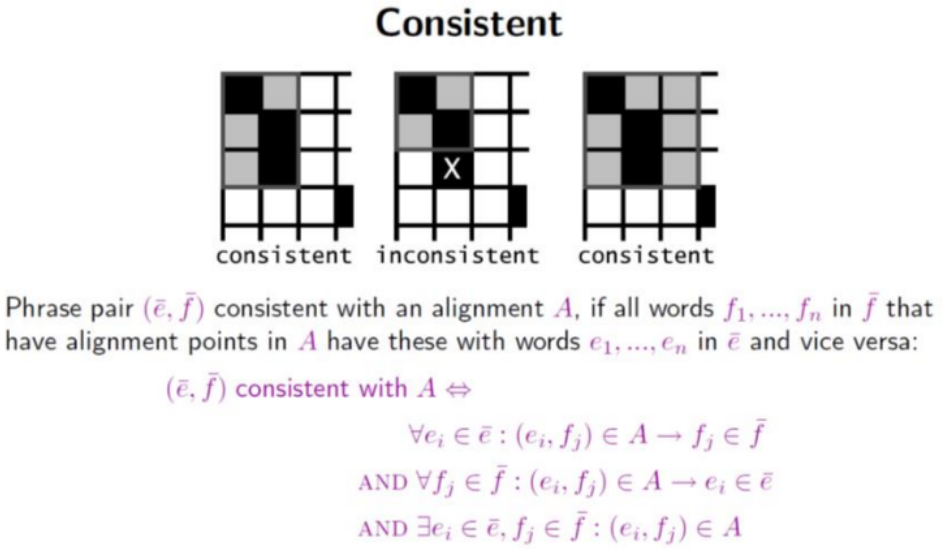
Machine Translation: Summer Term 2021

PB-SMT

Sangeet Sagar (7009050)
sasa00001@stud.uni-saarland.de

July 9, 2021

1. **Phrase Extraction: in your own words, explain the formal definition of phrase pair $(\bar{e}, \bar{f})$ being consistent with an alignment A below:**



The definition or the mathematical formulation given above says that for all the English words $\forall e_i$ in the English phrase $\bar{e}$, all the alignment points A that exists for that English word, there has to be a French word f_j in that French phrase $\bar{f}$.

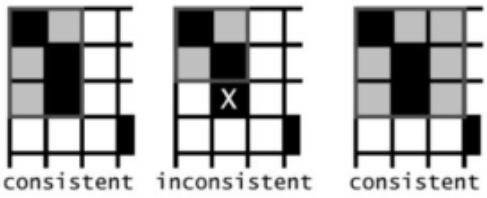
Its inverse states that for all the French words $\forall f_i$ in the French phrase $\bar{f}$, all the alignment points A that exists for that French word, there has to be a English word e_i in that English phrase $\bar{f}$.

$$\exists e_i \exists \bar{e}, f_j \exists \bar{f} : (e_i, f_j) \exists A$$

It says that it has to exist at least one English word e_i and foreign word f_j in the alignment.

2. **Phrase Extraction: given the following definitions of a phrase pair $(\bar{e}, \bar{f})$ being consistent with an alignment A:**

Consistent



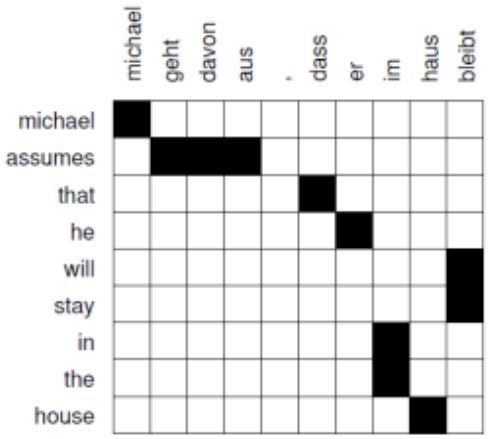
Phrase pair $(\bar{e}, \bar{f})$ consistent with an alignment A, if all words $f_1, ..., f_n$ in $\bar{f}$ that have alignment points in A have these with words $e_1, ..., e_n$ in $\bar{e}$ and vice versa:

$(\bar{e}, \bar{f})$ consistent with $A \Leftrightarrow$

$$\forall e_i \in \bar{e} : (e_i, f_j) \in A \rightarrow f_j \in \bar{f}$$
$$\text{AND } \forall f_j \in \bar{f} : (e_i, f_j) \in A \rightarrow e_i \in \bar{e}$$
$$\text{AND } \exists e_i \in \bar{e}, f_j \in \bar{f} : (e_i, f_j) \in A$$

and the word alignments below:

Word Alignment



which of the following phrase pairs are consistent with the alignment, which are not?

- (michael assumes , michael geht davon aus) → consistent
- (michael assumes , michael get davon aus) → not consistent
- (michael assumes , michael geht davon aus , dass) → not consistent
- (he will stay , er im Haus bleibt) → not consistent
- (he will stay in the house , er im Haus bleibt) → consistent
- (stay in the house , im Haus bleibt) → consistent

3. **Scoring phrases: given a phrase pair $(\bar{e}, \bar{f})$, how can you estimate $P(\bar{e}, \bar{f})$ from data using**

MLE and counts?

$$\begin{aligned} P(\bar{e}, \bar{f}) &= P(\bar{e}|\bar{f}) \cdot P(\bar{f}) \\ &= P(\bar{e}) \frac{P(\bar{f}|\bar{e})}{P(\bar{f})} \cdot P(\bar{f}) \\ &= P(\bar{e}) P(\bar{f}|\bar{e}) \\ &= P(\bar{e}) \frac{\text{count}(\bar{e}, \bar{f})}{\sum_{\bar{f}_i} \text{count}(\bar{e}, \bar{f})} \end{aligned}$$

4. **PB-SMT: draw a map of basic PB-SMT: which is the translation, the reordering and the language model in the formula below:**

$$e_{\text{best}} = \text{argmax}_e \prod_{i=1}^{I} \phi(\bar{f}_i|\bar{e}_i)\, d(start_i - end_{i-1} - 1) \prod_{i=1}^{|\mathbf{e}|} p_{LM}(e_i|e_1 ... e_{i-1})$$

$$\begin{aligned} \text{translation parameter} &: \phi(\bar{f}_i, \bar{e}_i) \\ \text{reordering parameter} &: d(start_i - end_{i-1} - 1) \\ \text{language model} &: \prod_{i=1}^{|e|} P_{LM}(e_i|e_1, e_2, \cdots e_{i-1}) \end{aligned}$$

5. **PB-SMT: in your own words, what are the main differences between IBM Model 3 and basic PB-SMT?**

$$P(a, f|e) = \binom{m - \varphi_0}{\varphi_0} \times p_0{}^{(m-2\varphi_0)} \times p_1{}^{\varphi_0} \times \prod_{i=1}^{l} n(\varphi_i|e_i) \times \prod_{j=1}^{m} t\left(f_j \middle| e_{a_j}\right) \times \prod_{j:a_j \neq 0}^{m} d(j|a_j, l, m) \times \prod_{i=0}^{l} \varphi_i! \times \frac{1}{\varphi_0!}$$

$$e_{\text{best}} = \text{argmax}_e \prod_{i=1}^{I} \phi(\bar{f}_i|\bar{e}_i)\, d(start_i - end_{i-1} - 1) \prod_{i=1}^{|\mathbf{e}|} p_{LM}(e_i|e_1 ... e_{i-1})$$

Think about: words, phrases, NULL, fertility, what are the independence assumptions in each?

6. **Logarithms: in your own words, explain** $\log_a(b)$**. What happens to probabilities in log-space? What are** $\log(1)$ **and** $\log(0)$**? If you want to maximize a probability, what do you have to do with the corresponding log, what would you have to do with the corresponding negative of the** log**? Can you express the log of a product as a sum? What is** $\log(x^y)$ **and why? Given** $\log_e(x)$ **what is its inverse function?**

In log-space, the product of the probabilities can be expressed as a sum of probabilities and they can be represented on the logarithmic scale instead of the log scale. Log probabilities are thus practical for computations and have an intuitive interpretation. $\log(1)$ is 0 and $\log(0)$ is not defined.

To maximize a log-probability, we compute the gradient. For a negative-log, we need to minimize it. Given that: $X = \{x_1, x_2, \ldots, x_N\}$

$$p(X \mid \Theta) = \prod_{i=1}^{N} p(x_i \mid \Theta)$$

$$\log p(X \mid \Theta) = \sum_{i=1}^{N} \log p(x_i \mid \Theta)$$

$\log(x^y) = x \cdot \log(y)$
Inverse of $\log_e^x = e^y$

7. **PB-SMT: in your own words, how are the following related:**

$$e_{\text{best}} = \text{argmax}_e \prod_{i=1}^{I} \phi(\bar{f}_i|\bar{e}_i)\ d(start_i - end_{i-1} - 1) \prod_{i=1}^{|\mathbf{e}|} p_{LM}(e_i|e_1 \ldots e_{i-1})$$

$$e_{\text{best}} = \text{argmax}_e \prod_{i=1}^{I} \phi(\bar{f}_i|\bar{e}_i)^{\lambda_\phi}\ d(start_i - end_{i-1} - 1)^{\lambda_d} \prod_{i=1}^{|\mathbf{e}|} p_{LM}(e_i|e_1 \ldots e_{i-1})^{\lambda_{LM}}$$

$$p(x) = \exp \sum_{i=1}^{n} \lambda_i h_i(x)$$

$$\begin{aligned} p(e, a|f) = \exp(\lambda_\phi & \sum_{i=1}^{I} \log\ \phi(\bar{f}_i|\bar{e}_i) + \\ & \lambda_d \sum_{i=1}^{I} \log\ d(a_i - b_{i-1} - 1) + \\ & \lambda_{LM} \sum_{i=1}^{|\mathbf{e}|} \log\ p_{LM}(e_i|e_1 \ldots e_{i-1})) \end{aligned}$$

The first two equations are very similar with slight differences.

1. Equation 1 represents an advanced modeling technique. The model consists of three sub-models phrase translation model, the reordering model, and the language model (LM is not a probabilistic model). The best translation is one that maximizes these three sub-models together. However, some sub-models may be more important than others. This issue is addressed in Equation 2.
2. In order to favor any sub-model because they may be more important than others in preserving some information, we introduce weights $\lambda_\phi, \lambda_d, \lambda_{LM}$ for each of these models. For each probability that we get we take it to the power of the weight.
3. Equation 3, represents the log-linear model. Product of probabilities sometimes introduces underflow errors and is not so easy to interpret. In a log-space, these probabilities can be expressed as sum and are therefore computationally efficient and easy to derive intuition from.
4. Equation 4 is the weighted model of the phrase-based model in log space. It's merely an expansion of Equation 3. This is useful because, in log-space, we get a separate expression for each of these sub-models and these can be computed efficiently and independently.

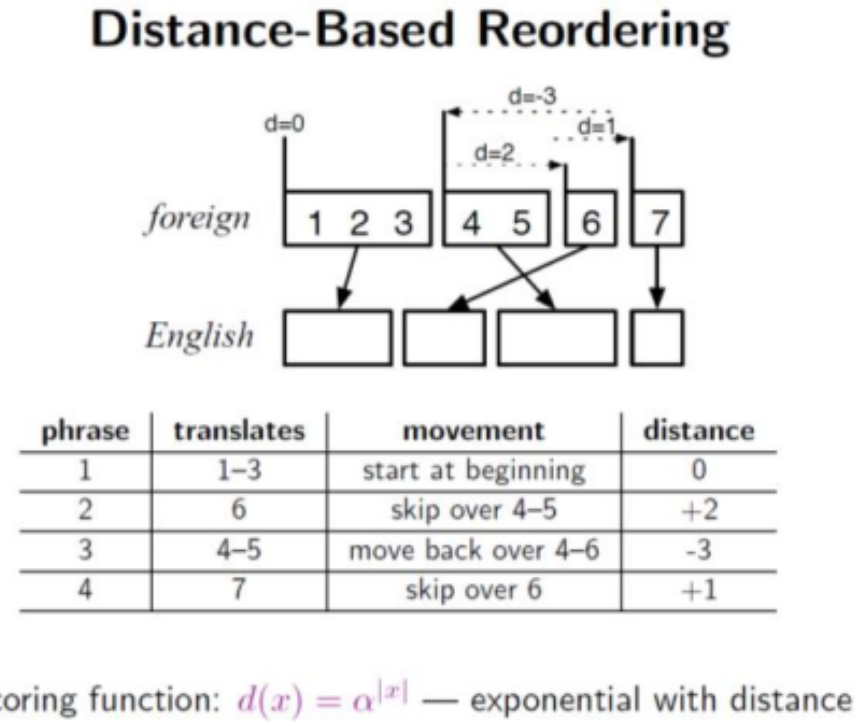
8. **Distance-based reordering: given the following simple definition of PB-SMT:**

$$e_{\text{best}} = \text{argmax}_e \prod_{i=1}^{I} \phi(\bar{f}_i|\bar{e}_i)\, d(start_i - end_{i-1} - 1) \prod_{i=1}^{|\mathbf{e}|} p_{LM}(e_i|e_1 ... e_{i-1})$$

with simple distance based reordering:

$$XX$$

in your own words please describe:

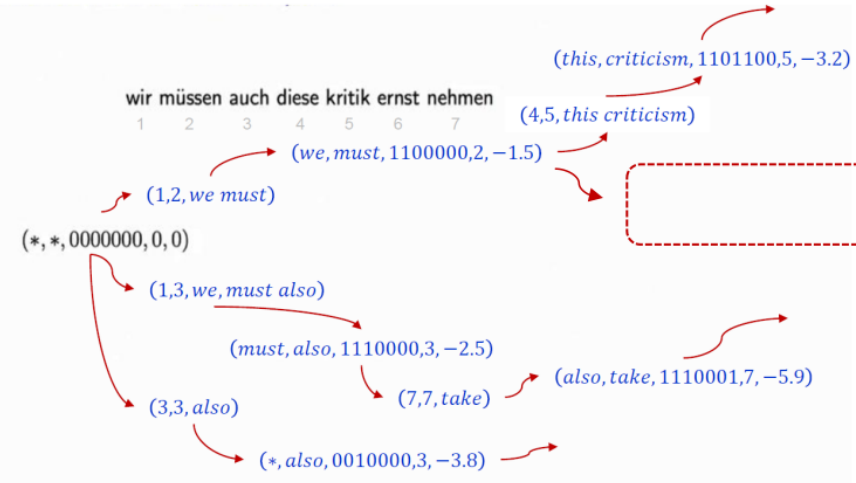


phrase	translates	movement	distance
1	1–3	start at beginning	0
2	6	skip over 4–5	+2
3	4–5	move back over 4–6	-3
4	7	skip over 6	+1

Scoring function: $d(x) = \alpha^{|x|}$ — exponential with distance

Here we have a French sentence being mapped into an English sentence. For each French phrase, we compute how much was it reordered i.e. **distance**. We start at the beginning and there is no re-ordering at the beginning so phrase 1 that translates from 1,2,3 have 0 distance. Now phrase 2 translates from 6 and to reorder it we must skip over 4-5 this traveling a distance of +2 (4,5,6,7 all move by +2 units). Then we make up for this in phrase 3 that translates from 4-5 by going over 4-6. This involves shifting back by -3 units (since we had moved +2 units further away, we must go back). Finally, we jump +1 for phrase 3 that translates from 7(XXX Doubt: the last step is not clear). Next, we define a scoring function $d(x) = \alpha^{|x|}$ where alpha is some kind of factor that penalizes the re-ordering and what that alpha is set to.

9. **PB-SMT decoder: please extend the following partial decoder graph at the position indicated in the red dashed rectangle below**



with $(3, 3, also)$ and compute the next state $(\_, \_, \_, \_, \_)$. (You can make up the cost score) In your own words, what information do the slots in state quintuples $(\_, \_, \_, \_, \_)$ capture? What would state representation tuples look like if instead of a 3-gram LM we had used a 5-gram LM?

1. Lets start with the initial state $(*, *, 0000000, 0, 0)$ which is of the form (e_1, e_2, b, r, α). Since we are in initial state $e_1 = e_2 = *$ and $r = 0$. We have b (the bit string) and since the given string is of length 7, $b = 0000000$.
2. We move to the next phrase which would of the form (s, t, e) (Where s is the index at the beginning of the phrase, t is the index at the end of the phrase and e is the phrase itself (English)). We observe from the phrase table that *wir müssen* correspond to *we must* and are at index $1, 2$ (they begin at 1 and end at 2). So the next phrase is $(1, 2,$ we must$)$.
3. Now we update the last state as $(we, must, 1100000, 2, -1.5)$. Lets understand this:
   - we is the beginning of the phrase so $e_1 = we$
   - $must$ is the end of the phrase so $e_2 = must$
   - Since 1^{st} and 2^{nd} index has been updated with the phrase derived above, the bit string becomes 1100000. Here 11 represents the update made at the indices 1 and 2.
   - Since end point of the current translation stops at position 2 so $r = 2$.
   - $\alpha = -1.5$ is the score associated with this state.

We repeat step 2 with a new phrase and update our state as explained in step 3. We are now in a position to compute the score for this state as updated in step 3.

$$0 + \log p(we \mid *, *) + \log p(we|*, *) + \log p(must|*, we) + g(1, 2, wemust) + (\eta \times \mid 0 + 1 - 1 \mid)$$

- 0: score of the initial state
- $\log p(we|*, *), \log p(must|*, we)$: translation probability.
- $g(1, 2, wemust)$: log probability of the phrase that we used.
- η: η is the distortion parameter. $\mid 0 + 1 - 1 \mid$ is of the general form $r + 1 - s$, where r is the end of the last string which is 0 (since this state was updated from the initial state), s is position at the beginning of the string.

Information stored in state quintuples

$$(e_1, e_2, b, r, \alpha)$$

1. e_1: starting word of the English phrase.
2. e_2: end word of the English phrase.
3. b: bit string.

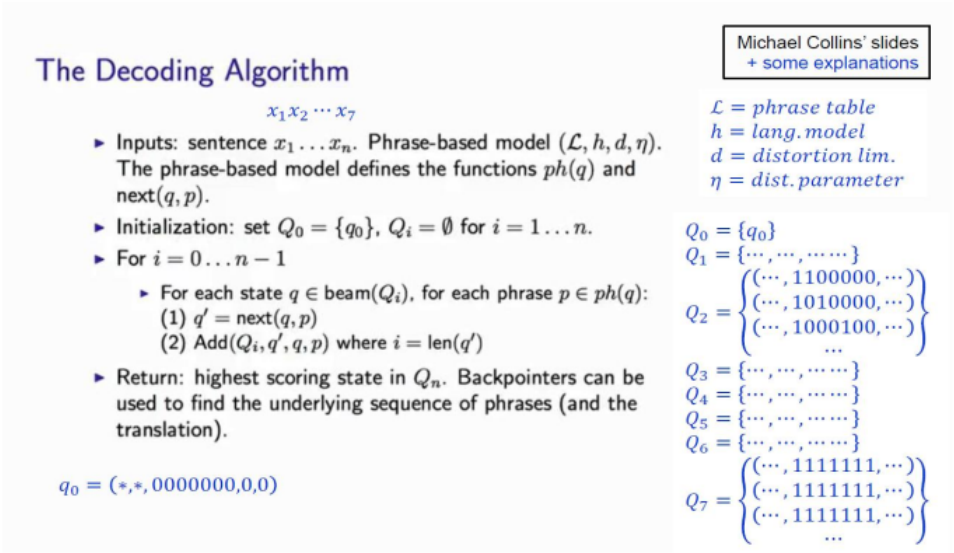
4. r: position (corresponding to the given German string) at which the end point of the current translation stops.
5. α: score associated with the state (usually negative).

We begin with $(*, *, 0000000, 0, 0)$ and the next phrase is $(3, 3, also)$. Next state is

1. e_1: $*$
2. e_2: $also$
3. b: 0001000
4. r: 3
5. α: -3.0 (we make up any score)

$(*, also, 0010000, 3, -3.0)$.

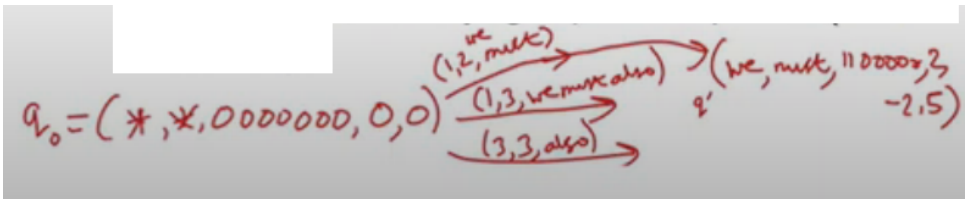
10. **PB-SMT Decoder: in your own words, please describe how the sets Q_i (in blue on the right) evolve during decoding, given the decoder pseudo code on the left:**



In particular, what does index i in Q_i capture? Let our input be a sentence $x_1, x_2, \cdots, x_7$ and a phrase model represented as $(\mathfrak{L}, h, d, \eta)$ where $\mathfrak{L}$ is the lexicon (or phrase table), h is the language model, d is the distortion limit and η is the distortion parameter. Given a state q we have a set of admissible phrases $ph(q)$ and $next(q, p)$ to be a state formed by combining state q with phrase p.

The algorithm initializes 8 sets for an input of 7 words with the first set $Q_0 = q_0$ being the starting state of the decoding algorithm. The other sets $Q_1, Q_2, \cdots, Q_7$ are a collection of states consisting of the number of words that were translated. E.g. Q_1 consists of all the states in which only one word was translated, Q_2 consists of all the states in which two words were translated, and so on.

The algorithm states from the first set and successively generates sets $Q_1, Q_2, \cdots, Q_7$. The algorithm begins a loop $i = 0 \cdots n-1$ where for each value of i we look at every possible states in Q_i. Let's begin with Q_0 where we look at only one state q_0 and then we explore every possible phrase $p \in ph(q)$ that can be formed with this state (see picture below).



For each possible phrase we compute the next state $q\prime = next(q, p)$ and add that state to the appropriate Q_i. For example, if we start with the phrase $(1, 2, wemust)$, the state is updated to $q\prime = (we, must, 1100000, 2, -2.5)$ and this state gets added to Q_2 because only 2 words were translated in the state $q\prime$. So Q_2 would consist of all states with the restriction that each state has only 2 words translated. The idea of the beam search algorithm here is to group together states that have the same number of words translated. The *Add* function $Add(Q_i, q\prime, q, p)$ is going to add some state $q\prime$ to Q_i where $i = len(q\prime)$ which is nothing but the number of words translated in $q\prime$.
We follow the same procedure with Q_1 where we find one or more states and find all the phrases that can be formed with this state. We further generate the next state and concatenate it to the next set. So this way we traverse through each state and we populate more and more states into each set i.e. Q_i's.

Finally, we return the highest-scoring state in Q_n or Q_7 in our case and this corresponds to the highest scoring translation found by the beam search algorithm. Further, we use backpointer to find the underlying sequence of phrases and the translation.

11. **PB-SMT Decoder: please explain why we use beam search in the decoder? Can you describe two forms of beam search?**

References