CS447: Natural Language Processing	Fall 2018
Homework 4	
Handed Out: 11/09/2018	*Due: 11:59pm, 12/07/2018*

Please submit an archive of your solution (including code) on Compass by 11:59pm on the due date. Please document your code where necessary.

Getting started

All files that are necessary to do the assignment are contained in a tar ball which you can get from:

`http://courses.engr.illinois.edu/cs447/HW/cs447_HW4.tar.gz`

You need to unpack this tar ball (`tar -xzvf cs447_HW4.tar.gz`) to get a directory that contains the code and data you will need for this homework.

IBM word alignment (10 points)

1.1 Goal

Your tasks for this assignment are to implement and train the IBM Model 1 for word alignment (see Lecture 22) 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 two parallel corpora, `eng-spa.txt` (English-Spanish) and `eng-ger.txt` (English-German). In both cases, the target language for translation is English. We also provide a smaller corpus for the testing script, `eng-spa_small.txt`.

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.

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 (7 points, +2 EC)

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.[1]

`getTranslationLengthProbability`:

> Return the pre-computed translation length probability $q(m|n)$.

`initializeWordTranslationProbabilities`:

> For each English word that occurs in the corpus (including the `NULL` word), choose initial values for the translation probabilities $p_t(\mathbf{f}|\mathbf{e})$: 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. (You could also use a uniform distribution over all words in the corpus, but that would require considerably more memory).

`getWordTranslationProbability`:

> Return the (current) value of the translation probability $p_t(f_y|e_x)$.

`computeExpectedCounts`:

> For each pair of sentences $(\mathbf{f}^{(s)}, \mathbf{e}^{(s)})$, $1 \leq s \leq S$ with $\mathbf{f}^{(s)} = f_1^{(s)}...f_m$, and $\mathbf{e}^{(s)} = e_0^{(s)}...e_n$, 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 $p_t(f_y|e_x)$:
>
> $$p_t(f_y|e_x) = \frac{\sum_{s=1}^{S} \langle c(f_y|e_x; \mathbf{f}^{(s)}, \mathbf{e}^{(s)})\rangle}{Z_{e_x}} \tag{2}$$

`writeModel`:

> Write the parameters of your model to a file in a legible format. You should be able to examine these files to compare different set of parameters (e.g. right after initialization, after training, or in between traiing iterations) and see where your model is improving.

[1] If we were using a better language model, we could try to generate a translation; because Model 1 treats alignment probabilities uniformly, the resulting sentence would be word salad.

复习完课程之后再加EC

Convergence (+2 EC) During training, the provided code has you repeat `computeExpectedCounts` and `updateTranslationProbabilities` for 10 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).[2] You will get two bonus points if you compute log likelihood and run until converence (mention this in your README).

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:

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

You need to finish implementing the `align` method: `align`:

> 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 on the sentence (`tSen`[1]), etc.

1.4.3 Test script

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).

1.5 What you will be graded on

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

3 points: Accuracy of your model's alignments vs. the course staff's implementation (as we don't have gold human-annotated alignments).

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] 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

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 (we should be able to run your code on a full corpus in under 20 minutes).

2 points: Quality of README discussion (see below) and clarity of distribution output from `writeModel`.

+2 points EC: Implementing the log-likelihood stopping criterion in `trainUsingEM`.

In your README, discuss where your model improved improved on the initial parameters during training (e.g. on rare words, function words, punctuation, phrases, etc.), and examine some of the alignments that your model produces (especially on sentences longer than 5 words). Is it a usable translation system? What kinds of systematic errors does your model make? Are these errors due to (a lack of) data, or are they a result of assumptions built into the model? If the latter, what changes would you make to the model to improve its performance?

1.6 What to submit

The only file you need to submit for this part is your completed `hw4_translate.py` program. You should submit your solution as a compressed tarball on Compass; to do this, **save your files 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 following files:

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

`README.txt`: a text file explaining your system and summarizing your findings.

1.7 Other language pairs

If you want to see how your model performs on other languages, you can download more language pairs from `http://opus.lingfil.uu.se/OpenSubtitles_v2.php` (the lower triangle of this matrix has plain text files). NB: I know many of you speak Chinese, but the Chinese text on this site is not word-segmented, so that may be an issue. Feel free to try anyway. Note that these corpora come as two separate files, one per language.

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 **f** (non-English) to the target language **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, **f** is a vocabulary/set of word types, and $\mathbf{f}^{(s)}$ is a sentence/list of word *tokens* (the same goes for **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, $\mathbf{f^{(s)}} = f_1^{(s)}....f_m$ and $\mathbf{e^{(s)}} = e_0^{(s)} e_1^{(s)} ... e_n$. That is, $f_j^{(s)}$ is the jth word in source sentence $\mathbf{f}^{(s)}$, $e_i^{(s)}$ is the ith word in source sentence $\mathbf{e}^{(s)}$, and $e_0^{(s)} = \texttt{NULL}$ 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 fmaily 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 $\langle c(f_y, e_x|\mathbf{f}^{(\mathbf{s})}, \mathbf{e}^{(\mathbf{s})})\rangle$ for a given sentence pair $(\mathbf{f}^{(\mathbf{s})}, \mathbf{e}^{(\mathbf{s})})$? Obviously, $\mathbf{f}^{(\mathbf{s})}$ has to contain the word f_y and $\mathbf{e}^{(\mathbf{s})}$ 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 $\mathbf{f}^{(\mathbf{s})}$ (i.e. $f_j^{(s)} = f_y$), and that e_x occurs at position i in $\mathbf{e}^{(\mathbf{s})}$ (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 $P(a_j = i \mid \mathbf{e}^{(\mathbf{s})}, \mathbf{f}^{(\mathbf{s})})$, i.e. the probability of $a_j = i$, given $\mathbf{e}^{(\mathbf{s})}$ and $\mathbf{f}^{(\mathbf{s})}$. Let $\mathcal{A}$ be the set of all possible alignments of $\mathbf{f}^{(\mathbf{s})}$ and $\mathbf{e}^{(\mathbf{s})}$, and $\mathcal{A}_{a_j=i}$ be the set of all alignments in which $a_j = i$. Then,

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

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 $\mathbf{e}^{(\mathbf{s})}$ (including `NULL`):

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

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 $P(\mathbf{a} \mid \mathbf{e}^{(\mathbf{s})}, \mathbf{f}^{(\mathbf{s})})$. The probability model defines the probability of sentence $\mathbf{f}^{(\mathbf{s})} = f_1^{(s)}....f_m$ and alignment $\mathbf{a} = a_1...a_m$ given $\mathbf{e}^{(\mathbf{s})} = e_0^{(s)}e_1^{(s)}...e_n$ as

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

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

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

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.