Word Alignment Models

David Talbot

Yandex School of Data Analysis

March 28, 2017

Abstract

Word alignment models play a fundamental role in phrase-based statistical machine translation and have made a reappearance in neural machine translation systems in the guise of *attention mechanisms*. A word alignment for a parallel sentence pair can be thought of as a binary matrix where a one in cell (i, j) indicates an alignment between the i-th word in the source sentence and j-th word in the target sentence.

Word alignments can be used directly to build a word-level translation model and as constraints to build phrase-level models by decomposing a parallel sentence pair into loosely independent components. Word alignments can be used within a neural machine translation system to help resolve long-range dependencies in the translation and to handle unknown words.

The main word alignment models that we will consider here were originally proposed by a team from IBM [1]. These are generative models that describe how a target sentence is generated given a source sentence making use of (hidden) word level alignments in the process.

These notes start with a review of maximum likelihood parameter estimation for generative models before introducing the Expectation-Maximization algorithm for handling missing data and showing how this can be applied to the word alignment problem.

1 Generative Models

Loosely stated, a generative is one that defines a joint probability distribution over some observed data and associated labels. When describing a generative model we typically specify a sequence of steps by which the data is generated and state any independence assumptions that the model makes.

For example, if we are given a bag of coins of two different colours (red and blue) and are told that all coins of the same colour have the same (unknown) probability of turning up heads, then the following model could describe a sample of data generated by picking a coin at random from the bag and flipping it n times:

1. Choose a red coin with probability λ otherwise choose a blue coin;

2. Flip the coin n times generating a head with probability H_{red} or H_{blue} depending on the coin chosen in step one.

Using $Z \in \{R, B\}$ to denote the coin chosen in step 1 and $X_i \in \{H, T\}$ to denote the outcome of the i-th flip, the joint probability of one such sample of data can be written as

$$\Pr(Z, X_1, X_2, ..., X_n) = \Pr(Z) \prod_{i=1}^{n} \Pr(X_i|Z)$$

since each of the flips is independent given the identity of the coin. In this example the joint probability factors naturally into a prior probability over the choice of coin and the conditional probabilities of heads or tails given the choice of coin. This factorization corresponds nicely to the physical steps involved in generating the data. In this sense you could say that this model is the true one.

In our word alignment models we will also usually want to factorize the probabilty our models assign to data, but this factorization will be driven by our intuitions of what should and can be modelled in the translation process rather than access to the true model that generated the data.

2 Maximum Likelihood Estimation

One way to estimate the parameters for this model $\lambda \doteq \Pr(Z)$, $H_{blue} \doteq \Pr(H|Z = B)$ and $H_{red} \doteq \Pr(H|Z = R)$ is to maximize the likelihood of some observed data. This can be done as follows. Given m samples D, we can

write down the joint likelihood as

$$\begin{aligned}
\Pr(D|\lambda, H_{red}, H_{blue}) &= \prod_{j=1}^{m} \Pr(Z_j) \prod_{i=1}^{n} \Pr(X_{ij}|Z_j) \\
&= \prod_{j=1}^{m} \lambda^{\delta(Z_j=R)}(1-\lambda)^{\delta(Z_j=B)} \times \\
&\quad \prod_{i=1}^{n} H_{red}^{\delta(Z_j=R,\ X_{ij}=H)}(1-H_{red})^{\delta(Z_j=R,\ X_{ij}=T)} \times \\
&\quad\quad H_{blue}^{\delta(Z_j=B,\ X_{ij}=H)}(1-H_{blue})^{\delta(Z_j=B,\ X_{ij}=T)}
\end{aligned}$$

where $\delta(x = y) = 1 \iff x = y$ otherwise 0.[1]

The maximum likelihood principle directs us to find those values of the parameters that maximize the likelihood (i.e., probability) of the data given the parameters. In practice we usually maximize the log-likelihood rather than the likelihood itself since this is a simpler and equivalent optimiziation. For instance, the maximum likelihood estimate for λ can be computed as follows

$$\begin{aligned}
\hat{\lambda} &= \operatorname*{argmax}_{\lambda} \log \Pr(D|\lambda, H_{red}, H_{blue}) \\
&= \operatorname*{argmax}_{\lambda} \log \prod_{j=1}^{m} \Pr(Z_j) \prod_{i=1}^{n} \Pr(X_{ij}|Z_j) \\
&= \operatorname*{argmax}_{\lambda} \sum_{j=1}^{m} \log \Pr(Z_j) + \sum_{i=1}^{n} \log \Pr(X_{ij}|Z_j) \\
&= \operatorname*{argmax}_{\lambda} \sum_{j=1}^{m} \delta(Z_j = R) \log \lambda + \delta(Z_j = B) \log(1-\lambda) + \ldots
\end{aligned}$$

where we can drop terms not containing λ from the final line.

The sum in this last line can be expressed as the total count of red and blue coins in our sample D. Using $\#(\cdot)$ to denote the count of an event in our sample D, we reformulate this as,

$$\hat{\lambda} = \operatorname*{argmax}_{\lambda} \#(R) \log \lambda + \#(B) \log(1-\lambda).$$

Since this function has a unique maximum, we can find it by differentiating

[1] $\delta(\cdot)$ allow us to pick out the terms that are actually needed, i.e. if $Z_j = B$ then $\lambda^{\delta(Z_j=R)}(1-\lambda)^{\delta(Z_j=B)}$ just evaluates to $(1-\lambda)$.

w.r.t. λ as follows

$$\begin{aligned} 0 &= \frac{\partial}{\partial \lambda} \Pr(D|\lambda, H_{red}, H_{blue}) \\ 0 &= \frac{\partial}{\partial \lambda} (\#(R) \log \lambda + \#(B) \log(1-\lambda)) \\ 0 &= \frac{\#(R)}{\lambda} + \frac{\#(B)}{(1-\lambda)} \end{aligned}$$

which implies that

$$\hat{\lambda} = \frac{\#(R)}{\#(R) + \#(B)}$$

which after all is rather obvious: the maximum likelihood estimate for a parameter of a discrete random variable is just the proportion of times the event occurred in the data.

Similarly the maximum likelihood estimates for H_{red} and H_{blue} can be found to be the proportion of heads in the samples generated by red coins and blue coins respectively, i.e.

$$\hat{H}_{blue} = \frac{\#(H, B)}{\#(B)}$$

and

$$\hat{H}_{red} = \frac{\#(H, R)}{\#(R)}.$$

3 Handling Hidden Variables

But how should we proceed if a careless friend were to drop our precious bag of coins into a tub of water and the colours were to wash off? In this case we will have no idea whether each coin we draw from the bag is red or blue. Since we still know, however, that there are two different types of coin in the bag (even if they are indistinguishable), it does not seem unreasonable to attempt to estimate parameters for the same underlying model: $\lambda, H_{blue}, H_{red}$.

We are faced with something of a chicken and egg problem here: if we knew the labels we could estimate the model parameters and if we knew the model parameters we might be able to infer the labels. Given that we know neither, one natural approach is to cluster the samples of coin flips into groups according to how many heads appear in them. The larger each sample is, the more chance we will have of determining the underlying distribution from which it is drawn and hence the correct underlying label. Let's assume for now that the samples are large enough to do this.

Having split the samples into two clusters which we will arbitrarily[2] call *red* and *blue*, we could then employ the maximum likelihood estimation formulae derived in the previous section to estimate our parameters. In fact there are two very common iterative algorithms that basically boil down to just this: k-*means* and *expectation-maximization*. Before explaining why they actually might work, let's first walk through them for our example model.

3.1 K-means algorithm

k-means is the simplest of these algorithms and is often further simplified by assuming a uniform prior, i.e. we fix $\hat{\lambda} = 0.5$.

The algorithm starts by randomly selecting (distinct) values for the parameters. We then iterate between assigning each sample to the class which assigns it greatest posterior probability (given the current parameter estimates) and re-estimating the parameters as though these assignments were the true labels.[3]

(Initialize) Choose $\hat{H}^0_{blue}, \hat{H}^0_{red}$ $u.a.r.$ $\in [0, 1]$.

(Label) At time t for $t \geq 1$ assign each sample in our data to the class which best *explains* it, i.e. for each sample X_j of n coin flips in D let

$$\hat{Z}^t_j = \underset{z \in \{B,R\}}{\operatorname{argmax}} \Pr(Z_j = z | X_j, \hat{H}^{t-1}_{blue}, \hat{H}^{t-1}_{red}).$$

(Re-estimate) Use these labels to re-estimate our two parameters using maximum likelihood, i.e. taking counts over all samples in D, let

$$\hat{H}^t_{blue} = \frac{\#(\hat{Z}^t = B, H)}{\#(\hat{Z}^t = B)}$$

and

$$\hat{H}^t_{red} = \frac{\#(\hat{Z}^t = R, H)}{\#(\hat{Z}^t = R)}.$$

It can be shown that each iteration of k-means will not decrease the likelihood assigned to the data and hence that the algorithm will converge. There are, however, no guarantees that it will converge to a global maximum.

[2] Of course *red* and *blue* are now just placeholders for two distinct but unidentifiable classes since we cannot hope to determine which of the two clusters of coins corresponds to which original colour, even if we do succeed in correctly splitting them into these two classes.

[3] Superscripts here indicate the iteration to which a parameter estimate belongs.

The posterior probabilities $\Pr(\hat{Z}^t = z|X_j, \hat{H}^{t-1}_{blue}, \hat{H}^{t-1}_{red})$ can easily be computed using Bayes' rule and the current parameter estimates. For $z \in \{R, B\}$

$$\Pr(\hat{Z}^t = z|X_j, \hat{H}^{t-1}_{blue}, \hat{H}^{t-1}_{red}) = \frac{\Pr(\hat{Z}^t = z)\Pr(X_j|\hat{Z}^t = z, \hat{H}^{t-1}_{blue}, \hat{H}^{t-1}_{red})}{\sum_{z' \in \{R,B\}} \Pr(\hat{Z}^t = z')\Pr(X_j|\hat{Z}^t = z', \hat{H}^{t-1}_{blue}, \hat{H}^{t-1}_{red})}$$

where all terms on the RHS can easily be computed from the current model, i.e. $\hat{\lambda}, \hat{H}^{t-1}_{blue}, \hat{H}^{t-1}_{red}$ and the counts of heads and tails in the sample X_j.

3.2 Expectation Maximization algorithm

Comparing the re-estimation formulae for k-means with those presented previously for maximum likelihood estimation (where the labels are available) we can see that we have just replaced the actual labels by our 'best guess' (given the current parameter estimates). The *expectation-maximization* algorithm takes this one step further by using the posterior distribution over each label directly as a *fractional weight* for that label assignment for the sample rather than forcing a hard assignment to one class or another.

The algorithm proceeds as follows. Initialize all parameter estimates randomly. Then repeatedly estimate the posterior distribution over the hidden class labels using the current parameters (as in k-means) for each sample and update the parameters using this posterior distribution as a fractional class assignment.

(Initialize) Choose $\hat{\lambda}^0, \hat{H}^0_{blue}, \hat{H}^0_{red}$ $u.a.r. \in [0, 1]$.

(E-step) Compute $\Pr(\hat{Z}^t = z|X_j, \hat{H}^{t-1}_{blue}, \hat{H}^{t-1}_{red})$ for each sample in D

(M-step) Update the parameters using these as *fractional observations*, e.g.

$$\hat{\lambda}^t = \frac{\sum_{j=1}^m \Pr(\hat{Z}^t = R|X_j, \hat{H}^{t-1}_{blue}, \hat{H}^{t-1}_{red})}{\sum_{z \in \{R,B\}} \sum_{j=1}^m \Pr(\hat{Z}^t = z|X_j, \hat{H}^{t-1}_{blue}, \hat{H}^{t-1}_{red})},$$

$$\hat{H}^t_{blue} = \frac{\sum_{j=1}^m \Pr(\hat{Z}^t = B|X_j, \hat{H}^{t-1}_{blue}, \hat{H}^{t-1}_{red})\#(H \in X_j)}{\sum_{j=1}^m \Pr(\hat{Z}^t = B|X_j, \hat{H}^{t-1}_{blue}, \hat{H}^{t-1}_{red})\#(H \cup T \in X_j)} \text{ etc.}$$

3.2.1 Correctness and convergence

Leaving behind our coins example now and considering the more general setting where we have some observed samples $X_j, j \in [1, m]$ and some associated hidden variables $Z_j, j \in [1, m]$ taking values from a set $\mathcal{Z}$ and we wish to estimate

model parameters θ for the joint distribution, we can restate the expectation-maximization algorithm in its more traditional form. The algorithm optimizes the following objective function known as the *expected log-likelihood*

$$\hat{\theta}^t = \operatorname*{argmax}_{\theta} \sum_{j=1}^{m} \sum_{z \in \mathcal{Z}} \mathbf{Pr}(Z_j = z | X_j, \hat{\theta}^{t-1}) \log \mathbf{Pr}(X_j, Z_j = z | \hat{\theta}^{t-1})$$

by iteratively computing the *expectation* over the hidden variables Z_j and *maximizing* this function (hence EM algorithm).

It is relatively straightforward to show that the *expected log-likelihood* is a lower bound on the *observed* (or *marginal*) *log-likelihood*, i.e. the log-likelhood of the observed data X_j once we sum out over the unknown hidden variables Z_j.

$$\begin{aligned}
\log \mathbf{Pr}(X_j | \hat{\theta}^{t-1}) &= \log \sum_{z \in \mathcal{Z}} \mathbf{Pr}(X_j, Z_j = z | \hat{\theta}^{t-1}) && (1) \\
&= \log \sum_{z \in \mathcal{Z}} q(Z_j = z) \frac{\mathbf{Pr}(X_j, Z_j = z | \hat{\theta}^{t-1})}{q(Z_j = z)} && (2) \\
&\geq \sum_{z \in \mathcal{Z}} q(Z_j = z) \log \frac{\mathbf{Pr}(X_j, Z_j = z | \hat{\theta}^{t-1})}{q(Z_j = z)} && (3) \\
&= \sum_{z \in \mathcal{Z}} q(Z) \log \mathbf{Pr}(X_j, Z_j = z | \hat{\theta}^{t-1}) - H(q(Z)). && (4)
\end{aligned}$$

Here in line 2 we introduced an arbitrary probability distribution q over Z_j, in line 3 we used Jensens inequality to move the logarithm inside the sum. Since the final term in line 4 is the entropy of $q(Z)$ which does not depend on our parameters θ, we can see that by increasing the expected log-likelihood in the M-step, we are also increasing our bound on the observed log-likelihood.

While $q(Z_j)$ can be an arbitrary probability distribution, we can show that the choice $q(Z_j) = \Pr(Z_j | X_j, \hat{\theta}^{t-1})$ will make the bound tight as follows

$$\begin{aligned}
\log \mathbf{Pr}(X_j | \hat{\theta}^{t-1}) &= \log \sum_{z \in \mathcal{Z}} \mathbf{Pr}(X_j, Z_j = z | \hat{\theta}^{t-1}) \\
&= \log \sum_{z \in \mathcal{Z}} q(Z_j = z) \frac{\mathbf{Pr}(X_j, Z_j = z | \hat{\theta}^{t-1})}{q(Z_j = z)} \\
&\geq \sum_{z \in \mathcal{Z}} q(Z_j = z) \log \frac{\mathbf{Pr}(X_j | \hat{\theta}^{t-1}) \mathbf{Pr}(Z_j | X_j, \hat{\theta}^{t-1})}{q(Z_j = z)} \\
&= \sum_{z \in \mathcal{Z}} q(Z_j = z) \log \mathbf{Pr}(X_j | \hat{\theta}^{t-1}) - \sum_{z \in \mathcal{Z}} q(Z_j) \log \frac{q(Z_j)}{\mathbf{Pr}(Z_j | X_j, \hat{\theta}^{t-1})} \\
&= \log \mathbf{Pr}(X_j | \hat{\theta}^{t-1}) - KL(q(Z_j) || \mathbf{Pr}(Z_j | X_j, \hat{\theta}^{t-1}))
\end{aligned}$$

since KL divergence term is minimized by setting $q(Z_j) = \Pr(Z_j | X_j, \hat{\theta}^{t-1})$.

4 Word Alignment Models

We now finally turn to the problem of word alignment. To those who were wondering what the preceding pages have to do with this problem, hopefully it will be clear that given sentence aligned parallel corpora (i.e. sentence pairs in source and target language that are translations of one another), word alignments can be thought of as *hidden variables* and therefore that, we might be able to use the EM algorithm to estimate them. First let's fix some of our notation.

- $\mathbf{e}$ indicates a source sentence of $e_i, i \in [1, I]$ tokens;
- $\mathbf{f}$ indicates a target sentence of $f_i, j \in [1, J]$ tokens;
- $\mathbf{a}$ indicates a set of alignments between tokens in sentence pair $(\mathbf{e}, \mathbf{f})$.

In general a set of *word alignments* $\mathbf{a}$ for a parallel sentence pair $(\mathbf{e}, \mathbf{f})$ can be thought of as a $I \times J$-maxtrix where cell (i, j) contains a value $a_{ij} \in \{0, 1\}$ indicating whether source token e_i should be aligned to target token f_j (see Figure 1). We can write the likelihood of an observed sentence pair $(\mathbf{e}, \mathbf{f})$ as a sum over all possible values of this hidden alignment matrix $\mathbf{a}$

$$\mathbf{Pr}(\mathbf{f}|\mathbf{e}, \theta) = \sum_{\mathbf{a} \in \mathcal{A}} \mathbf{Pr}(\mathbf{f}, \mathbf{a}|\mathbf{e}, \theta)$$

where $\mathcal{A}$ consists of all binary matrices of size IJ. Note, in particular, that we consider the source sentence $\mathbf{e}$ to be given and will not try to model it generatively. Our model is over word alignments and target words only. Here θ are some arbitrary and, as yet, unspecified parameters.

In the previous examples used to present the EM algorithm above, we were given our model for free. The probability of picking a red coin on each draw was independent of all previous draws and equal to λ. The probability of flipping a head was then conditionally independent of all previous outcomes given the colour of the coin. Unfortunately a tractable model that perfectly describes the process of generating a sequence of target words $\mathbf{f}$ given a source sentence $\mathbf{e}$ has yet to be discovered. This means we must design the model ourselves. Luckily a few people have had a go at this problem before us.

Figure 2 shows a set of word alignments for an English-French sentence pair taken from the paper that first seriously tackled this problem. The models proposed in that paper have become so standard that they are referred to by name: IBM models 1 to 5.

Before launching into these models, it is worth considering what constraints we might need to impose on candidate word alignment models. The following are probably a good start:

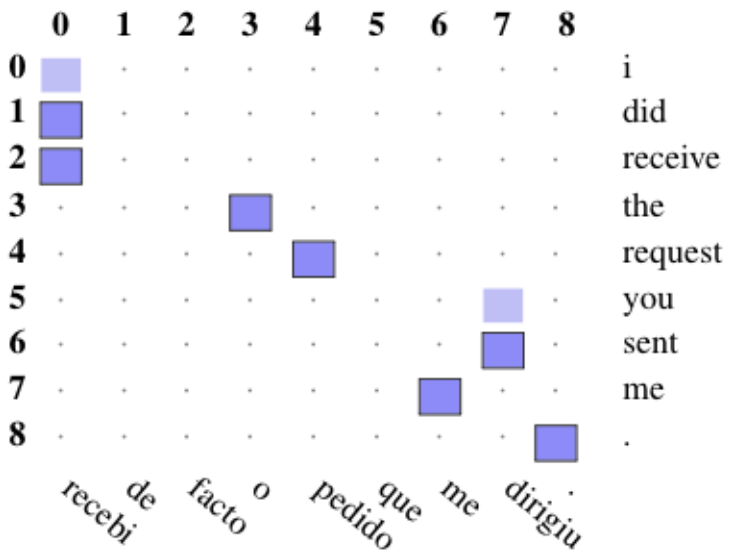

Figure 1. Word alignment matrix.

1. Not too many parameters: we don't have infinite amounts of parallel data, so if possible we should keep the number of free parameters in our word alignment models to a minimum;

2. Efficient inference: since the E-step of the EM algorithm (at least as presented above) requires us to compute the full posterior distribution over hidden variables, we should try to ensure that it is possible to enumerate these efficiently;

3. Efficient parameter updates: since we need to re-estimate the parameters on each M-step of the algorithm, it would be great if this is available in closed form.

Each of these constraints significantly limits the models we can use in practice, often against our intuitions of how translation should work; however, ignoring any one of them, may make a model significantly more difficult to work with.

We might, for example, wish to include separate parameters in our model for the translation of the same word in different contexts based on the fact that many words have multiple senses that are translated into completely different words in most other languages (e.g. river *bank* vs. financial *bank*). Doing this naively would, however, blow up the number of parameters significantly and in almost all cases result in much poorer word alignments since we would not have enough data to estimate these additional parameters. In general, the question of the number of parameters in a model is probably best determined empirically using some development data.

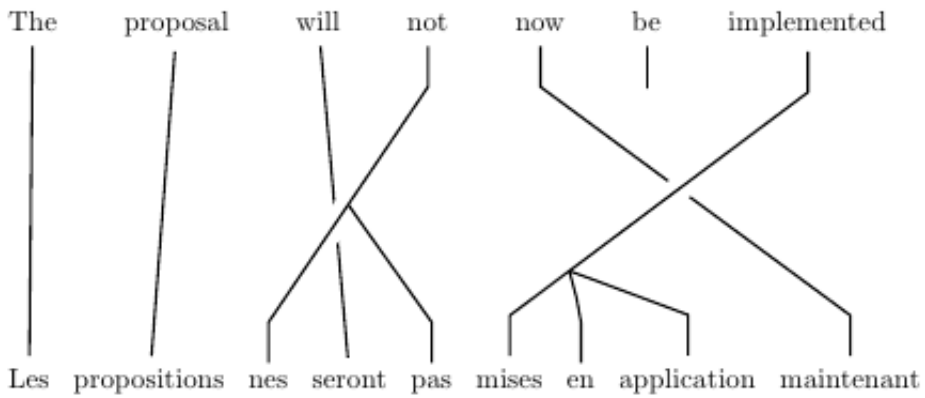

Figure 2. Word alignment example from Brown et al. (1993).

The second constraint (efficient inference) is often the most difficult to deal with in practice. As we will now see, some very natural word alignment models are completely intractable in this respect.

Let's take the initial model we specified above and first re-write it in terms of a *prior* and a *translation* model. These can be seen as analogous to the model that determined the choice of coin and the model that determined the probability of heads given the choice of coin in our previous example.

$$\begin{aligned}\mathbf{Pr}(\mathbf{f}|\mathbf{e},\theta) &= \sum_{\mathbf{a}\in\mathcal{A}} \mathbf{Pr}(\mathbf{f},\mathbf{a}|\mathbf{e},\theta) \\ &= \sum_{\mathbf{a}\in\mathcal{A}} \underbrace{\mathbf{Pr}(\mathbf{a}|\mathbf{e},\theta_a)}_{\text{prior model}} \underbrace{\mathbf{Pr}(\mathbf{f},|\mathbf{a},\mathbf{e},\theta_t)}_{\text{translation model}}\end{aligned}$$

As it stands right now, our model is completely intractable since the sum over $\mathbf{a} \in \mathcal{A}$ involves 2^{IJ} terms and, depending on how we actually parameterize the translation model, we run the risk of conditioning on all possible different source sentences. To make inference more tractable and reduce the number of parameters to a more reasonable size it is often desirable to introduce independence assumptions into a model. This is what we will now do.

4.1 IBM Model 1

Let's assume for now that each target word is aligned to a *single source word*. This isn't going to work very well if we have a phrase in the source sentence that was translated as a single word in the target sentence or vice versa, but it

should be a reasonable assumption much of the time. This assumption reduces the matrix $\mathbf{a}$ of size IJ to a vector a of size J where the j-th element a_j is an index $a_j \in [1, I]$ that indicates which source word the j-th target word f_j is aligned to. This doesn't yet make the model tractable but the set of possible alignments is now I^J rather than 2^{IJ}.

To make the model tractable for use with EM, we need to be able to compute the posterior distribution over alignments given each sentence pair $(\mathbf{e}, \mathbf{f})$; this entails a sum over all competing alignments. To make this sum manageable we make two independence assumptions: (1) each target word is *conditionally independent* given the source word to which it is aligned and (2) alignments are independent of one another, i.e. that a_j is independent of all other alignments $\mathbf{a} \setminus j$. The former assumption makes sense if we consider that word alignments should indicate 'dependencies' between words in the sentence pair. In fact without this or a similar assumption it seems unlikely that any of these models would work at all. The latter assumption is one that makes this first model particularly simple to estimate, but can be relaxed in more sophisticated models.

Given our modelling assumptions, we can now express the likelihood of each sentence pair as follows

$$
\begin{aligned}
\mathbf{Pr}(\mathbf{f}|\mathbf{e}) &= \sum_{\mathbf{a}\in\mathcal{A}} \mathbf{Pr}(\mathbf{a}|\mathbf{e},\theta)\mathbf{Pr}(\mathbf{f}|\mathbf{e},\mathbf{a},\theta) \\
&\approx \prod_{j=1}^{J} \sum_{\mathbf{a}\in\mathcal{A}} \mathbf{Pr}(\mathbf{a}|\mathbf{e},\theta)\mathbf{Pr}(f_j|e_{a_j},\theta) \\
&= \prod_{j=1}^{J} \sum_{\mathbf{a}\setminus j\in\mathcal{A}} \mathbf{Pr}(\mathbf{a}\setminus j|\mathbf{e},\theta) \sum_{i=1}^{I} \mathbf{Pr}(a_j = i|\mathbf{a}\setminus j,\mathbf{e},\theta)\mathbf{Pr}(f_j|e_{a_j},\theta) \\
&\approx \prod_{j=1}^{J} \sum_{i=1}^{I} \mathbf{Pr}(a_j = i|\mathbf{e},\theta)\mathbf{Pr}(f_j|e_{a_j},\theta) \sum_{\mathbf{a}\setminus j\in\mathcal{A}} \mathbf{Pr}(\mathbf{a}\setminus j|\mathbf{e},\theta) \\
&= \prod_{j=1}^{J} \sum_{i=1}^{I} \mathbf{Pr}(a_j = i|\mathbf{e},\theta)\mathbf{Pr}(f_j|e_{a_j},\theta)
\end{aligned}
$$

where, in the second line, we use the assumptions that target words are aligned to a single source word and generated independently given the source word to which they are aligned; in the third line we partition the alignments into those for the current target word a_j and those for all other target words $\mathbf{a} \setminus j$; in the fourth line we use the assumption that the current alignment is independent of all other alignments to move the sum over them to the far right and in the final line we have magically ended up with something that is quite easy to compute by recognizing that the righmost term on the penultimate line just sums to 1 (being a probability distribution). This is known as IBM Model 1.

When applying EM to this model, we can compute the posterior distribution $\mathbf{Pr}(a_j|\mathbf{f},\mathbf{e},\theta)$ needed by using Bayes' rule for each target word as follows

$$\begin{aligned}\mathbf{Pr}(a_j = i|\mathbf{f},\mathbf{e},\theta) &\approx \frac{\mathbf{Pr}(f_j, a_j = i|\mathbf{e},\theta)}{\mathbf{Pr}(f_j|\mathbf{e},\theta)} \\ &\approx \frac{\mathbf{Pr}(a_j = i|\theta)\mathbf{Pr}(f_j|e_i,\theta)}{\sum_{k=1}^{I}\mathbf{Pr}(a_j = k|\theta)\mathbf{Pr}(f_j|e_k,\theta)} \\ &= \frac{\epsilon t(f_j|e_i)}{\sum_{k=1}^{I}\epsilon t(f_j|e_k)} \\ &= \frac{t(f_j|e_i)}{\sum_{k=1}^{I} t(f_j|e_k)}.\end{aligned}$$

Under this model the posterior probability that a target word f_j is aligned to a source word e_i will be higher when the model assigns a high probability to the pair $t(f_j|e_i)$ and the other source words in the sentence don't assign high probabilities to it. Or to put it another way, the posterior will be low, even if the parameter $t(f_j|e_i)$ is relatively high, if one or more of the other words provides a better explanation (i.e. assigns a higher probability) to this alignment (these are the terms in the denominator).

We can now collect counts over the corpus for each pair of source and target words weighted by their posterior alignment probability and normalize these into separate conditional distributions $t(f|e)$ for each source word e, i.e.

$$t(f|e)_{t+1} \propto \sum_{k\in\mathcal{D}}\sum_{j}\sum_{i}\mathbf{Pr}(a_{jk} = i|\mathbf{e}_k,\mathbf{f}_k,\theta_t)\delta(f_{jk} = f, e_{ik} = e).$$

4.2 Improving the model

IBM Model 1 consists solely of a table of translation probabilities $t(f|e)$. In fact such a table of translation probabilities dominates most word alignment models but it's clear that having a separate parameter for each pair of source and target words isn't very efficient. For example, we probably expect the translation of two different source words with the same lemma (e.g. 'dog' and 'dogs') to be related in the target language but having a table with entirely separate entries for each pair of source and target types doesn't allow any of these relationships between words in the same language to be used by the model. Lemmatization or stemming can be applied to parallel corpora prior to alignment if this is judged to be a problem. The data that you have been provided with for the assignment includes lemmas of the Czech parallel corpus.

In your assignment you might want to explore various ways of mapping the input words to simplify the parameter space. In other words, rather than modelling e

and f directly as $t(f|e)$ you might model some functions $\mathcal{E}(e)$ and $\mathcal{F}(f)$ of the actual words, i.e. build a table for $t(\mathcal{F}(f)|\mathcal{E}(e))$. This is known as 'parameter tying'. If the functions $\mathcal{E}(e)$ and $\mathcal{F}(f)$ map words in such a way that the parameter space is significantly reduced but words that need to 'compete' with one another for alignments are still distinguished then you should see significant improvements in word alignment accuracy. It's interesting to consider that the word alignment task is probably much simpler than full translation since we are only required to choose between a very small set of possible words (i.e. those in the source sentence). For this reason, it is likely that the task requires much less information than is actually present in the raw parallel corpus.

Here are some exercises to help you improve the translation model:

- What alignment error rate (AER) do you get for the current Model 1?
- Why might it be okay not to model the exact words e and f in the translation table $t(f|e)$?
- What simple mappings of the words can you think of?
- What changes would you need to make to the class $TranslationModel$ if you decide to map words on-the-fly rather than in the corpus files?
- How does AER change for different mappings?
- How could you use the training data itself to determine the mapping before starting training?
- How could you optimize a mapping using the dev and test sets?
- Why might the performance of a mapping depend on the amount of training data?

4.3 More Complex Priors

IBM Model 1 presented above is the simplest of the models proposed in [1]. One key way to improve on this model is to replace the uniform prior $\mathbf{Pr}(a_j|\mathbf{e},\theta) = \epsilon$ with something more informative. In [1] the authors proposed Model 2 in which the prior takes into account the source and target token indices i and j as well as the sentence lengths I and J, i.e. $\mathbf{Pr}(a_j|\mathbf{e},\theta) = \mathbf{Pr}(a_j = i|j,I,J)$. Let's denote the parameters for the prior as $p(i|j,I,J)$. This results in the following

expected complete log likelihood for a sentence pair

$$\begin{aligned}\mathbb{E}[\log \mathbf{Pr}(\mathbf{f}, \mathbf{a}|\mathbf{e}, \theta)] &\approx \sum_{j=1}^{J}\sum_{i=1}^{I} \mathbf{Pr}(a_j|\mathbf{e}, \mathbf{f}) \log \mathbf{Pr}(a_j = i|\theta)\mathbf{Pr}(f_j|e_{a_j}, \theta) \\ &= \sum_{j=1}^{J}\sum_{i=1}^{I} \mathbf{Pr}(a_j|\mathbf{e}, \mathbf{f}) \log p(i|j, I, J)\ t(f_j|e_{a_j}).\end{aligned}$$

We can compute the posterior distribution $\mathbf{Pr}(a_j|\mathbf{e}, \mathbf{f}, \theta)$ for this model in a similar way to Model 1 above as

$$\begin{aligned}\mathbf{Pr}(a_j = i|\mathbf{e}, \mathbf{f}, \theta) &\approx \frac{\mathbf{Pr}(a_j = i|\theta)\mathbf{Pr}(f_j|e_i, \theta)}{\sum_{k=1}^{I} \mathbf{Pr}(a_j = k|\theta)\mathbf{Pr}(f_j|e_k, \theta)} \\ &= \frac{p(i|j, I, J)t(f_j|e_i)}{\sum_{k=1}^{I} p(k|j, I, J)t(f_j|e_k)}.\end{aligned}$$

The updates for the translation parameters $t(f|e)$ are the same as for Model 1 above, but we now need to re-estimate the prior probabilities $p(i|j, I, J)$. Given the posterior distributions over alignments we can update the prior parameters as follows by summing over the sentence pairs in the corpus keeping separate counters for sentences of different lengths

$$p(i|j, I, J)_{t+1} \propto \sum_{k \in \mathcal{D}} \mathbf{Pr}(a_j = i|\mathbf{e}_k, \mathbf{f}_k, \theta)\delta(|\mathbf{f}_k| = J, |\mathbf{e}_k| = I).$$

The only real difference in the updates for the prior versus the translation table parameters is in statistics collected by the $\delta(\cdot)$ function.

Exercises to help you improve this prior model:

- What AER do you get when you add this prior to Model 1?
- Does it help even when you only use 10K or 1K sentence pairs?
- What happens if you first train Model 1 to convergence and then add this prior?
- Does it really make sense to have separate parameters for all different sentence lengths and positions?
- How could you map the positional information (i, j, I, J) to build a more succinct prior model?

4.4 Hidden Markov Alignment Model

There are many ways in which the prior model could be improved. The most obvious extension which was proposed by [6] is to add a dependency between alignments. A simple way to do this is to make the alignment for the j-th target word depend on the alignment of the $j-1$-th target word. This corresponds to a 'Markov' assumption: a Markov chain is a sequence of variables where each variable depends only on the preceding one. Looking at some of the reference word alignments it's clear 'a priori' (i.e. without actually considering the words involved in the alignment) the most useful piece of information for deciding where to place the alignment a_j is the value of a_{j-1}.

The prior probabilities from Model 2, $p(i|j, I, J)$ are replaced in this Hidden Markov model by a matrix of transition probabilities to model $\mathbf{Pr}(a_j|a_{j-1}, \mathbf{e}, \theta)$. We will denote these by $h(i|i', j, I, J)$ where i' is the source index to which f_{j-1} is aligned. Under this HMM model the joint probability of a target sentence and its alignment is given by

$$\mathbf{Pr}(\mathbf{f}, \mathbf{a}|\mathbf{e}, \theta) = \prod_{j=1}^{J} h(a_j|a_{j-1}, I, J)t(f_j|e_{a_j}).$$

Computing the posterior distribution over alignments is, however, no longer a simple matter of summing over possible values of a_j for each target word in turn since these are no longer independent.

In addition, the posterior distribution that we need to reestimate the HMM transition parameters differs in that we need to know the 'expected counts' of pairs of alignments (a_j, a_{j1}) rather than just the posterior distribution at a single position j as for Models 1 and 2.

There is however an efficient algorithm for computing this posterior distribution in time $O(JI^2)$ known as the Forward-Backward algorithm.[4] The algorithm is based on dynamic-programming and makes use of the fact that given the hidden alignment at position $j-1$ the alignment at position j is conditionally independent of all other alignments.

Given the posterior distribution over pairs of consecutive alignments for sentences in our corpus, i.e. $\mathbf{Pr}(a_j = i, a_{j-1} = i'|\mathbf{f}, \mathbf{e}, \theta)$ computed by the Forward-Backward algorithm, we can compute the new transition parameters as follows in the M-step

$$h(i|i', I, J) \propto \sum_{k} \sum_{j=1}^{J} \mathbf{Pr}(a_{jk} = i, a_{(j-1)k} = i'|\mathbf{e}_k, \mathbf{f}_k, \theta)\delta(|\mathbf{e}_k| = I, |\mathbf{f}_k| = J)$$

[4] A nice overview of inference and parameter estimation for HMMs is given in [5].

where we must normalize over all other indices transitioned to from position i' on sentence pairs of length I and J respectively for source and target. The translation parameters $t(f|e)$ are updated identically to Models 1 and 2 from the posterior alignment probabilities for each position j marginalizing over the alignment for position $a_{(j-1)}$.

As with Model 2 above there is no need to make the transition model condition on the target sentence length J. There's also no need to have it condition on the target index j although that might be useful. Removing this second condition makes the HMM 'time-homogenous'. In practice it is common to simply model the size of the 'jump' from a_{j-1} to a_j using the parameters $h(i|i', I)$ (see [6] for details).

If you implement an HMM alignment model you might want to consider the following issues:

- Should you model absolute positions or distance between consecutive alignments?
- What should you condition the transition matrix on aside from the position? E.g. you could condition on the tag of the previously aligned source word $e_{a_{j-1}}$ to capture the fact that certain parts of speech result in specific alignment patterns.
- The Forward-Backward algorithm can suffer from underflow on longer sentences. You might want to try re-scaling terms at each position in the dynamic program to prevent this (see [5]).

4.5 Words Appearing from Nowhere

Some words in the target language might not have a good 'explanation' in the source. These might be language specific words such as articles 'the', 'a' etc. in English or just words that the translator felt necessary to add for clarity or by mistake.

As you can see from this sentence pair taken at random, the Czech includes a phrase 'za minuly rok' which doesn't seem to correspond to anything in the English but was probably inferred from the context by the translator.

(en) Revenue rose to $ 225 million from $ 161 million .

(cs) Vynosy vzrostly na 225 milionu z 161 milionu za minuly rok .

Unfortunately, the models we considered above, all require each target word to be aligned to exactly one source token.

One way to deal with this is to introduce an imaginary *NULL* word in each source sentence. This can be incorporated easily in Model 1 simply by defining the $NULL$ word to be e_0 and starting the sum over i from 0, e.g. we imagine that each source sentence looks something like

(en) NULL Revenue rose to $ 225 million from $ 161 million .

Rather than assume a uniform prior over the NULL word, we might add a separate parameter ϕ_{null} for this and estimate it during EM or alternatively on a small amount of held out data optimizing either the log-likelihood or word alignment error rate. The expected complete data log likelihood for Model 1 with a separate NULL word prior is

$$\mathbb{E}[\log(\mathbf{f}, \mathbf{a}|\mathbf{e})] = \prod_{j=1}^{J} \sum_{i=0}^{I} \mathbf{Pr}(a_j = i|\mathbf{f}, \mathbf{e}, \theta) \log t(f_j|e_{a_j})(1-\phi_{null})^{\delta(a_j \neq 0)} \phi_{null}^{\delta(a_j=0)}$$

Incorporating the NULL word into Model 2 is not hard, but it does require you to be careful to avoid making the prior probability depend on the NULL word's position (we just placed it at position 0 for convenience not because we really believe it's located there - it's a figment of our imagination and our model).

4.6 Using Linguistic Annotations as Priors

The data provided for your assignment includes lemmas, part of speech tags and morphological analysis for the training and test corpora. For example

(en) NN VBD IN $ CD CD IN $ CD CD .

(cs) NN Vp RR C= NN RR C= NN RR AA NN Z:

Part of speech (POS) tags are labels such as noun, verb, adjective etc. The tag sets for English and Czech are different, but there some obvious patterns that a statistical model should be able pick these up. Linguistic information could be incorporated as a prior or as part of the translation model. Here are a few possible approaches

- Define a prior over English and Czech tags, e.g. $\mathbf{Pr}(a_j = i) \approx \mathbf{Pr}(tag(f_j)|tag(e_i))$.

- Map low frequency words in the corpus to their POS tags or lemmas.
- Condition the translations of ambiguous words on their POS, e.g. $\mathbf{Pr}(f_j|e_i) \approx \mathbf{Pr}(f_j|e_i, tag(e_i))$ in order to distinguish homonyms such (to) *play* v. and (a) *play* n.
- Develop a post-processor that corrects common alignment errors based on the parts of speech.
- Have a separate prior for the NULL word alignment for each POS (e.g. English articles might be very likely to be null aligned).

4.7 Incorporating Heuristic Priors

Looking at (the same) random sentence pair from the Czech-English parallel corpus, there are a few word pairs here that we could probably align without any knowledge of either language or looking at any other data: numbers, dates, addresses and names are commonly translated identically; but often these tokens are actually quite hard for purely word based statistical models to align since they are rare.

(en) Revenue rose to $ 225 million from $ 161 million .

(cs) Vynosy vzrostly na 225 milionu z 161 milionu za minuly rok .

If we can accurately identify tokens that must be aligned, e.g. (225, 225) and (161, 161) in this sentence pair, then there's no reason we shouldn't try to incorporate that into our model. Having build a heuristic aligner that finds such tokens we could try a couple of different approaches.

1. Replace tokens that should be aligned by some new common symbols (being careful to distinguish multiple pairs in the same sentence). By using the same small set of symbols accross sentence pairs in the corpus, the statistical models should learn to align them with high probability.
2. Encorporate the knowledge as a hard prior for the target tokens for which we have a constraint (i.e. set the prior to 1 for the token to which it must be aligned and 0 for all others).

As well as finding pairs of identical low frequency tokens on either side of the corpus, we could also try to learn more sophisticated heuristic prior models, such as, for example, a character edit model that detects similar words. This might work particularly well for related languages which have many *cognates*

that have retained the same meaning. For instance, in this English - French sentence pair most words could be aligned by such heuristics.

(en) *Donald Trump* wants to repeal *Obama* 's *environmental measures*

(fr) *Donald Trump* veut revenir sur les *mesures environnementales* d' *Obama*

4.8 Using Data from Related Languages

For the assignment you have been provided with data in Czech and English on which you should train and evaluate your word alignment models. In addition, however, you may make use of some additional parallel data provided for Slovak - English and Slovenian - English. These languages are both closely related to Czech (Slovak more so than Slovenian). It is up to you whether or how you make use of this additional data, but you might find it helpful for resolving alignments for rare words in the Czech corpus.

One way to use the related language data would be to train word alignment models as usual on all three language pairs separately and then to train a character level model that assigns translation probabilities between Czech and Slovak or Slovenian words. This could be estimated from pairs of Czech and Slovak/Slovenian words that were aligned confidently with the same English words in the different corpora. By parameterizing this model at the character level, you could then attempt to use it to help align rare Czech words in the main corpus.

You are not required to use this additional data, but I will give extra credit to students who try to improve their models with it.

5 Experimentation

Given a golden set of manually created word alignments consisting of probable P and sure S alignments, we can measure the error rate of an automatic alignment A as follows

$$Precision(A; P) = \frac{|P \cap A|}{|A|}$$

$$Recall(A; S) = \frac{|S \cap A|}{|S|}$$

$$AlignmentErrorRate(A; S, P) = 1 - \frac{|P \cap A| + |S \cap A|}{|S| + |A|}.$$

You have been provided with two sets of manual word alignments (dev and test) and a script to compute the alignment error rate (AER) on these.

In your assignment I would like to see you try out a number of different models and to test and analyze various hypotheses using this data. As well as providing graphs of AER scores, I would like to see evidence of you examining errors on particular sentence pairs and comparing your models' alignments with the reference alignments provided.

You will get credit for any analyses regarding your word alignment models that you can support with data and/or examples.

References and Notes

[1] 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*, 19:263-311, 1993.

[2] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, Series. B, 39(1):1-38, 1977.

[3] Joao Graca, Joana Paulo Pardal, Luisa Coheur, and Diamantino Caseiro. Building a golden collection of parallel multi-language word alignment. In *Proceedings of LREC'2008*, May 2008.

[4] R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. *Learning in Graphical Models*, pages 355-368, 1999.

[5] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257-286, 1989.

[6] Stephan Vogel, Hermann Ney, and Christoph Tillmann. HMM-based word alignment in statistical translation. In *Proceedings of the 16th Conference on Computational Linguistics*, Volume 2, COLING '96, pages 836-841, 1996.