Parameter learning (efficient)
How to sum alignments efficiently
Efficient EM via \( p((j, i) \in a \mid o, s) \)

but what about Exponential cost?

- the learnability of translation probabilities in an unsupervised fashion from just a corpus of pairs is a remarkable thing
- however, as we have formulated it, each possible alignment has to be considered in turn, and each contributes increments to expected counts
- it was already noted that the number of possible alignments is \((\ell_s + 1)^{\ell_o}\) – ie. exponential in the length of \(o\). For \(\ell_s + 1 = \ell_o = 10\), this is \(10^{10}\), or 10,000 million
- so unless a way can be found to make the EM process on this model much more efficient, its learnability in principle would just be an interesting curiosity
- it turns out that by studying a little more closely the formula where alignments are summed over, and doing some conversions of ‘sums-over-products’ to ‘products-over-sums’, it is indeed possible to make the EM process on this model much more efficient.
Summing over alignments

Looking at the brute-force EM algorithm, need to calculate \( p(\mathbf{o}|\mathbf{a}, s) \) – call this \( \gamma_d(a) \). It’s fairly easy to see that this is²

\[
\gamma_d(a) = \frac{\prod_j p(o_j|s_{a(j)})}{\sum_{\mathbf{a}^*} \prod_j p(o_j|s_{a^*(j)})}
\]  

(11)

- The numerator is a product.
- It turns out the denominator can also be turned into a product of sums

² in the formula for \( p(\mathbf{o}|\mathbf{a}, t_s, s) \) everything except the translation probs is going to cancel out when you take ratios …

Pause: did you believe that?

the key step above was a conversion from a sum-of-products to a product-of-sums.

for the case of \( \mathbf{o}, s \) having length 2 can relatively easily verify by brute force

\[
\sum_{a(1)=0}^{2}\sum_{a(2)=0}^{2} \prod_{j=1}^{2} t(o_j|s_{a(j)}) =
\]

\[
= t(o_1|s_0) t(o_2|s_0) + t(o_1|s_0) t(o_2|s_1) + t(o_1|s_0) t(o_2|s_2) +
\]

\[
+ t(o_1|s_1) t(o_2|s_0) + t(o_1|s_1) t(o_2|s_1) + t(o_1|s_1) t(o_2|s_2) +
\]

\[
+ t(o_1|s_2) t(o_2|s_0) + t(o_1|s_2) t(o_2|s_1) + t(o_1|s_2) t(o_2|s_2)
\]

\[
= t(o_1|s_0)[t(o_2|s_0) + t(o_2|s_1) + t(o_2|s_2)] +
\]

\[
+ t(o_1|s_1)[t(o_2|s_0) + t(o_2|s_1) + t(o_2|s_2)] +
\]

\[
+ t(o_1|s_2)[t(o_2|s_0) + t(o_2|s_1) + t(o_2|s_2)] =
\]

\[
= [t(o_1|s_0) + t(o_1|s_1) + t(o_1|s_2)][t(o_2|s_0) + t(o_2|s_1) + t(o_2|s_2)]
\]

Summing over alignments contd

each \( j \) can be aligned to any \( i \) between 0 and \( J \), hence

\[
\sum_{a} \prod_{j} t(o_j|s_{a(j)}) = \sum_{a(1)=0}^{J} \ldots \sum_{a(J)=0}^{J} t(o_j|s_{a(j)})
\]

\[
= \sum_{a(1)=0}^{J} \ldots \sum_{a(J)=0}^{J} [t(o_1|s_{a(1)}) \ldots t(o_J|s_{a(J)})]
\]

each \( \sum_{a(1)=0}^{J} \) effects just one \( t(o_j|s_{a(j)}) \) term, and this means we can use a sum-of-products to product-of-sums conversion, hence

\[
= \prod_{j=1}^{J} \left[ \sum_{a(1)=0}^{J} t(o_j|s_{a(j)}) \right]
\]

\[
= \prod_{j=1}^{J} \left[ \sum_{a(1)=0}^{J} t(o_j|s_{a(j)}) \right]
\]

Making \( p(\mathbf{o}|\mathbf{s}) \) into a product

Armed with this, we can rewrite (11) the formula for \( \gamma_d(a) \) as

\[
\gamma_d(a) = \frac{\prod_{j=1}^{J} t(o_j|s_{a(j)})}{\prod_{j=1}^{J} \sum_{a(1)=0}^{J} t(o_j|s_{a(j)})}
\]

and this is just one big product

\[
= \prod_{j=1}^{J} \left[ \frac{t(o_j|s_{a(j)})}{\sum_{a(1)=0}^{J} t(o_j|s_{a(j)})} \right]
\]

each term in this product can be seen as the probability of a particular alignment step \((j, i)\), given \( \mathbf{o}, s \), and it makes sense for the overall alignment probability to be a product of the individual steps. If we use the notation \( \gamma_d(j, i) \) for this probability of a single alignment step, we get

\[
\gamma_d(a) = \prod_{j=1}^{J} \gamma_d(j, a(j))
\]
We have for $\gamma_d(j, i)$

$$
\gamma_d(j, i) = \frac{t(o|s)}{\sum_{i'=0}^I t(o|s')}
$$

(12)

- crucially the cost of calculating $\gamma_d(j, i)$ is trivial – its linear in length of $s$
- The efficient version of EM rests on seeing that once $p(j, i|o, s)$ is worked out for each $j, i$, the desired expected $(o, s)$ counts can be worked out from them.

Summing over the alignments gives just $\gamma_d(j, i)$

for $o$ position $j$, $s$ position $i$ is fixed. For every other $o$ position $j', j'$ can be aligned to any $i$ between 0 and $I$, hence

$$
\sum_{a(j,i) \in a} \gamma_d(a) = \sum_{a(1)=0}^I \sum_{a(j-1)=0}^I \sum_{a(j+1)=0}^I \sum_{a(j)=0}^I \gamma_d(j, i) \prod_{j' \neq j} \gamma_d(j', a(j'))
$$

we can pull out $\gamma_d(j, i)$ and again do a sum-of-products to product-of-sums conversion with the rest, hence

$$
= \gamma_d(j, i) \prod_{j' \neq j} \left[ \sum_{a(j')=0}^I \gamma_d(j', a(j')) \right]
$$

each sum runs over every possible alignment destination for $j'$ and so each one sums to one, so you get just

$$
= \gamma_d(j, i)
$$

call the $[E]$ step of the brute-force algorithm (if $o, s$ are the $d^{th}$ pair):

for each pair $(o, s)$

for each $a$ calculate $p(a|o, s)$ // pseudo counts of $(o, s)$ word pairs

for each $j \in 1: \ell_o$

$\#(o_j, s_{x(j)}) += p(a|o, s)$ // in virtual data

consider a particular $(j, i)$. As you go through all possible $a$ for $o, s$, each time the alignment $a$ contains this pairing you make the increment $\gamma_d(a)$.

... aim for an algorithm which works out quickly for each $j, i$ what the sum of these increments will be, ie.

$$
\sum_{a(j,i) \in a} \gamma_d(a)
$$

(13)

$3$ The notation $\sum_{a(j, i) \in a}$ means 'sum over only those $a$ that have $(j, i) \in a$

Efficient EM algorithm for IBM Model 1 training

initialise $tr(o|s)$ uniformly

repeat $[E]$ followed by $[M]$ till convergence

$[E]$

for each $o \in V_o$

for each $s \in V_s \cup \{\text{NULL}\}$

$\#(o, s) = 0$

for each pair $o, s$

for each $j \in 1: \ell_o$

for each $i \in 0: \ell_s$

$\#(o_j, s_i) += p((j, i)|o, s)$ (using (12))

$[M]$

for each $s \in V_s \cup \{\text{NULL}\}$

for each $o \in V_o$

$tr(o|s) = \frac{\#(o, s)}{\sum_o \#(o, s)}$
Further details

from the above outline to real code is a fairly short distance

1. the formula for \( p((j, i)|o, s) \) is
\[
\frac{t(o|s)}{\sum_{i'=0}^{t(o|s')}}
\]
and the denominator stays the same as \( i \) is varied in \( p((i, j)|o, s) \), so this denominator should be calculated once for each \( j \)

2. likewise in the M step, in
\[
\frac{\#(o,s)}{\sum_{o'} \#(o',s)}
\]
the denominator stays the same as \( o \) is varied, so this denominator should be calculated once for each \( s \)

Example One

Assuming a corpus of 2 pairs:

| s1 | la maison | o1 | the house |
| s2 | la fleur  | o1 | the flower |

initialising all \( tr(o|s) \) to uniformly to \( \frac{1}{2} \), the evolution of looks like this:

<table>
<thead>
<tr>
<th>Obs</th>
<th>Src</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
<th>final</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>la</td>
<td>0.33</td>
<td>0.5</td>
<td>0.6</td>
<td>0.69</td>
<td>0.77</td>
<td>0.84</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>house</td>
<td>la</td>
<td>0.33</td>
<td>0.25</td>
<td>0.2</td>
<td>0.15</td>
<td>0.11</td>
<td>0.081</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>flower</td>
<td>la</td>
<td>0.33</td>
<td>0.25</td>
<td>0.2</td>
<td>0.15</td>
<td>0.11</td>
<td>0.081</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>the</td>
<td>maison</td>
<td>0.33</td>
<td>0.5</td>
<td>0.43</td>
<td>0.36</td>
<td>0.3</td>
<td>0.24</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>house</td>
<td>maison</td>
<td>0.33</td>
<td>0.5</td>
<td>0.57</td>
<td>0.64</td>
<td>0.7</td>
<td>0.76</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>flower</td>
<td>maison</td>
<td>0.33</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>the</td>
<td>fleur</td>
<td>0.33</td>
<td>0.5</td>
<td>0.43</td>
<td>0.36</td>
<td>0.3</td>
<td>0.24</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>house</td>
<td>fleur</td>
<td>0.33</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>flower</td>
<td>fleur</td>
<td>0.33</td>
<td>0.5</td>
<td>0.57</td>
<td>0.64</td>
<td>0.7</td>
<td>0.76</td>
<td>1.00</td>
<td></td>
</tr>
</tbody>
</table>

Example Two (Koehn p92)

assuming a corpus of 3 pairs

| s1 | das Haus | s2 | das Buch | s3 | ein Buch |
| o1 | the house | o2 | the book | o3 | a book |

initialising all \( t(o|s) \) uniformly to 0.25, evolution of \( t(o|s) \) is

<table>
<thead>
<tr>
<th>Obs</th>
<th>Src</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
<th>final</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>das</td>
<td>0.25</td>
<td>0.5</td>
<td>0.6364</td>
<td>0.7479</td>
<td>...</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>book</td>
<td>das</td>
<td>0.25</td>
<td>0.25</td>
<td>0.1818</td>
<td>0.1208</td>
<td>...</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>house</td>
<td>das</td>
<td>0.25</td>
<td>0.25</td>
<td>0.1818</td>
<td>0.1313</td>
<td>...</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the</td>
<td>buch</td>
<td>0.25</td>
<td>0.25</td>
<td>0.1818</td>
<td>0.1208</td>
<td>...</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>book</td>
<td>buch</td>
<td>0.25</td>
<td>0.5</td>
<td>0.6364</td>
<td>0.7479</td>
<td>...</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>buch</td>
<td>0.25</td>
<td>0.25</td>
<td>0.1818</td>
<td>0.1313</td>
<td>...</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>book</td>
<td>ein</td>
<td>0.25</td>
<td>0.5</td>
<td>0.4286</td>
<td>0.3466</td>
<td>...</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>ein</td>
<td>0.25</td>
<td>0.5</td>
<td>0.5714</td>
<td>0.6534</td>
<td>...</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the</td>
<td>haus</td>
<td>0.25</td>
<td>0.5</td>
<td>0.4286</td>
<td>0.3466</td>
<td>...</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>house</td>
<td>haus</td>
<td>0.25</td>
<td>0.5</td>
<td>0.5714</td>
<td>0.6534</td>
<td>...</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>