristic search in this workbook. IBM threatened to describe decoding in a “forthcoming paper” which never came forth. It is described in their US patent.

---

30. Practical Problems with Model 3 Training

There are a couple of serious problems with the training algorithm we presented above. First, EM does not find a globally best solution. In practice, for example, it may seize upon imaginary regularities in word distortion. It may decide that the best way to improve P(f | e) is to always link the first two respective words in any sentence pair. In that case, distortion costs will be small, and word-translation costs will be large, because a given English word will have very many possible translations. Linguistically, it may make more sense to have fewer word translations and higher distortion costs, but the EM algorithm does not know anything about linguistics. To solve this problem, we may want to hold the distortion probabilities at

uniform values until we update our translation probabilities through a few EM iterations.

The second problem is more serious. As described above, the EM algorithm needs to iterate over every alignment of every sentence pair. If a sentence pair contains 20 words of English and 20 words of French, then there are too many alignments to consider. (How many?). There's no way to iterate over all of them, even for just this one sentence pair.

The solution is to iterate only over a subset of "good-looking" (probable) alignments. For example, we could collect fractional counts only over the best 100 alignments for each sentence pair. That brings up two questions. First, how to find the best 100 alignments without enumerating them all. And second, how to get the thing off the ground. At the first iteration, remember that the parameter values are set uniformly. That means any alignment is just as probable as any other.

Both problems can be addressed if we have a quick-and-dirty way of initializing parameter values. In fact, Model 1 provides such a way. It turns out that we can collect Model 1 fractional t counts from all possible alignments without having to enumerate all possible alignments.

---

31. Efficient Model 1 Training

Recall that training involves collecting fractional counts that are weighted by P(a | e,f). Also recall:

$$P(a \mid e,f) = P(a,f \mid e) / P(f \mid e) = P(a,f \mid e) / \sum_{a} P(a,f \mid e)$$

Consider the computation of the denominator above. Using Model 1, we can write it as:

$$\sum_{a} \prod_{j=1}^{m} t(fj \mid eaj)$$

Since there are (l+1)^m possible alignments, we have to do m * (l+1)^m arithmetic operations to evaluate this expression. For clarity, let's actually write it out in terms of t parameter values:

```
t(f1 | e0) * t(f2 | e0) * ... * t(fm | e0) +
t(f1 | e0) * t(f2 | e0) * ... * t(fm | e1) +
t(f1 | e0) * t(f2 | e0) * ... * t(fm | e2) +
  ...
t(f1 | e0) * t(f2 | e1) * ... * t(fm | e0) +
t(f1 | e0) * t(f2 | e1) * ... * t(fm | e1) +
t(f1 | e0) * t(f2 | e1) * ... * t(fm | e2) +
  ...
t(f1 | el) * t(f2 | el) * ... * t(fm | e0) +
t(f1 | el) * t(f2 | el) * ... * t(fm | e1) +
t(f1 | el) * t(f2 | el) * ... * t(fm | e2) +
  ...
t(f1 | el) * t(f2 | el) * ... * t(fm | el)
```

Each row here corresponds to a particular alignment, and each column corresponds to a French word. In the first row, all French words are connected to e0 (NULL). The second row is the same except that the final

French word fm is connected to e1 instead of e0. The last row represents the alignment in which all French words are connected to the sentence-final English word el. Now you can clearly see that there are m * (l+1)^m arithmetic operations required to compute P(f | e).

But we can take advantage of certain regularities in the above expression. For example, we can factor out t(f1 | e0) from all of the rows (alignments) that contain it, yielding:

```
t(f1 | e0) * [ t(f2 | e0) * ... * t(fm | e0) +
               t(f2 | e0) * ... * t(fm | e1) +
               t(f2 | e0) * ... * t(fm | e2) +
                 ...
               t(f2 | e1) * ... * t(fm | e0) +
               t(f2 | e1) * ... * t(fm | e1) +
               t(f2 | e1) * ... * t(fm | e2) +
                ... ]
t(f1 | el) * t(f2 | el) * ... * t(fm | e0) +
t(f1 | el) * t(f2 | el) * ... * t(fm | e1) +
t(f1 | el) * t(f2 | el) * ... * t(fm | e2) +
   ...
t(f1 | el) * t(f2 | el) * ... * t(fm | el)
```

We just saved a bunch of multiplications. It's like the difference between “xy+xz” (3 operations) and “x(y+z)” (2 operations). If we continue with this style of factoring out, we get:

```
[ t(f1 | e0) + t(f1 | e1) + ... + t(f1 | el) ] *
[ t(f2 | e0) + t(f2 | e1) + ... + t(f2 | el) ] *
   ...
[ t(fm | e0) + t(fm | e1) + ... + t(fm | el) ]
```

Exercise. Verify that this expression is equal to the expressions above.

In more compact notation:

$$\sum_{a} \prod_{j=1}^{m} t(f_j \mid e_{a_j}) = \prod_{j=1}^{m} \sum_{i=0}^{l} t(f_j \mid e_i)$$

This last expression only requires a quadratic number of arithmetic operations to evaluate (to be exact: l+1 * m operations). That means we can efficiently compute P(f | e) for Model 1. It is not hard to figure out how to collect fractional counts during EM training efficiently as well.

---

32. Best Alignment for Model 1

Model 1 has another advantage. It is easy to figure out what the best alignment is for a pair of sentences (the one with the highest P(a | e,f) or P(a,f | e)). We can do this without iterating over all the alignments.

Remember that P(a | e,f) is computed in terms of P(a,f | e). In Model 1, P(a,f | e) has m factors, one for each French word. Each factor looks like this: t(fj | eaj). Suppose that the French word f4 would rather connect to e5 that e6, i.e., t(f4 | e5) > t(f4 | e6). That means if we are given two alignments which are identical

except for the choice of connection for f4, then we should prefer the one that connects f4 to e5 -- no matter what else is going on in the alignments. So we can actually choose an English "home" for each French word independently.

for $1 <= j <= m$,

$$a_j = \operatorname{argmax}_i t(f_j \mid e_i)$$

That's the best alignment, and it can be computed in a quadratic number of operations (l+1 * m).

Exercise. Find the best alignment for the sentence pair "bcd/xy." First enumerate all nine alignments (forget about NULL), and evaluate P(a | e,f) for each separately. Then use the quadratic method above. Verify that the results are the same. Use the following parameter values: t(x | b) = 0.7, t(y | b) = 0.3, t(x | c) = 0.4, t(y | c) = 0.6, t(x | d) = 0.9, t(y | d) = 0.1.

Unfortunately, the "best" alignment according to a fully-trained Model 1 may not be very intuitively satisfying. Because each French word chooses its preferred English "home" independently, they may very well all choose English word e5. According to the Model 1 generative story (which I didn't write down for you), that means e5 generates every French word.

---

33. Back to Model 3

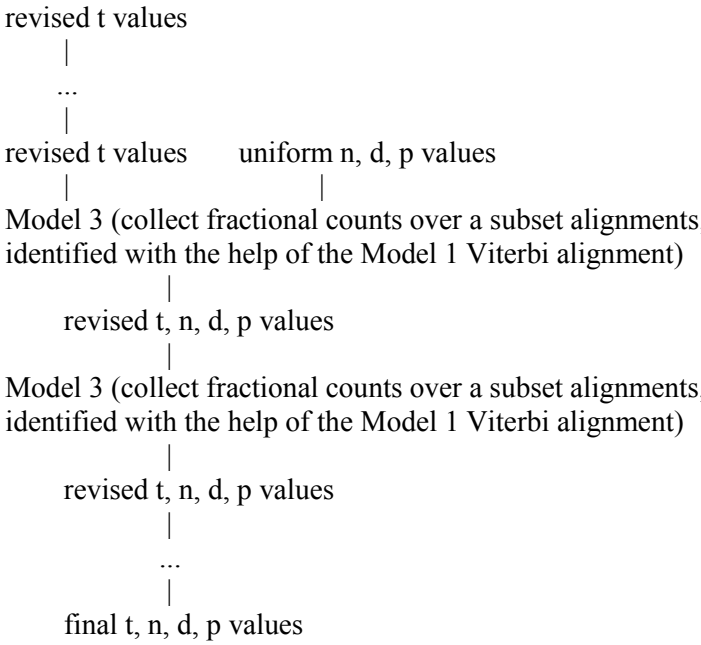
Our linguistic intuitions tell us that e5 generating every French word is a bit unlikely. In fact, Model 3 can penalize such an alignment by using its fertility probabilities. If it is unlikely that e5 has a huge fertility, and it is unlikely that the rest of the words all coincidentally have zero fertility, then such alignment will look bad. Model 1 has no way to achieve this.

Now we can see that the "fertility" idea is the result of linguistic intuitions. These intuitions must be tested, of course. It may turn out that in real translation, fertilities are pretty random. We can always compare Model 1 to Model 3 by inspecting their "best" alignments and seeing which are more intuitive. We can also compare models by computing their respective perplexities (section 28).

Fertilities might be good for modeling translation, but they're not good for efficient computation. We cannot compute the best Model 3 alignment using the trick from section 32, because French words cannot choose their "homes" independently. If two words choose the same home, that may change the whole P(a,f | e) value. Furthermore, in computing P(f | e), we must iterate over all alignments. We cannot do the factoring trick from section 31. All is not bleak: we can still compute P(a,f | e) for a specific alignment fairly quickly, using the formula in section 25.

Here is a practical plan for Model 3 training. First I'll show a diagram, and then explain it.

uniform t values
|
Model 1 iteration (collect fractional counts over all alignments)
|
revised t values
|
Model 1 iteration (collect fractional counts over all alignments)
|



Okay: we first train Model 1 for several iterations. When that is done, we find the best alignment (also called the Viterbi alignment) for each sentence pair, according to Model 1 parameter values. For each sentence pair, we then re-score the Model 1 Viterbi alignment using the P(a,f | e) formula of Model 3. (If this is the first Model 3 iteration, then we will have to use uniform fertility and distortion probabilities).

Once we have a somewhat reasonable alignment and a Model 3 score, we can consider small changes to the alignment. (For example, changing the connection of f4 from e5 to e7). If a small change improves the Model 3 score, we make that change. We keep changing the alignment (hill climbing) until there is no simple way to improve it anymore. We'll call this the "Viterbi" alignment for Model 3, even though we don't know for a fact that this is the best alignment according to the P(a | e,f) formula of Model 3.

Now we consider this Model 3 Viterbi alignment together with its neighborhood -- those alignments that look very similar to it (are obtainable from very small changes). This gives us a reasonable subset of Model 3 alignments. We can collect fractional counts only over that subset. When we normalize these counts, we have new values of t, n, d, and p. We can then repeat the process. Again we find the best alignments according to Model 1 and the t values, and we use these alignments to greedily find best Model 3 alignments. Notice that our new values for t feed back into Model 1, and may actually yield new Model 1 Viterbi alignments.

If we want a larger variety of good-looking alignments, we can start up several hill climbers -- each begins at a variant of the Model 1 Viterbi alignment in which a certain connection is manually set and pegged for the duration of the climb.

---

34. Model 2

In the end, we're going to do something slightly different and somewhat more interesting. You knew that there was a Model 2, right? This model involves word translations (like Model 1) but also distortions. As with Model 1, we can find Viterbi alignments and collect fractional counts efficiently. The distortion probabilities in Model 2 are different from the ones in Model 3. Instead of

d(French-position | English-position, English-sentence-length, French-sentence-length)

We will use reverse-distortion probabilities:

a(English-position | French-position, English-sentence-length, French-sentence-length)

I apologize profusely for using the notation "a" here. I realize that "a" usually stands for an alignment. Dreadfully, dreadfully sorry. If there's any way I can make it to you, please let me know. Here's the P(a,f | e) formula for Model 2:

$$P(a,f \mid e) = \prod_{j=1}^{m} t(fj \mid eaj) \prod_{j=1}^{m} a(aj \mid j,l,m)$$

This model might penalize an alignment that connects positions that are very far apart, even if the connected words are good translations of each other. Or during training, it might prefer to connect up words on the basis of position, which might actually be good if the words in question are rare (not enough co-occurrences to establish reliable word-translation probabilities).

To find best Model 2 alignment, we select for each French word fj (independently) the English position i that maximizes the word-translation and distortion factors:

$$\text{for } 1 <= j <= m, \quad aj = \operatorname*{argmax}_{i} t(fj \mid ei) * a(i \mid j,l,m)$$

To compute P(f | e) efficiently, we use factoring-out to do the following rewrite:

$$\sum_{a} \prod_{j=1}^{m} t(fj \mid eaj) * a(aj \mid j,l,m) = \prod_{j=1}^{m} \sum_{i=0}^{l} t(fj \mid ei) * a(aj \mid j,l,m)$$

Exercise. Convince yourself that this rewriting is correct.

It is not hard to extend this reasoning to algorithm that collects fractional counts efficiently over sentence pairs.

---

35. Transferring Parameter Values from One Model to Another

We could apply Model 2 directly to the sentence pair corpus, bypassing Model 1. As I mentioned earlier, it might be better to settle on some reasonable word-translation probabilities before bringing distortions into the picture. So we will run Model 1 for several iterations, and then use the resulting "t" values as input to the first Model 2 iteration, instead of uniform t values. This simple idea is called transferring parameter values from one model to another. For the first Model 2 iteration, we will use uniform "a" values.

It's more interesting when we transfer parameter values from Model 2 to Model 3. We would like to use everything we learned in Model 2 iterations to set initial Model 3 parameter values. While we could set up a uniform distortion table d, it is fairly easy to set up better values. We can insert a "one time only" special

transfer iteration in between Model 2 iterations and Model 3 iterations. During this transfer, we can do quadratic-time count collection, but in addition to collecting t and a counts, we can collect d counts as well.

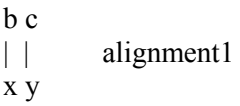
How about fertilities? We could use some uniform (or other simple) initialization, but again, we would like to use what we have learned in Model 2 to set up better initial values. Imagine that Model 2 identified some very highly probable alignments. In that case, we could inspect these alignments to get a reasonable first idea about the fertility of a word like "house." If, according to Model 2, "house" tends to connect to a single French word, then we can conclude that it tends to have fertility one. Subsequent Model 3 iterations may modify this conclusion, of course.

Unfortunately, there seems to be no quadratic-time algorithm for turning t and a knowledge into fertility (n) knowledge. We must give up on the idea of iterating over all alignments, so that option is out. But there is a middle ground.

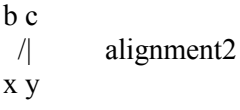
For purposes of explanation, let's forget about the "a" (reverse distortion) parameters, and just compute initial fertility parameter values from previously-learned Model 1 (word-translation) "t" parameter values. Consider a corpus with the sentence pair bc/xy, and suppose that Model 1 has learned the following:

t(x | b) = 0.8
t(y | b) = 0.2
t(x | c) = 0.2
t(y | c) = 0.8

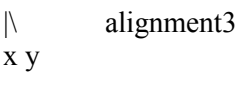
Let's entertain the following four alignment possibilities:



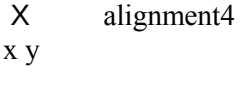
alignment1



alignment2



alignment3



alignment4

Now we want to guess initial values for the fertility (n) table. We will look at four methods.

Method One. Assign uniform probabilities. We notice from the given alignments that the word b may have a fertility of 0, 1, or 2. We therefore assign n(0 | b) = n(1 | b) = n(2 | b) = 1/3.

Method Two. Collect fractional counts of fertilities over all alignments, but pretend that each alignment is equally likely. Of the four alignments, only alignment2 shows a zero fertility for word b, so n(0 | b) = 1/4. This method can be applied efficiently to very long sentence pairs.

Exercise. Using this method, what are the values for n(1 | b) and n(2 | b)? Do the results make sense to you?

Method Three. Collect fractional counts of fertilities over all alignments, but take word-translation probabilities into account. This is like normal EM training. For each alignment, we compute P(a | e,f) and use this value for weighting counts that we collect. Remember that P(a | e,f) = P(a,f | e) / sum-over-a P(a,f | e), and for Model 1, P(a,f | e) is the product of all the word-translation probabilities indicated by the alignment.

P(alignment1 | e,f) = 0.8 * 0.8 -> normalized -> 0.64
P(alignment2 | e,f) = 0.2 * 0.8 -> normalized -> 0.16
P(alignment3 | e,f) = 0.8 * 0.2 -> normalized -> 0.16

Exercise. What is P(alignment4 | e,f)?

Now we collect fractional counts. Alignment2 is the only one in which we observe b to have fertility zero. So we collect a fertility count for zero with weight 0.16:

nc(0 | b) = 0.16

Alignments 1 and 4 both show b with fertility one. So we add the two fractional counts:

nc(1 | b) = 0.64 + 0.04 = 0.68
nc(2 | b) = 0.16

In this case, the values of the nc counts happen to already add to one, so they remain as they are when normalized: n(0 | b) = 0.16, n(1 | b) = 0.68, n(2 | b) = 0.16. The distribution 0.16/0.68/0.16 is better than those obtained by Methods One and Two, because it takes into account what Model 1 has learned, i.e., that alignment1 is very probable, and hence both words probably have fertility one.

Method Four. We know that Method Three does not scale to long sentences, because we cannot enumerate alignments. But we can get exactly the same results as Method Three with a computationally cheaper algorithm:

l = length of English sentence
m = length of French sentence
ei = ith English word
fj = jth French word
t(f | e) = probability English word e translates to French word f
phi-max = maximum fertility of any word
big-gamma(n) = partition of n into a sum of integers all >= 1
  (e.g., 1+2+4 is a partition of 7, and so is 1+1+1+1+3)
times(partition, k) = number of times integer k appears in partition

```
for i = 1 to l
  for k = 1 to phi-max
    beta = 0.0
    for j = 1 to m
      beta += [ t(fj | ei) / (1 - t(fj | ei) ]^k
    alpha[i,k] = (-1)^(k+1) * beta / k
for i = 1 to l
  r = 1.0
  for j = 1 to m
```

```
  r = r * (1 - t(fj | ei))
for phi = 0 to phi-max
  sum = 0.0
  for each partition p in big-gamma(phi)
    prod = 1.0
    for k = 1 to phi
      prod = prod * alpha[i,k]^times(p,k) / times(p,k)!
    sum += prod
  c(phi | ei) += r * sum
```

You can verify by hand that this algorithm yields the same results on the example above as Method Three does. You can make the algorithm work for Model 2 by multiplying in an a(i | j,l,m) factor next to every t(fj | ei) expression above. The proof of Method Four can be found after (Brown et al, 1993)'s equation 108. Note that this equation has an error (please remove the factor of k! from the computation of alpha). Finally, there is a nice way to march through the partitions of an integer in linear time, without allocating space for each one. Almost as though I were a hip youngster, I found some Pascal code on the World Wide Web to do that.

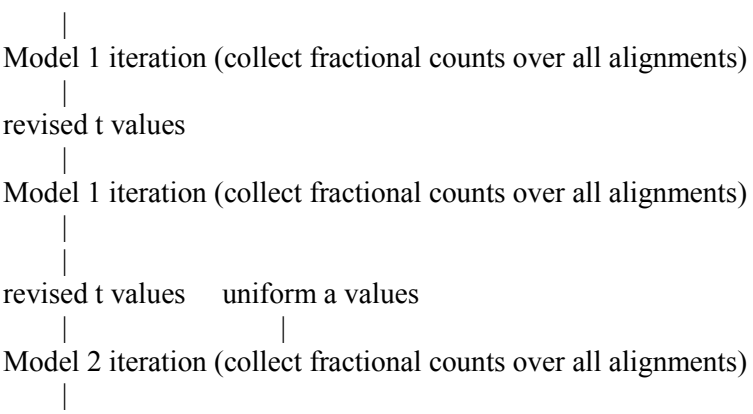
You may be wondering: if we can compute fertility counts without having to iterate over all alignments, then haven't we solved the Model 3 training problem, without heuristic recourse to a "small subset" of alignments? Not really. The method above takes t and a values as input and produces fertility values. If we try to iterate this same method, then we would have to throw away the fertility values -- the ones we just learned! The only inputs are t and a. The method cannot accept one set of fertility values and then revise them, which is what we want from Model 3 training.

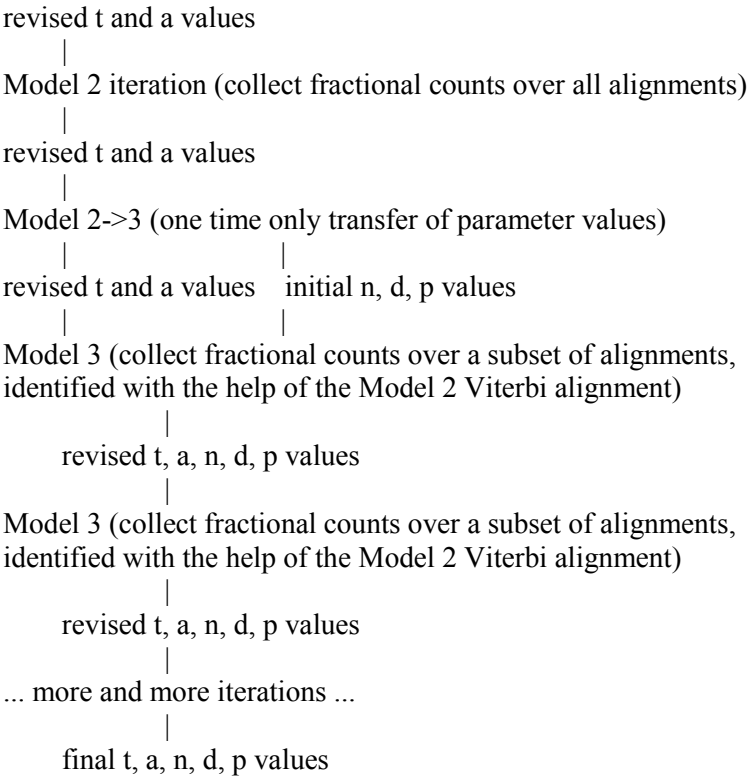
It is possible to use a similar method to determine an initial value for the parameter p1, but I suggest you just assign it a value and wing it.

---

36. The Final Training Scheme

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:





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.

---

37. Wow

With this prime-numbered section, we draw a close to the proceedings. If you made it all the way, then I'll buy you a beer!