IBM Model 1

For the first model, we implement the IBM Model 1 using the EM algorithm.
We need to compute $\mathrm{p}(\alpha|\mathbf{e},\mathbf{f}) = \prod_{j=1}^{l_e} \frac{t(e_j|f_{\alpha(j)})}{\sum_{i=0}^{l_f} t(e_j|f_i)}$,
where $t(e_j|f_i)$ is the translation probability of f_i to e_j.

To calculate $t(e_j|f_i) = \frac{\sum_{(\mathbf{e},\mathbf{f})} c(e|f;\mathbf{e},\mathbf{f})}{\sum_e \sum_{(\mathbf{e},\mathbf{f})} c(e|f;\mathbf{e},\mathbf{f})}$, we need $c(e|f;\mathbf{e},\mathbf{f}) = \frac{t(e|f)}{\sum_{i=0}^{l_f} t(e|f_i)} \sum_{j=1}^{l_e} \delta(e,e_j) \sum_{i=0}^{l_f} \delta(f,f_i)$,
where $\delta(i,j)$ is the Kronecker delta function.

We use the EM algorithm in order to obtain our $t(e|f)$ and our $c(e|f)$.
First, we initialize $t(e|f)$ uniformly. Then, for i iterations (in our experiments we set i = 5), we perform the expectation (E) step to get counts, and then the maximization (M) step to update our translation probabilities. Exact pseudocode for this can be found at:
http://mt-class.org/jhu/slides/lecture-ibm-model1.pdf

Once translation probabilities are calculated, to find an alignment of a sentence pair (**e,f**), we simply do an argmax:

for word e_j in sentence **e**:
    i = $\mathrm{argmax}_{f_i \in \mathbf{f}}[t(e_j|f_i)]$
    α_j = i

Note that in order to allow for a null alignment (an English word is not mapped to any foreign word), simply add a null token to every foreign sentence being trained on.

Hidden Markov Model

One of the main problems with the IBM Model 1 alignment algorithm is it only considers translation probabilities: it does not consider things such as position. For example, if you have the word "the" twice in an English sentence, and the words "le" and "la" in a french sentence, both "the"s will map to the same word, since in training it will find one of those translations more likely than the other - even though we can tell by where in the sentence they appear which word each "the" should map to.

Another issue is that the alignments created by IBM Model 1 have a tendency to be sporadic: they will bounce around. However, its likely that a phrase in English will translate to a phrase in the foreign language, and the foreign phrase will likely be clustered in the sentence.

We would like to fix these issues by introducing alignment probabilities. We create a first order Markov Model by deciding that an alignment α_j is dependent on the previous alignment α_{j-1}. Specifically, we want α_j to be dependent on the jump from α_{j-1} to α_j. So we want to calculate $\mathrm{p}(\alpha_j|\alpha_{j-1}, I)$, where I is the length of the foreign sentences, used for normalization purposes.

The HMM alignment probabilities can be written in the following form:
$\mathrm{p}(\alpha_j|\alpha_{j-1}, I) = \frac{s(\alpha_j - \alpha_{j-1})}{\sum_{i=1}^{I} s(i-\alpha_{j-1}}$, where s is a non-negative set of parameters defined:
$s(i-i') = \sum_\alpha P(\alpha|\mathbf{e},\mathbf{f}) \sum_j \delta(i', \alpha_{j-1})\delta(i, \alpha_j)$.

Rather than calculate these s counts in the E step of the EM algorithm, we instead do a maximum approximation where we only collect counts from the Viterbi alignment path. To calculate the Viterbi alignment path, we use the following dynamic programming formula:
$Q(i,j) = p(e_j|f_i) max_{i'=1,\ldots,I}[p(i|i', I)Q(i', j-1)]$, where I = len($\mathbf{f}$).
The Viterbi alignment is also used to calculate alignments after training.

Putting all these pieces together, we get the following algorithm for our HMM model:
1. Train translation probabilities with the EM algorithm from IBM Model 1.
2. Initialize transition probabilities uniformly.
3. Do the following steps for i iterations (5 in my trials):
4. set s parameters to 0
5. For each pair of sentences, calculate the Viterbi alignment path.
6. From the collected alignments, calculate s parameters.
7. From the calculated s parameters, update transition probabilities.

Ideally, the model learns that transition probabilities for jumps of 1, 2, even 3 are common, but it becomes extremely rare to have larger jumps, thus giving us a more "clustered" alignment.

One trouble this model has is our old way of including null transitions no longer works: since we choose a location in the sentence for the null token arbitrarily, words will have different transition probabilities to null based on their position in the sentence.

First, we continue adding a null token during the EM training to calculate translation probabilities. We remove these extra null tokens before beginning maximum approximation. To introduce the empty word into the HMM network, we extend $\mathbf{f}$ by I empty words. The empty words will have position i+I, and thus can encode the position of previous visited word, i. We then set the transition probability of a null token to a different null token to 0, and the transition of a word i to a null token not at i + I to 0. This way, we can only transition to the correct null state, and the null state will keep the position information of the last alignment.

Unfortunately, the HMM does not solve all the problems the IBM Model 1 has. For example, it doesn't really stop rare words from acting as garbage collectors. However, adding in the transition probabilities results in a small improvement to the Model 1 (in my trials, Model 1 scored an AER of 0.29, whereas HMM scored an AER of 0.24).