Word Alignment and the Expectation-Maximization Algorithm

Adam Lopez
University of Edinburgh

The purpose of this tutorial is to give you an example of how to take a simple discrete probabilistic model and derive the expectation maximization updates for it and then turn them into code. We give some examples and identify the key ideas that make the algorithms work. These are meant to be as intuitive as possible, but for those curious about the underlying mathematics, we also provide some derivations, and point the reader to additional tutorials when mathematical depth goes beyond the scope of the tutorial. But you needn't follow these derivations in detail to understand the main concepts, so whether you try to is to is up to you.

We'll consider IBM Model 1, which has been the subject of other similar tutorials [Collins(2011), Knight(1999)]. A great deal of the material here is inspired, either directly or indirectly, by these papers, as well as the concise original description in Sections 4 and 4.1 of [Brown et al.(1993)], and of course, Chapter 4 of Philipp Koehn's textbook (2010) . Everyone learns differently, though, so I urge you to take a look at one of those if you're still confused after reading this one.

The basic plan is to learn a conditional probabilistic model of a French sentence $\mathbf{f}$ given an English sentence $\mathbf{e}$, which we'll denote $p_\theta(\mathbf{f}|\mathbf{e})$. Subscript θ refers to the set of parameters in the model; we'll describe what it looks like shortly. We also have a dataset $\mathcal{D}$ of N sentence pairs that are known to be translations of each other, $\mathcal{D} = \{(\mathbf{f}^{(1)}, \mathbf{e}^{(1)})...(\mathbf{f}^{(N)}, \mathbf{e}^{(N)})\}$, where each superscript (n) indexes a different sentence pair. The object of our model will be to uncover the hidden word-to-word correspondences in these translation pairs. We will learn the model from this data, and once we've learned it, we'll use it to predict the existence of the missing word alignments.

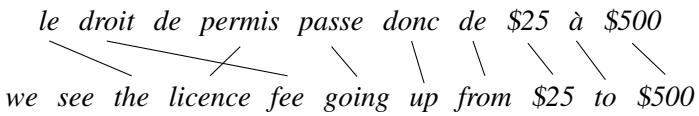
1 IBM Model 1

There are many ways we could define $p(\mathbf{f}|\mathbf{e})$. A very simple but natural model is one based on *lexical translation*—that is, word-to-word translation. IBM Model 1 assumes that each word in the French sentence is a translation of exactly zero or one word of the English sentence (Note that we make no such assumption for each English word!) Everything else about the model is chosen to be as simple as possible so that we can focus single-mindedly on this one idea.

Since sentences are sequences of words, we'll want a notation to refer to each word. Let a French sentence $\mathbf{f}$ be represented by an array of I words, $\langle f_1, ..., f_I \rangle$, and let an English sentence $\mathbf{e}$ be represented by an array of J words, $\langle e_1, ..., e_J \rangle$. Now, since we made the assumption that each French word aligned to exactly one English word, we can represent an alignment of the French words by an array $\mathbf{a}$ of length I, denoted $\langle a_1, ..., a_I \rangle$. Alignment variable a_i takes a value in the range $[0, J]$ denoting the index of the English word to which French word f_i is aligned. If $a_i = 0$, this means that f_i is not aligned to any word in the English sentence, called *null alignment*. Allowing French words to align to an imaginary *null word* enables us to model French words that have no corresponding English word. As a practical matter, you can

simulate this by simply pretending that each English sentence has an extra word in it at position 0, and treat it like any other English word.[1]

To make this concrete, consider the sentence pair $\langle \mathbf{f}, \mathbf{e} \rangle = \langle$*le droit de permis passe donc de $25 à $500, we see the licence fee going up from $25 to $500*$\rangle$. We have $I = 10$ and $J = 11$, $f_1 =$*le*, $f_2 =$*droit*, $f_3 =$*de*, and so on for the French sentence, and $e_1 =$*we*, $e_2 =$*see*, $e_3 =$*the*, and so on for the English sentence. Now suppose that we observe this alignment:



In this case, the value of $\mathbf{a}$ will be $\langle 3, 5, 0, 4, 6, 7, 8, 9, 10, 11 \rangle$. With this notation, we imagine that our probabilistic model generates the French sentence from the English using a simple procedure. First, the length I is chosen according to a distribution $p(I|J)$, in this case $p(10|11)$. Then, each French word position aligns to an English word (or null) according to a uniform distribution $p(a_i = j|J) = \frac{1}{J+1}$, in this case $\frac{1}{11}$. Finally, each French word f_i is translated according to a distribution conditioned on the aligned English word, $p(f_i|e_{a_i})$. So for this alignment we multiply $p($*le*$|$*the*$)$, $p($*droit*$|$*fee*$)$, $p($*de*$|$null$)$, $p($*permis*$|$*license*$)$, and so on. The joint probability of the French sentence and its alignment conditioned on the English is simply the product of all these probabilities.

$$p(\mathbf{f}, \mathbf{a}|\mathbf{e}) = p(I|J) \prod_{i=1}^{I} p(a_i|J) \cdot p(f_i|e_{a_i}) \tag{1}$$

Now we have a definition of θ: It's simply two tables of numbers: $p(I|J)$, for all pairs of sentence lengths I and J; and $p(f|e)$ for all pairs of co-occurring French and English words f and e. Since these numbers represent probabilities, the set of valid assignments of numbers to these tables must follow basic rules of probability.

$$\forall_{e,f} p(f|e) \in [0, 1]. \tag{2}$$

$$\forall_e \sum_f p(f|e) = 1. \tag{3}$$

We're not going to worry too much about $p(I|J)$ because the sentence lengths are observed, and in any case we don't need to worry about these parameters if the goal is simply to align sentences. We do however care about $p(f|e)$, which seems difficult to estimate because we haven't observed the alignments.

2 Maximum Likelihood Estimation

Before proceeding, let's think about how you would estimate $p(f|e)$ if you actually *had* observed the alignments of the sentences. To take an example, suppose that you only ever saw the English word *fee* aligned to two different French words: three times to *droit*, and seven times to *frais*. Intuitively, you might decide that a good setting for θ in this case is to let $p($*droit*$|$*fee*$) = 0.3$ and $p($*frais*$|$*fee*$) = 0.7$, since these values

[1] There are many different models for null words, but this is simple. Even simpler is to leave out nulls altogether.

correspond to the proportions in the data that you observed. You can calculate them just by adding up and dividing counts:

$$p(\textit{droit}|\textit{fee}) = \frac{count(\langle \textit{droit}, \textit{fee} \rangle)}{count(\langle \textit{droit}, \textit{fee} \rangle) + count(\langle \textit{frais}, \textit{fee} \rangle)} = \frac{3}{10} = 0.3 \tag{4}$$

$$p(\textit{frais}|\textit{fee}) = \frac{count(\langle \textit{frais}, \textit{fee} \rangle)}{count(\langle \textit{droit}, \textit{fee} \rangle) + count(\langle \textit{frais}, \textit{fee} \rangle)} = \frac{7}{10} = 0.7 \tag{5}$$

Why is this reasonable? To understand this, we'll introduce something called the *likelihood* function. It's simply the probability of the data given the parameters.

$$\mathcal{L}(\theta|\mathcal{D}) = p_\theta(\mathcal{D}|\theta) = \prod_{n=1}^{N} p_\theta(\mathbf{f}^{(n)}, \mathbf{a}^{(n)}|\mathbf{e}^{(n)}) = \prod_{n=1}^{N} p(I^{(n)}|J^{(n)}) \prod_{i=1}^{I^{(n)}} p(a_i^{(n)}|J^{(n)}) \cdot p(f_i^{(n)}|e_{a_i}^{(n)}) \tag{6}$$

Now, the data is observed, and the parameters are what we want to estimate, so this is really a function from parameters to a probability. It's reasonable to assume that the data are highly probable under the model, so one possible strategy (there are many!) is to choose parameters $\hat{\theta}$ that maximize the probability.

$$\hat{\theta} = \arg\max_\theta \prod_{n=1}^{N} p_\theta(\mathbf{f}^{(n)}, \mathbf{a}^{(n)}|\mathbf{e}^{(n)}) \tag{7}$$

Whenever you see $\arg\max$ like in this formulation, you're looking at a *search* problem: there are an infinite number of possible assignments for θ (since the range of each parameter is continuous), so we're going to have to search for the one that maximizes this expression. Search problems are ubiquitous in artificial intelligence, and machine translation is no exception. For this case, though, the search is trivial, because there is a closed-form solution for $\hat{\theta}$ when the data described by our model is fully observed.

Let's return to our intuitive guess about the parameters $p(\textit{droit}|\textit{fee})$ and $p(\textit{frais}|\textit{fee})$. It turns out that this is the maximum likelihood solution! As long as you understand the intuitive strategy of setting probabilities proportional to observations in the data, you've understood the **key idea** behind maximum likelihood, and you should be able to derive simple estimators whenever you have observed data. Although it's not necessary for this class, you should know that the solution can be derived using basic calculus. For the curious, this is sketched in Appendix A.

At this point, let's consider an algorithm to learn θ from our hypothetical aligned data that actually instantiates this strategy. It's pretty simple: we scan the data, observing the alignments and counting them up for each French-English word pair. To obtain probabilities, we simply normalize each count by the number of times that we observed the corresponding English word participating in *any* alignment. This yields a simple algorithm, listed in Algorithm 1.

Note that the algorithm loops over all pairs of words in each sentence in order to collect counts, a computation that's quadratic in sentence length. This isn't strictly necessary: we could have just looped over the alignment variable to collect the counts, which is linear. However, thinking about the algorithm as one that examines all pairs of words will be useful when we move to the case of unobserved alignments, which turns out to be an extension of this algorithm.

Of course, this algorithm isn't *completely* hypothetical: we often have a small handful of sentences with observed alignments that we have collected from bilingual annotators. We could train using only this observed data, but you might suspect that this wouldn't produce a good estimate of θ, and you would be right. Think about all the word pairs that you would never see in a few hundred or thousand sentences. How much observed data do you think you would you need to get good estimates?

Algorithm 1 Maximum Likelihood estimation for (somewhat hypothetical) case of observed alignments

```
Initialize all counts to 0
for each n in [1, .., N] do
  for each i in [1, ..., I^(n)] do
    for each j in [1, ..., J^(n)] do                ▷ Start from 0 to include null words
      if a_i^(n) = j then
        count[⟨f_i^(n), e_j^(n)⟩]++                 ▷ Increment count of alignments between these two words
        count[e_j^(n)]++                            ▷ Increment marginal count of English word
for each ⟨f, e⟩ in count do
  p(f|e) = count(⟨f, e⟩)/count(e)                   ▷ Normalize
```

3 Expectation Maximization for Model 1

One problem with the maximum likelihood estimate we sketched above is that for the vast majority of our sentences, the alignment $\mathbf{a}$ isn't observed. Since Model 1 is defined in terms of this alignment, we call the alignment a *latent variable* (or a *hidden variable* or, if you're feeling less charitable, a *nuisance variable*). Expectation Maximization (EM) is a way to learn the parameters of a latent variable model like IBM Model 1. There are many others, but it is particularly useful to understand EM since it is relatively simple, widely applicable, and related to many other techniques. There are two key ideas in EM. The first is that we are going to replace the observed counts of alignment links with *expected counts* of alignment links, computed with respect to some previous estimate of θ. The second is that we compute these expected counts with respect to a previous estimate of θ, and iteratively improve it.

We'll take our inspiration for EM from maximum likelihood estimation. We need to make a slight change to Equation 6 in order to account for our new reality.

$$\mathcal{L}(\theta|\mathcal{D}) = p_\theta(\mathcal{D}|\theta) = \prod_{n=1}^{N} \underbrace{p_\theta(\mathbf{f}^{(n)}|\mathbf{e}^{(n)})}_{\text{remove } \mathbf{a} \text{ here}} = \prod_{n=1}^{N} p(I^{(n)}|J^{(n)}) \underbrace{\sum_{\mathbf{a}^{(\mathbf{n})}}}_{\text{marginalize over } \mathbf{a}} \prod_{i=1}^{I^{(n)}} p(a_i^{(n)}|J^{(n)}) \cdot p(f_i^{(n)}|e_{a_i}^{(n)}) \tag{8}$$

We're still maximizing the likelihood of the observed data, but since the observed data no longer includes alignments, they've been left out of the third expression. To account for this we must marginalize them out in the fourth expression. In other words, since we don't see the alignment, we *add up* the probabilities of *all possible alignments* that could have produced the data, because the rules of probability tell us that this gives us the marginal probability of $\mathbf{f}$, which is the quantity that we want to maximize. This idea sounds promising, but introduces a slight wrinkle, in that the sum over the alignments prevents an analytic solution.[2] This means that we'll have to solve the problem algorithmically, but it's going to turn out that the intuition behind the algorithm is very similar to the one for maximum likelihood.

Let's look more closely at the summation over alignments. Suppose that we want to estimate the parameter $p(f|e)$. Let's focus our attention on a sentence pair $\langle \mathbf{f}, \mathbf{e} \rangle$ for which French word $f_i = f$ and English

[2] Among other problems, if you look at Appendix A, you'll notice a common trick involving logarithms. This trick no longer works when we marginalize, because we are now taking the log of sums, rather than a log of products, which is much easier to manipulate. If you take partial derivatives of the likelihood directly, you'll obtain sums of products that couple many parameters together, with little hope for an analytic solution.

word $e_j = j$. The sum over alignments of $\langle \mathbf{f}, \mathbf{e} \rangle$ in Equation 8 includes many with a link between f_i and e_j, and many without. A question we can ask is: *on average*, what percentage of the (exponentially many) alignments contain the link between f_i and e_j. A formal way of asking this question is: what is the *posterior probability* $p(a_i = j|\mathbf{f}, \mathbf{e})$ of an alignment link between f_i and e_j? That is, given that we observe $\mathbf{f}$ and $\mathbf{e}$, what is the probability that there's a link between the two words? We also refer to $p(a_i = j|\mathbf{f}, \mathbf{e})$ as the *expected count* of the link, and we can calculate it with Bayes' rule.

$$p(a_i = j|\mathbf{f}, \mathbf{e}) = \frac{p(\mathbf{f}|a_i = j, \mathbf{e})p(a_i = j|\mathbf{e})}{p(\mathbf{f}|\mathbf{e})} = \frac{p(\mathbf{f}, a_i = j|\mathbf{e})}{p(\mathbf{f}|\mathbf{e})} \tag{9}$$

This equation says is that the posterior probability is the sum of the probabilities of all of the alignments containing a link between f_i and e_j, divided by the sum of the probability of all of the possible alignments. Already seems somewhat familiar, doesn't it? This number is simply a probability. Remember that in the MLE case, if we had seen the link we would have incremented our count 1, and if we hadn't seen it, we wouldn't have incremented anything at all, which is the same as adding 0. So this posterior probability, which is a number between 0 and 1, is a way of formally expressing our uncertainty about whether a link between f_i and e_j is actually there. If it's close to 0, we're fairly certain there is no link, and if it's close to 1, we're fairly certain that there is. If it's somewhere in the middle, we're not certain at all. The first **key idea** of Expectation Maximization is to replace counts of observed events with posterior probabilities of latent events.[3]

We need to calculate these posterior probabilities, and to do so we observe that:

$$\frac{p(\mathbf{f}, a_i = j|\mathbf{e})}{p(\mathbf{f}|\mathbf{e})} = \frac{p(I|J) \sum_{\mathbf{a}:a_i=j} \prod_{i'=1}^{I} p(a_{i'}|J) \cdot p(f_{i'}|e_{a_{i'}})}{p(I|J) \sum_{\mathbf{a}} \prod_{i=1}^{I} p(a_i|J) \cdot p(f_i|e_{a_i})} = \frac{\sum_{\mathbf{a}:a_i=j} \prod_{i'=1}^{I} p(f_{i'}|e_{a_{i'}})}{\sum_{\mathbf{a}} \prod_{i=1}^{I} p(f_i|e_{a_i})} \tag{10}$$

Note that we're able to cancel out the sentence length probability $p(I|J)$. Because $p(a_i|J)$ is uniform, the expression $\prod_{i=0}^{I} p(a_i|J)$ is constant across all alignments of the sentence, so we can also cancel it out. This leaves us with an expression written purely in terms of lexical translation probabilities, but both the numerator and denominator are sums over exponentially many terms. Let's rewrite the equation slightly by enumerating the summations over each element of $\mathbf{a}$.

$$\frac{\sum_{\mathbf{a}:a_i=j} \prod_{i'=1}^{I} p(f_{i'}|e_{a_{i'}})}{\sum_{\mathbf{a}} \prod_{i=1}^{I} p(f_i|e_{a_i})} = \frac{p(f_i|e_j) \sum_{a_1=0}^{J} \cdots \sum_{a_{i-1}=0}^{J} \sum_{a_{i+1}=0}^{J} \cdots \sum_{a_I=0}^{J} \prod_{i'=1}^{I} p(f_{i'}|e_{a_{i'}})}{\sum_{a_1=0}^{J} \cdots \sum_{a_I=0}^{J} \prod_{i=1}^{I} p(f_i|e_{a_i})} \tag{11}$$

The denominator is a sum over products of exactly I terms, and we can exploit the regularity of these terms to simplify the expression. The summation over values of a_1 first requires $p(f_1|e_1)$ to be multiplied by all possible permutations of probabilities of translations of words f_2 through f_I. We then multiply $p(f_1|e_2)$

[3] If you'd prefer a more mathematical intuition for this, refer to Appendix A and imagine what happens when we add the sum over alignments to the objective. You can derive a variant of Equation 17 in which $count(\langle f, e \rangle)$ is replaced by its expectation $\mathbb{E}_{p_\theta(\mathcal{D})} count(\langle f, e \rangle)$. While the expression is intuitive, it's also intractable because the exponent now depends on the parameters rather than a sufficient statistic.

by all permutations, and so on. Exploiting distributivity, we can therefore move the sum over values of a_1 inside the product over values of i. We can then do the same for the sum over values of a_2, and so on. We can similarly manipulate sums and products in the numerator, and simplify by canceling most of the terms.

$$\frac{p(f_i|e_j)\prod_{i'=1}^{I}\sum_{a_1=0}^{J}\cdots\sum_{a_{i-1}=0}^{J}\sum_{a_{i+1}=0}^{J}\cdots\sum_{a_I=0}^{J}p(f_{i'}|e_{a_{i'}})}{\prod_{i=1}^{I}\sum_{a_1=0}^{J}\cdots\sum_{a_I=0}^{J}p(f_i|e_{a_i})} = \frac{p(f_i|e_j)}{\sum_{a_i=1}^{J}p(f_i|e_{a_i})} \tag{12}$$

The final expression of Equation 12 tells us how to calculate the posterior probability $p(a_i = j|\mathbf{f}, \mathbf{e})$ in terms of a linear number of parameters $p(f|e)$. Now that we have a way to compute the fractional expected counts of a link, we can collect these counts across the entire dataset and then divide, just as if we were doing maximum likelihood estimation. In the parlance of EM, computing the expected counts is the *Expectation step*, or *E-step*, while normalization is the *Maximization step*, or *M-step*.

You probably noticed a small problem with this arrangement, which is that the parameters used to compute the expectations are what we were searching for in the first place! This problem is not fatal, however. What we can do is assign some initial value to θ, which we'll call θ_0. We can then compute the expected counts in terms of θ_0, and call the resulting maximum likelihood estimate on the expected counts θ_1. We then use θ_1 to compute expected counts that we use to estimate θ_2, and so on. This idea of iteratively improving our estimate of θ is the second **key idea** behind EM. Now that we have both of them, we can extend the code from Algorithm 1 to perform EM, shown in Algorithm 2.

Algorithm 2 Expectation Maximization for IBM Model 1

```
k = 0
Initialize θ_0                                   ▷ A common choice is to initialize uniformly
repeat
  k+ = 1
  Initialize all counts to 0
  for each n in [1, .., N] do                    ▷ E-Step: Compute expected counts
    for each i in [1, ..., I^(n)] do
      Z = 0                                      ▷ Z is commonly used to denote a normalization term
      for each j in [1, ..., J^(n)] do
        Z+ = p_{θ_{k-1}}(f_i^(n) | e_j^(n))
      for each j in [1, ..., J^(n)] do
        c = p_{θ_{k-1}}(f_i^(n) | e_j^(n)) / Z   ▷ Compute expected count
        count[⟨f_i^(n), e_j^(n)⟩]+ = c           ▷ Increment count of alignments by expected count
        count[e_j]+ = c                          ▷ Increment marginal count of English word by expected count
  for each ⟨f, e⟩ in count do
    p_{θ_k}(f|e) = count(f, e)/count(e)          ▷ M-Step: Normalize
until parameters converge or some other criterion (such as a fixed number of iterations) is met
```

Some questions naturally arise with this strategy, such as: what should θ_0 be? And when do we stop? There are actually two different answers to this question. The first is a theoretical answer, and it depends on

two important properties of EM, namely that the likelihood of θ_i will always be higher than the likelihood of θ_{i-1}, and that the likelihood function of Model 1 is convex. These properties guarantee that, in the limit, EM on IBM Model 1 will converge to the true maximum likelihood estimate. So in theory, the answers are that it doesn't matter what θ_0 is, and you should run it for "a long time". The empirical answer is somewhat different: most people run EM for only three to five iterations, and [Moore(2004)] shows that in this case, using heuristic intialization of θ_0 can significantly improve its accuracy. If you are implementing it, you may want to experiment with different approaches to initialization and stopping.

A final question: why does EM even work at all? Though we hinted at this above, the question is well beyond the scope of this tutorial. The essential idea is that the E-step constructs a function that is a lower bound on the true (log-)likelihood and which touches this likelihood at the current parameter setting. The M-step then chooses new parameters to maximize this lower bound. For more details see one of the many recent tutorials on EM [Dellaert(2002), for example]. The one by [Minka(1998)] includes a particularly helpful illustration on the second page.

4 Decoding with Model 1

One last question remains: how do we actually align the data once we've learned the model? When we ran EM, we marginalized out $\mathbf{a}$ while we were searching for a good value for θ. Now that we have $\hat{\theta}$, we can use it to make predictions about $\mathbf{a}$. A common way to do this is to find the assignment of $\mathbf{a}$ with highest probability. This simply finds the value $\hat{\mathbf{a}}$ that solves:

$$\hat{\mathbf{a}} = \arg\max_{\mathbf{a}} p(\mathbf{a}|\mathbf{e}, \mathbf{f}) \tag{13}$$

For Model 1, this is easy, again because alignments factor over the French words.

$$\hat{a}_i = \arg\max_{a_i} p(I|J) \prod_{i=1}^{I} p(a_i|J) \cdot p(f_i|e_{a_i}) \tag{14}$$

Since the only quantity that depends on a_i is $p(f_i|e_{a_i})$, this reduces to $\hat{a}_i = \arg\max_{a_i} p(f_i|e_{a_i})$. This yields Algorithm 3.

Algorithm 3 Most Probable Alignment

```
for each n in [1, .., N] do
  for each i in [1, ..., I^(n)] do
    best_prob = 0
    best_j = 0
    for each j in [1, ..., J^(n)] do
      if p(f_i^(n) | e_j^(n)) > best_prob then
        best_prob = p(f_i^(n) | e_j^(n))
        best_j = j
    align(n, i, best_j)
```

The most probable assignment is not the only way to predict alignments with IBM Model 1. An alternative suggested by [Liang et al.(2006)], is to threshold alignments by their posterior probability. They also give an alternative objective function for learning a model of $p(\mathbf{f}|\mathbf{e})$ and a model of $p(\mathbf{e}|\mathbf{f})$ simultaneously,

which they approximate by multiplying their posterior matrices during the E-step. This can be seen as an extension of a commonly used technique that combines the predictions of two models by intersecting them.

A Maximum Likelihood Estimation via Langrange Multipliers

Here's the basic idea. We want to solve for

$$\hat{\theta} = \arg\max_{\theta} \prod_{n=1}^{N} p(I^{(n)}|J^{(n)}) \prod_{i=1}^{I^{(n)}} p(a_i^{(n)}|J^{(n)}) \cdot p(f_i^{(n)}|e_{a_i}^{(n)}) \tag{15}$$

Something that will turn out to be helpful is to take the logarithm of this objective function.

$$\hat{\theta} = \arg\max_{\theta} \log \left(\prod_{n=1}^{N} p(I^{(n)}|J^{(n)}) \prod_{i=1}^{I^{(n)}} p(a_i^{(n)}|J^{(n)}) \cdot p(f_i^{(n)}|e_{a_i}^{(n)}) \right) \tag{16}$$

Since logarithm is monotonic, values of θ that maximize it will also maximize the original objective function. So we can directly optimize this *log-likelihood* function instead of the original likelihood function.

Before we go any further, let's rewrite this equation in a more useful form. The function is simply one big product of many parameters, including a bunch of alignment parameters $p(a_i|J)$ that we decided to hold uniform and sentence length parameters $p(J|I)$, which (as you'll see implied if you work through the manipulations below) we needn't worry about. Each translation probability $p(f|e)$ appears in the product exactly as many times as we saw an alignment link between French word f and English word e. Hence, by replacing the product of the various values we don't care about with C, we can write:

$$\hat{\theta} = \arg\max_{\theta} \log \left(C \cdot \prod_{f,e} p(f|e)^{count(\langle f,e \rangle)} \right) \tag{17}$$

Since this expression is the log of a product, we can rewrite it further as a sum of logs.

$$\hat{\theta} = \arg\max_{\theta} \log C + \sum_{f,e} count(\langle f,e \rangle) \log p(f|e) \tag{18}$$

In a typical optimization problem, we simply take the partial derivatives of the function with respect to each variable, set them equal to zero, and solve for parameters to yield an extremum (or saddlepoint) of the function, one which is hopefully a maximum. However, ours is a *constrained* optimization problem, which means that we require that the solution obey certain constraints.

$$\forall_e \sum_f p(f|e) = 1. \tag{19}$$

We can account for these constraints using Lagrange mutlipliers.[4] The idea is to add a term λ_e for each constraint, yielding a new function (called an auxiliary function) Λ in terms of θ and these multipliers.

$$\Lambda(\theta, \lambda) = \log C + \sum_{f,e} count(\langle f,e \rangle) \log p(f|e) - \sum_e \lambda_e \left(\sum_f p(f|e) - 1 \right) \tag{20}$$

[4] If this idea is new to you, it might be worth taking a look at one of the many tutorials on Lagrange multipliers [Klein(2004), for example, is popular with NLP audiences]. Again, this depth of understanding isn't crucial to our class, but Lagrangian techniques are extremely useful and crop up in a variety of contexts, so they are worth knowing.

Each multiplier λ_e simply expresses the original constraint, and we can now solve for θ under this new function without further complication. Taking the partial derivative with respect to a single parameter $p(f|e)$ yields:

$$\frac{\partial \Lambda(\theta, \lambda)}{\partial p(f|e)} = \frac{count(\langle f, e \rangle)}{p(f|e)} - \lambda_e \tag{21}$$

Solving the resulting system of equations for the $p(f|e)$'s under the constraint yields the anticipated solution:

$$p(f|e) = \frac{count(\langle f, e \rangle)}{\sum'_f count(\langle f', e \rangle)} \tag{22}$$

If you work this out by hand you'll see that in effect, the Lagrange multiplier λ_e expresses the normalization term for English word e.

References

[Brown et al.(1993)] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, Jun 1993.

[Collins(2011)] Michael Collins. Statistical machine translation: IBM models 1 and 2. 2011. URL `http://www.cs.columbia.edu/ mcollins/courses/nlp2011/notes/ibm12.pdf`.

[Dellaert(2002)] Frank Dellaert. The expectation maximization algorithm. 2002. URL `http://www.cc.gatech.edu/ dellaert/em-paper.pdf`.

[Klein(2004)] Dan Klein. Lagrange multipliers without permanent scarring. 2004. URL `http://www.cs.berkeley.edu/ klein/papers/lagrange-multipliers.pdf`.

[Knight(1999)] Kevin Knight. A statistical MT tutorial workbook. 1999. URL `http://www.isi.edu/natural-language/mt/wkbk.pdf`.

[Koehn(2010)] Philipp Koehn. *Statistical Machine Translation*. 2010.

[Liang et al.(2006)] Percy Liang, Ben Taskar, and Dan Klein. Alignment by agreement. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*, 2006.

[Minka(1998)] Tom Minka. Expectation-maximization as lower bound maximization. 1998. URL `http://www.cse.psu.edu/ rcollins/CSE598G/papers/minka98expectationmaximization.p`

[Moore(2004)] Robert C. Moore. Improving IBM word-alignment model 1. In *Proceedings of the Association for Computational Linguistics*, 2004.