CS447: Natural Language Processing — Fall 2015

Homework 4

Handed Out: 11/12/2015 — *Due: 10pm, 12/9/2015*

- Please submit an archive of your solution (including code) on Compass by 10:00pm on the due date.
- **Note that this assignment is due on Wednesday, 12/9/2015** (the last day of classes).
- Please document your code where necessary.

Getting started

The source code and handout for the assignment are contained in a tarball that you can get from:

`http://courses.engr.illinois.edu/cs447/HW/cs447_hw4.tar.gz`

You need to unpack this tarball (`tar -xzvf cs447_hw4.tar.gz`) to get a directory that contains the code and data you will need for this homework.

See section 1.4.3 for instructions on how to download additional data for the discussion questions on Compass.

IBM word alignment

1.1 Goal

Your tasks for this assignment are to implement and train the IBM Model 1 for word alignment (see Lecture 23) on a parallel corpus of movie subtitles and to find the best alignment for a set of test sentence pairs.

It may be helpful to start with the Appendix for a extended review of IBM Model 1, and to familiarize yourself with the notation used in this handout.

1.2 Data

The homework release contains a small corpus for the testing script, `eng-spa_small.txt`. The target language for translation is English. Later, you will download larger parallel corpora in the language(s) of your choice to train your model in full.

1.3 Provided code

We have provided functionality in `hw4_translate.py` to get you started on your solution. An instance of `IBMModel1` is initialized with a text file containing a parallel training corpus. The text file is split up into pairs of unaligned sentences (English followed by non-English). The training corpus is stored in the following data structures (using the provided `initialize` method):

`fCorpus`: a list of foreign (e.g. Spanish) sentences.

`tCorpus`: a list of target (e.g. English) sentences (the sentence at position i in `tCorpus` is a translation of the sentence at position i in `fCorpus`).

`trans`: a map of frequency counts; trans$[e_x][f_y]$ is initialized with a count of how often target word e_x and source word f_y appear together.

You may define and use any other data structures you need (`hw4_translate.py` contains the distribution classes from previous homeworks, if you choose to use them).

1.4 What you need to implement

Your task is to finish implementing the methods in the `IBMModel1` class. You should initialize any additional data structures you need in the `__init__` method.

1.4.1 Part 1: Training the model (5 points)

You need to train your model using the EM algorithm by implementing the submethods in `trainUsingEM`. Treat each English sentence as if it begins with a `NULL` word in position 0 (this is already implemented), and refer to the Appendix for notation in these descriptions.

`computeTranslationLengthProbabilities()`:

Compute the translation length probabilities $q(m|n)$ from the corpus. These probabilities aren't necessary for calculating an alignment between two given sentences, but they would be necessary in order to produce a target sentence from a source sentence.

`getTranslationLengthProbability(n, m)`:

Return the pre-computed translation length probability $q(m|n)$, where `n` is the length of a sentence in the target language (English) and `m` is the length of a sentence in the source language (non-English).

`initializeWordTranslationProbabilities()`:

For each English word that occurs in the corpus (including the `NULL` word), choose initial values for the translation probabilities $p_t(\mathbf{f}|\mathbf{e})$[1]: Each distribution $p_t(...|e_x)$ should be initialized as a uniform distribution over all the words f_y that occur in the source translation of at least one of the sentences in which e_x occurs.

`getWordTranslationProbability(f_y, e_x)`:

Return the (current) value of the translation probability $p_t(f_y|e_x)$, where `f_y` is a word in the source language and `e_x` is a word in the target language.

`computeExpectedCounts()`:

For each pair of sentences $(\mathbf{f}^{(s)}, \mathbf{e}^{(s)})$, $1 \leq s \leq S$ with $\mathbf{f}^{(s)} = f_1^{(s)}...f_m^{(s)}$, and $\mathbf{e}^{(s)} = e_0^{(s)}...e_n^{(s)}$, compute the expected counts for the translation probabilities. That is, for each English word $e_i^{(s)}$ with $0 \leq i \leq n$ and for each source word $f_j^{(s)}$ with $1 \leq j \leq m$:

$$\langle c(f_j^{(s)}|e_i^{(s)}; \mathbf{f}^{(s)}, \mathbf{e}^{(s)})\rangle = \frac{p_t(f_j^{(s)}|e_i^{(s)})}{p_t(f_j^{(s)}|e_0^{(s)}) + \cdots + p_t(f_j^{(s)}|e_i^{(s)}) \cdots + p_t(f_j^{(s)}|e_n^{(s)})} \tag{1}$$

`updateTranslationProbabilities()`:

For each English word e_x that appears at least once in our corpus:

- Compute a normalization factor $Z_{e_x} = \sum_y \sum_{s=1}^{S} \langle c(f_y|e_x; \mathbf{f}^{(s)}, \mathbf{e}^{(s)})\rangle$
- For each source word f_y that appears in at least one of the sentence pairs in which e_x occurs, compute a new $p_t(f_y|e_x)$:

$$p_t(f_y|e_x) = \frac{\sum_{s=1}^{S} \langle c(f_y|e_x; \mathbf{f}^{(s)}, \mathbf{e}^{(s)})\rangle}{Z_{e_x}} \tag{2}$$

[1] More explanation of our notation can be found in the Appendix (specifically the "Notation" and "Model parameters" sections).

1.4.2 Part 2: Finding the best alignment (3 points)

Your second task is to find the best word alignment $\mathbf{a} = a_1...a_m$ for a pair of sentences $(\mathbf{f}^{(s)}, \mathbf{e}^{(s)})$. Using Equations 8-10, the best alignment is:

$$\mathbf{a}*_{(\mathbf{f}^{(s)},\mathbf{e}^{(s)})} = \text{argmax}_{\mathbf{a}} P(\mathbf{f}^{(s)}, \mathbf{a}|\mathbf{e}^{(s)}) = \text{argmax}_{\mathbf{a}} \prod_j p_t(f_j^{(s)}|e_{a_j}^{(s)}) \quad (3)$$

You need to finish implementing the `align` method:

`align(fSen, tSen)`:

> Return the best alignment between `fSen` and `tSen`, according to your model. The alignment must be returned as a list of integers where the j^{th} integer indicates the alignment for the j^{th} word in the foreign sentence `fSen` (the list should have the same length as `fSen` has words). A value of 0 means that the foreign word is aligned with the `NULL` token of the target sentence `tSen`, a value of 1 means that it is aligned with the first word of the sentence (`tSen`[1]), etc.

1.4.3 Part 3: Evaluating the alignments (2 points)

The third part of the assignment requires you to train and evaluate your model on a larger corpus and answer a set of discussion questions on Compass, available in the "Homework 4 Discussion Questions" test.

We provide a number of parallel corpora on the course website in the same format as `eng-spa_small.txt`:

- `eng-fra.txt.zip`: English/French
- `eng-ger.txt.zip`: English/German
- `eng-spa.txt.zip`: English/Spanish
- `eng-chi.txt.zip`: English/Chinese (produced using automatic word segmentation)
- `eng-hin.txt.zip`: English/Hindi
- `all_corpora.zip`: contains each of the above corpora.

The corpora are available in the `http://courses.engr.illinois.edu/cs447/HW/hw4_corpora/` directory; you can access the files by appending the filename to that address. For example, the English/French corpus is located at `http://courses.engr.illinois.edu/cs447/HW/hw4_corpora/eng-fra.txt.zip`.

Each of these corpora contain 150K parallel sentences; the French, German, Spanish, and Chinese corpora are transcribed from proceedings of the United Nations, while the Hindi corpus is a translation of the Quran downloaded from the Tanzil project. The individual compressed files are about 15-20MB each; the `all_corpora.zip` file is 85MB.

Though we have tried to provide a selection of languages, we recognize that you may want to choose a different language. If so, we recommend looking at the OPUS project[2] to find additional corpora to use in the part of the assignment (you may need to do some pre-processing to get these corpora into the same format).

Please download at least one of the corpora, run your model (train and produce alignments using `trainUsingEM()` and `generateAndSaveAlignments()`, respectively), and provide responses the following questions on Compass:

1. Which parallel corpus (or corpora) are you investigating? If you choose to train/evaluate on corpora beyond the ones we supplied, please provide a link to the data as well. (*0.0 pts*)

[2] `http://opus.lingfil.uu.se/`

2. Please copy and paste at least 10 sample alignments that your model produces when run on the corpus. (*0.5 pts*)
3. Please copy and paste three alignments that contain errors; for each sentence, explain the error that is made. At least two of these errors must be an alignment error, rather than a major difference in translation. (*1.0 pts*)
4. Please conclude with a brief summary (4-6 sentences) of your implementation of the IBM Model 1. You may use the following questions as prompts, but you are welcome to present other thoughts as well: (*0.5 pts*)
   - Is it usable as a translation system?
   - Given the errors made by your model, what changes would you make to improve its performance?
   - What sort of data would be useful for training a good translation model?

1.4.4 Extra Credit: Convergence (+2 points extra credit)

During training, the provided code for `trainUsingEM` has you repeat `computeExpectedCounts` and `updateTranslationProbabilities` for 20 iterations (by default). Instead, you should really iterate until the probabilites have converged. You could check convergence by computing the log likelihood of the corpus:

$$\begin{aligned} L(\mathcal{C}) &= S^{-1} \sum_s \log p(\mathbf{f}^{(s)}|\mathbf{e}^s) \\ &= S^{-1} \sum_s \log \left(\frac{q(m|n)}{(n+1)^m} \prod_{j=1}^{m} \sum_{i=0}^{n} p(f_j^{(s)}|e_i^{(s)}) \right) \\ &\propto \sum_s \log(q(m|n)) - m\log(n+1) + \sum_{j=1}^{m} \left(\log \sum_{i=0}^{n} p(f_j^{(s)}|e_i^{(s)}) \right) \end{aligned}$$

The model has fully converged when the change between log-likelihood between each iteration of training becomes zero (approximate this by stopping when log-likelihood decreases by an amount less than ϵ). You should experiment with different values of ϵ; if the value is too large, your training will abort before convergence. If the value is too small, a lot of time will be spent on the latter iterations without meaningfully altering the parameters.

You will get two bonus points if you compute log likelihood and run until convergence using the specified value for `convergenceEpsilon`. Your code should use the flag `useConvergenceEpsilon` to choose how many iterations to run; if the flag is `False` (as it is by default), then `trainUsingEM` should run a fixed `numIterations` of EM; if the flag is `True`, then the convergence criterion should be used to determing the number of iterations.

In order for us to grade your extra credit, please submit a second copy of your code under the Compass assignment "Homework 4 Extra Credit".

1.5 Additional hints

1.5.1 Test script

As in previous assignments, we provide a script for testing your implementation. The test script `hw4_test.py` uses the English-Spanish corpus `eng-spa_small.txt` to train your model over 20 iterations of EM, and then compares its results with a set of reference alignments (`reference_alignments.txt`). The reference alignments were produced by the course staff implementation under the same conditions as the test script; however, there are many rare words in the dataset and so tiebreaking may cause a valid implementation's accuracy to be below 100% (there will be no penalty in this case).

The test script validates your entire implementation; we do not provide unit tests for individual methods. We highly recommend debugging individual methods as you implement them.

1.5.2 PyPy

Training and evaluating your model on the large parallel corpora will take a considerable amount of time (and memory). One way to speed up the execution of your code is to use PyPy[3] (even using PyPy, our reference implementation takes several minutes to complete an iteration of EM). If your implementation is still too slow, we recommend testing your algorithms using a subset of the corpus and then coming to office hours for help.

1.6 What you will be graded on

The grade for your implementation of IBM Model 1 will be based on the following rubric:

1.6.1 Part 1 (5 points)

1 point: Correctly implementing the translation length probability $q(m|n)$ in `computeTranslationLengthProbabilities`, evaluated by calling `getTranslationLengthProbability` before the first iteration of EM.

1 point: Initializing the word translation probabilities uniformly in `initializeWordTranslationProbabilities`, evaluated by calling `getWordTranslationProbability` before the first iteration of EM.

2 points: Correctly implementing the EM algorithm (`computeExpectedCounts` and `updateTranslationProbabilities`). Since it is unlikely that you will be able to accurately align the sentences without implementing EM training correctly, this is a very important part of the assignment.

1 point: Efficiency (using PyPy, we should be able to run your code on the small testing corpus to train for 20 iterations of EM and generate the set of alignments in under one minute).

1.6.2 Part 2 (3 points)

For Part 2, your grade will depend on the agreement between the alignments predicted by your model and those predicted by the reference implementation (as we don't have gold human-annotated alignments). Accuracy will be mapped onto a grade of 0-3 points (due to tiebreaking within the model, you won't need 100% accuracy to get a perfect score, but other than that we won't be able to provide any additional details on the grade mapping).

1.6.3 Part 3 (2 points)

For Part 3, the grade for the discussion questions will be broken up using the values described in section 1.4.3 and on Compass. When analyzing the alignment errors, the grade will be split into 0.25 points total for the examples chosen (i.e., choose at least two alignment errors rather than just translation errors) and 0.25 points for each of your explanations of the errors (0.75 points total).

1.6.4 Extra credit (+2 points extra credit)

You can receive up to two points extra credit for correctly implementing the log-likelihood stopping criterion in `trainUsingEM`.

[3] `http://pypy.org`

1.7 What to submit

1.7.1 Parts 1 & 2

For Parts 1 & 2, the only file you need to submit is your completed `hw4_translate.py` program. You need to submit your solution as a compressed tarball on Compass; to do this, **save your file in a directory called *abc123*_cs447_hw4** (where *abc123* is your NetID) and then **create the archive from its parent directory** (`tar -czvf abc123_cs447_hw4.tar.gz abc123_cs447_hw4`). Please include the file containing your code for Parts 1 & 2:

`hw4_translate.py`: your Python implementation of IBM Model 1.

Upload this file under the "Homework 4 MP (Parts 1 & 2)" assignment on Compass.

1.7.2 Part 3

For Part 3, enter your answers to the discussion questions on Compass under the "Homework 4 Discussion Questions (Part 3)" test. You may submit multiple times, but only your last attempt will be graded. Please save a local copy of your answers in case you need to revise and re-submit later.

1.7.3 Extra Credit

If you choose to complete the extra credit from section 1.4.4, please submit a second copy of your code (**in the same tarball/zip format as for Parts 1 & 2**) to the "Homework 4 Extra Credit" assignment on Compass.

Appendix: A longer explanation of Model 1

To help understand what you have to do (and why), we provide a more detailed review of IBM Model 1 in this section. Don't worry – it is not as complicated as it looks at first!

Notation

Throughout this assignment, we'll be translating from a source language $\mathbf{f}$ (non-English) to the target language $\mathbf{e}$ (English).

We'll use $f_y \in \mathbf{f}$ to denote a word in the vocabulary of the source language, and $e_x \in \mathbf{e}$ to denote a word in the vocabulary of the target language (e_x and f_y are both word *types*).

The training corpus $\mathcal{C}$ consists of S sentence pairs $(\mathbf{f}^{(s)}, \mathbf{e}^{(s)})$, $1 \leq s \leq S$. Thus, $\mathbf{f}$ is a vocabulary/set of word types, and $\mathbf{f}^{(s)}$ is a sentence/list of word *tokens* (the same goes for $\mathbf{e}$ and $\mathbf{e}^{(s)}$).

We'll use m to denote the length of source sentence and n to denote the length of a parallel target sentence.

For each pair of sentences $(\mathbf{f}^{(s)}, \mathbf{e}^{(s)})$ in the training corpus, $\mathbf{f}^{(s)} = f_1^{(s)}....f_m^{(s)}$ and $\mathbf{e}^{(s)} = e_0^{(s)}e_1^{(s)}...e_n^{(s)}$. That is, $f_j^{(s)}$ is the j^{th} word in source sentence $\mathbf{f}^{(s)}$, $e_i^{(s)}$ is the i^{th} word in source sentence $\mathbf{e}^{(s)}$, and $e_0^{(s)}$ = `NULL` for all s.

When defining probabilities for a single sentence, we may drop the superscript (s); however, the subscript indices (j and i) should still be consistent with the previous definition.

Given an alignment $\mathbf{a}$, $a_j = i$ when word f_j in the target sentence is aligned with word e_i in the target sentence (when this holds, we can also refer to e_i as e_{a_j}).

Model parameters

Recall that Model 1 consists of the following distributions:

- **Length probabilities $q(m|n)$:** the probability that the non-English sentence is of length m given that the English sentence is of length n. You can obtain these probabilities simply by counting the lengths of the sentence pairs in the training data.
- **Alignment probabilities $p(\mathbf{a}|n, m)$:** the probability of alignment $\mathbf{a}$, given that the English sentence is of length n, and the non-English sentence is of length m. Recall that an alignment specifies for each position $1....m$ in the non-English sentence which English word (`NULL` or $e_1, ...e_n$) it is aligned to. That is, we have $1 + n$ choices for each of the m words. In Model 1, *all alignments have the same probability*; the probability that word f_j is aligned to word e_i is $\frac{1}{n+1}$, and so the probability of one particular alignment vector is simply $\mathbf{a} = a_1, ...a_m$ is $\frac{1}{(n+1)^m}$.
- **Translation probabilities** $p_t(f_y|e_x)$: the probability that the English word e_x is translated into ("generates") the non-English word f_y. We'll refer to this family of distributions as $p_t(\mathbf{f}|\mathbf{e})$, and we will use the EM algorithm to estimate these probabilities.

Training: estimating the translation probabilities $p_t(f_y|e_x)$ with the EM algorithm

Since Model 1 is so simple, we only need to learn (estimate) the translation probabilities $p_t(f_y|e_x)$. If the corpus was annotated with alignments, we could use standard relative frequency estimation: that is, we could simply count how often we see word e_x aligned to word f_y in our corpus (let us call this count $c(f_y, e_x)$), and divide this count by the count of how often we see word e_x aligned to any word f_z in our corpus:

$$p_t(f_y|e_x) = \frac{c(f_y, e_x)}{\sum_z c(f_z, e_x)} \qquad \text{(relative frequency estimate)} \tag{4}$$

However, our corpus consists only of unaligned sentence pairs, so we will have to replace the actual frequencies $c(f_y, e_x)$ with *expected* frequencies $\langle c(f_y, e_x)\rangle$:

$$p_t(f_y|e_x) = \frac{\langle c(f_y, e_x)\rangle}{\sum_z \langle c(f_z, e_x)\rangle} \qquad \text{(expected relative frequency estimate)} \tag{5}$$

Here, $\langle c(f_y, e_x)\rangle$ indicates how often we expect to see word e_x aligned to word f_y in our corpus. Since our corpus consists of many sentence pairs, $\langle c(f_y, e_x)\rangle$ is the sum of how often we expect to see word e_x aligned to word f_y in each of the S sentence pairs in our corpus. That is, if we write $\langle c(f_y, e_x|\mathbf{f}^{(s)}, \mathbf{e}^{(s)})\rangle$ for the number of times we expect to see word e_x aligned to word f_y in the s-th sentence pair, then:

$$\langle c(f_y, e_x)\rangle = \sum_{s=1}^{S} \langle c(f_y, e_x|\mathbf{f}^{(s)}, \mathbf{e}^{(s)})\rangle \tag{6}$$

So, how do we compute $\langle c(f_y, e_x|\mathbf{f}^{(s)}, \mathbf{e}^{(s)})\rangle$ for a given sentence pair $(\mathbf{f}^{(s)}, \mathbf{e}^{(s)})$? Obviously, $\mathbf{f}^{(s)}$ has to contain the word f_y and $\mathbf{e}^{(s)}$ has to contain the word e_x for this count to be non-zero, but let us assume this is the case. For simplicity let us also assume that f_y occurs at position j in $\mathbf{f}^{(s)}$ (i.e. $f_j^{(s)} = f_y$), and that e_x occurs at position i in $\mathbf{e}^{(s)}$ (i.e. $e_i^{(s)} = e_x$).

Word $e_i^{(s)}$ is aligned to $f_j^{(s)}$ if the alignment of f_j is i, i.e. if $a_j = i$. That is, we need to compute $P(a_j = i \mid \mathbf{e}^{(s)}, \mathbf{f}^{(s)})$, i.e. the probability of $a_j = i$, given $\mathbf{e}^{(s)}$ and $\mathbf{f}^{(s)}$. Let $\mathcal{A}$ be the set of all possible alignments of $\mathbf{f}^{(s)}$ and $\mathbf{e}^{(s)}$, and $\mathcal{A}_{a_j=i}$ be the set of all alignments in which $a_j = i$. Then,

$$P(a_j = i \mid \mathbf{e}^{(s)}, \mathbf{f}^{(s)}) = \frac{\sum_{\mathbf{a}\in\mathcal{A}_{a_j=i}} P(\mathbf{a} \mid \mathbf{e}^{(s)}, \mathbf{f}^{(s)})}{\sum_{\mathbf{a}'\in\mathcal{A}} P(\mathbf{a}' \mid \mathbf{e}^{(s)}, \mathbf{f}^{(s)})} \tag{7}$$

It turns out that this simplifies to the (current) translation probability of $f_j^{(s)}$ given $e_i^{(s)}$, divided by the sum of the (current) translation probabilities of $f_j^{(s)}$ given any word in $\mathbf{e}^{(s)}$ (including `NULL`):

$$P(a_j = i \mid \mathbf{e}^{(s)}, \mathbf{f}^{(s)}) = \frac{p_t(f_j^{(s)}|e_i^{(s)})}{\sum_{i'=0}^{n^{(s)}} p_t(f_j^{(s)}|e_{i'}^{(s)})} \tag{8}$$

This is what you will have to compute at each iteration when you train Model 1.

Why is this the correct thing to do? Let us look at how we compute $P(\mathbf{a} \mid \mathbf{e}^{(s)}, \mathbf{f}^{(s)})$. The probability model defines the probability of sentence $\mathbf{f}^{(s)} = f_1^{(s)}....f_m^{(s)}$ and alignment $\mathbf{a} = a_1...a_m$ given $\mathbf{e}^{(s)} = e_0^{(s)}e_1^{(s)}...e_n^{(s)}$ as

$$\begin{aligned} P(\mathbf{f}^{(s)}, \mathbf{a}|\mathbf{e}^{(s)}) &= \underbrace{q(m|n)}_{\text{length of } \mathbf{f}} \underbrace{p(\mathbf{a}|n,m)}_{\text{alignment}} \underbrace{\prod_{j=1.}^{m} p_t(f_j^{(s)}|e_{a_j}^{(s)})}_{\text{translation}} \\ &= \underbrace{q(m|n)}_{\text{length of } \mathbf{f}} \underbrace{\frac{1}{(n+1)^m}}_{\text{alignment}} \underbrace{\prod_{j=1.}^{m} p_t(f_j^{(s)}|e_{a_j}^{(s)})}_{\text{translation}} \end{aligned} \tag{9}$$

But since $\mathbf{f}^{(s)}$ is given, we need to know $P(\mathbf{a}|\mathbf{f}^{(s)}, \mathbf{e}^{(s)})$, which we can compute as follows (here $\mathcal{A}$ is the set of all possible alignments of $\mathbf{f}^{(s)}$ and $\mathbf{e}^{(s)}$, and we omit superscripts for $f_j^{(s)}$ and $e_i^{(s)}$ for legibility):

$$\begin{aligned} P(\mathbf{a}|\mathbf{f}^{(s)}, \mathbf{e}^{(s)}) &= \frac{P(\mathbf{a}, \mathbf{f}^{(s)}|\mathbf{e}^{(s)})}{\sum_{\mathbf{a}' \in \mathcal{A}} P(\mathbf{a}'|\mathbf{f}^{(s)}, \mathbf{e}^{(s)})} \\ &= \frac{q(m|n)\frac{1}{(n+1)^m}\prod_{j=1.}^{m} p_t(f_j|e_{a_j})}{\sum_{\mathbf{a}' \in \mathcal{A}} q(m|n)\frac{1}{(n+1)^m}\prod_{j=1.}^{m} p_t(f_j|e_{a'_j})} \\ &= \frac{q(m|n)\frac{1}{(n+1)^m}\prod_{j=1.}^{m} p_t(f_j|e_{a_j})}{q(m|n)\frac{1}{(n+1)^m}\sum_{\mathbf{a}' \in \mathcal{A}}\prod_{j=1}^{m} p_t(f_j|e_{a'_j})} \\ &= \frac{\prod_{j=1}^{m} p_t(f_j|e_{a_j})}{\sum_{\mathbf{a}' \in \mathcal{A}}\prod_{j=1}^{m} p_t(f_j|e_{a'_j})} \end{aligned} \tag{10}$$

That is, all we really need to know are the translation probabilities $p_t(f_y|e_x)$ for the words in $\mathbf{e}^{(s)}$ and $\mathbf{f}^{(s)}$. And since in Model 1, the different positions of $\mathbf{a}$ are filled independently, we don't even need to compute the probability of each indvidual alignment in order to compute $P(a_j = i \mid \mathbf{e}^{(s)}, \mathbf{f}^{(s)})$. Instead, we just consider word $f_j^{(s)}$, and that gives us equation 8.