Computer Science 401 — 11 February 2019
St. George Campus — University of Toronto

Homework Assignment #2
Due: Friday, 8 March 2019 at 19h00 (7 PM),

Statistical Machine Translation

TA: Mohamed Abdalla (mohamed.abdalla@mail.utoronto.ca); Raeid Saqur (raeidsaqur@cs.toronto.edu); Charlie Zhang (charliezcy.zhang@mail.utoronto.ca)

1 Introduction

This assignment will give you experience in working with n-gram models, smoothing, and statistical machine translation through word alignment. Knowledge of French is not required.

Your tasks are to build bigram and unigram models of English and French, to smooth their probabilities using add-δ discounting, to build a world-alignment model between English and French using the IBM-1 model, and to use these models to translate French sentences into English with a decoder that we provide.

The programming language for this assignment is Python3.

2 Background

Canadian Hansard data

The main corpus for this assignment comes from the official records (*Hansards*) of the 36^{th} Canadian Parliament, including debates from both the House of Representatives and the Senate. This corpus is available at `/u/cs401/A2_SMT/data/Hansard/` and has been split into `Training/` and `Testing/` directories.

This data set consists of pairs of corresponding files (`*.e` is the English equivalent of the French `*.f`) in which every line is a sentence. Here, sentence alignment has already been performed for you. That is, the n^{th} sentence in one file corresponds to the n^{th} sentence in its corresponding file (e.g., line n in `fubar.e` is aligned with line n in `fubar.f`). Note that this data only consists of sentence pairs; many-to-one, many-to-many, and one-to-many alignments are not included.

Furthermore, for the purposes of this assignment we have filtered this corpus down to sentences with between approximately 4 and 15 tokens to simplify the computational requirements of alignment and decoding. We have also converted the file encodings from ISO-8859-1 to ASCII so as to further simplify the problem. This involved *transliterating* the original text to remove *diacritics*, i.e., accented characters (e.g., *Chrétien* becomes *Chretien*).

To test your code, you may like to use the samples provided at `/u/cs401/A2_SMT/data/Toy/`.

Add-δ smoothing

Recall that the maximum likelihood estimate of the probability of the current word w_t given the previous word w_{t-1} is

$$P(w_t \,|\, w_{t-1}) = \frac{Count\,(w_{t-1}, w_t)}{Count\,(w_{t-1})}. \tag{1}$$

$Count\,(w_{t-1}, w_t)$ refers to the number of times the word sequence $w_{t-1}w_t$ appears in a training corpus, and $Count(w_{t-1})$ refers to the number of times the word w_{t-1} appears in that corpus.

Laplace's method of add-1 smoothing for n-grams simulates observing otherwise unseen events by providing probability mass to those unseen events by discounting events we have seen. Although the simplest of all smoothing methods, in practice this approach does not work well because too much of the n-gram probability mass is assigned to unseen events, thereby increasing the overall entropy unacceptably.

Add-δ smoothing generalizes Laplace smoothing by adding δ to the count of each bigram, where $0 < \delta \leq 1$, and normalizing accordingly. This method is generally attributed to G.J. Lidstone[1]. Given a known vocabulary $\mathcal{V}$ of size $\|\mathcal{V}\|$, the probability of the current word w_t given the previous word w_{t-1} in this model is

$$P(w_t \,|\, w_{t-1}; \delta, \|\mathcal{V}\|) = \frac{Count\,(w_{t-1}, w_t) + \delta}{Count\,(w_{t-1}) + \delta\|\mathcal{V}\|}. \tag{2}$$

3 Your tasks

1. Preprocess input text [5 marks]

First, implement the following Python function:

```
preprocess(in_sentence, language)
```

that returns a version of the input sentence `in_sentence` that is more amenable to training. For both languages, separate sentence-final punctuation (sentences have already been determined for you), commas, colons and semicolons, parentheses, dashes between parentheses, mathematical operators (e.g., +, -, <, >, =), and quotation marks. Certain contractions are required in French, often to eliminate vowel clusters. When the input `language` is 'french', separate the following contractions:

Type	**Modification**	**Example**
Singular definite article (*le*, *la*)	Separate leading **l'** from concatenated word	*l'election* ⇒ *l' election*
Single-consonant words ending in e-'muet' (e.g., 'dropped'-e *ce*, *je*)	Separate leading consonant and apostrophe from concatenated word	*je t'aime* ⇒ *je t' aime*, *j'ai* ⇒ *j' ai*
que	Separate leading **qu'** from concatenated word	*qu'on* ⇒ *qu' on*, *qu'il* ⇒ *qu' il*
Conjunctions *puisque* and *lorsque*	Separate following *on* or *il*	*puisqu'on* ⇒ *puisqu' on*, *lorsqu'il* ⇒ *lorsqu' il*

Any words containing apostrophes not encapsulated by the above rules can be left as-is. Additionally, the following French words should not be separated: *d'abord*, *d'accord*, *d'ailleurs*, *d'habitude*.

A template of this function has been provided for you at `/u/cs401/A2_SMT/code/preprocess.py`. Make your changes to a copy of this file and submit your version.

[1] Lidstone, G. J. (1920) Note on the general case of the Bayes-Laplace formula for inductive or *a priori* probabilities. *Transactions of the Faculty of Actuaries* 8:182–192.

2. Compute *n*-gram counts [15 marks]

Next, implement a function to simply count all unigrams and bigrams in the preprocessed training data, namely:

```
LM = lm_train(data_dir, language, fn_LM)
```

that returns a special language model structure (a dictionary), `LM`, defined below. This function trains on all of the data files in `data_dir` that end in either 'e' for English or 'f' for French (which is specified in the argument `language`) and saves the structure that it returns in the filename `fn_LM`.

The structure returned by this function should be called '`LM`' and must have two keys: '`uni`' and '`bi`', each of which holds structures (additional dictionaries) which incorporate unigram and bigram counts, respectively. The fieldnames (i.e. *keys*) to the '`uni`' structure are words and the values of those fields are the total counts of those words in the training data. The keys to the '`bi`' structure are words (w_{t-1}) and their values are dictionaries. The keys of those sub-dictionaries are also words (w_t) and the values of those fields are the total counts of '$w_{t-1}w_t$' in the training data.

E.g.,

```
>> LM['uni']['word'] = 5 % the word 'word' appears 5 times in training
>> LM['bi']['word']['bird'] = 2 % the bigram 'word bird' appears twice in training
```

A template of this function has been provided for you at `/u/cs401/A2_SMT/code/lm_train.py`. Note that this template calls `preprocess`.

Make your changes to a copy of the `lm_train.py` template and submit your version. Train two language models, one for English and one for French, on the data at `/u/cs401/A2_SMT/data/Hansard/Training/`. You will use these models for subsequent tasks.

3. Compute log-likelihoods and add-δ log-likelihoods [20 marks]

Now implement a function to compute the log-likelihoods of test sentences, namely:

```
logProb = lm_prob(sentence, LM, smoothing, delta, vocabSize) .
```

This function takes `sentence` (a previously preprocessed string) and a language model `LM` (as produced by `lm_train`). If the argument `smoothing` is (`'False'`), this function returns the maximum-likelihood estimate of the sentence. If the argument `type` is `'True'`, this function returns a δ-smoothed estimate of the sentence. In the case of smoothing, the arguments `delta` and `vocabSize` must also be specified (where $0 < \delta \leq 1$).

When computing your MLE estimate, if you encounter the situation where $\frac{Count(w_t w_{t+1})}{Count(w_t)} = 0/0$, then assume that the probability $P(w_{t+1} \mid w_t) = 0$ or, equivalently, $\log P(w_{t+1} \mid w_t) = -\infty$. Negative infinity in Python is represented by `float('-inf')`. **Use log base 2 (i.e. `log2()`).**

A template of this function has been provided for you at `/u/cs401/A2_SMT/code/log_prob.py`. Make your changes to a copy of the `log_prob.py` template and submit your version.

We also provide you with the function `/u/cs401/A2_SMT/code/perplexity.py`, which returns the perplexity of a test corpus given a language model. You do not need to modify this function. Using the language models learned in Task 2, compute the perplexity of the data at `/u/cs401/A2_SMT/data/Hansard/Testing/` for each language and for both the MLE and add-δ versions. Try at least 3 to 5 different values of δ according to your judgment. Submit a report, `Task3.txt`, which summarizes your findings. Your report can additionally include experiments on the log-probabilities of individual sentences.

4. Implement IBM-1 [25 marks]

Now implement the IBM-1 algorithm to learn word alignments between English and French words, namely:

```
AM = align_ibm1(train_dir, num_sentences, max_iter, fn_AM).
```

This function trains on the first `num_sentences` read in data files from `train_dir`. The parameter `max_iter` specifies the maximum number of times the EM algorithm iterates before being terminated. This function returns a specialized alignment model structure, `AM`, in which `AM['eng_word']['fre_word']` holds the probability (not log probability) of the word `eng_word` aligning to `fre_word`. In this sense, `AM` is essentially the t distribution from class, e.g.,

```
>> AM['bird']['oiseau'] = 0.8 % t(oiseau | bird) = 0.8
```

Here, we will use a simplified version of IBM-1 in which we ignore the NULL word and we ignore alignments where an English word would align with no French word, as discussed in class. So, the probability of an alignment A of a French sentence F, given a known English sentence E is

$$P(A, F \mid E) = \prod_{j=1}^{len_F} t(f_j \mid e_{a_j})$$

where a_j is the index of the word in E which is aligned with the j^{th} word in F and len_F is the number of tokens in the French sentence. Since we are only using IBM-1, we employ the simplifying assumption that every alignment is equally likely.

Note: The naïve approach to initializing `AM` is to have a uniform distribution over all possible English (`e`) and French (`f`) words, i.e., `AM['e']['f'] = 1/`$\|\mathcal{V}_F\|$, where $\|\mathcal{V}_F\|$ is the size of the French vocabulary. Doing so, however, will consume too much memory and computation time. Instead, you can initialize `AM['e']` uniformly over only those French words that occur in corresponding French sentences. For example, given only the training sentence pairs

the house	*la maison*
house of commons	*chambre des communes*
Andromeda galaxy	*galaxie d'Andromede*

, you would initialize the structure `AM['house']['la'] = 0.2`, `AM['house']['maison'] = 0.2`, `AM['house']['chambre'] = 0.2`, `AM['house']['des'] = 0.2`, `AM['house']['communes'] = 0.2`. There would be no probability of generating *galaxie* from *house*. **Note** that you can force `AM['SENTSTART']['SENTSTART'] = 1` and `AM['SENTEND']['SENTEND'] = 1`.

A template of this function has been provided for you at `/u/cs401/A2_SMT/code/align_ibm1.py`. You will notice that we have suggested a general structure of empty helper functions here, but you are free to implement this function as you wish, as long as it meets with the specifications above. Make your changes to a copy of the `align_ibm1.py` template and submit your version.

5. Translate and evaluate the test data [10 marks]

You will now produce your own translations, obtain reference translations from Google and the Hansards, and use the latter to evaluate the former, with a BLEU score. This will all be done in the file `evalAlign.py` (there is a very sparse template of this file at `/u/cs401/A2_SMT/code/`).

To decode, we are providing the function

```
english = decode(french, LM, AM),
```

at `/u/cs401/A2_SMT/code/decode.py`. Here, `french` is a preprocessed French sentence, `LM` and `AM` are your English language model from Task 2 and your alignment model trained from Task 4, respectively, and

`lmtype`, `delta`, and `vocabSize` parameterize smoothing, as before in Task 3. You do not need to change the `decode` function, but you may (see the Bonus section, below).

For evaluation, translate the 25 French sentences in `/u/cs401/A2_SMT/data/Hansard/Testing/Task5.f` with the `decode` function and evaluate them using corresponding reference sentences, specifically:

1. `/u/cs401/A2_SMT/data/Hansard/Testing/Task5.e`, from the Hansards.
2. `/u/cs401/A2_SMT/data/Hansard/Testing/Task5.google.e`, Google's translations of the French phrases[2].

To evaluate each translation, use the BLEU score from lecture 6, i.e.,

$$BLEU = BP_C \times (p_1 p_2 \dots p_n)^{(1/n)} \tag{3}$$

Repeat this task with at least four alignment models (trained on $1K$, $10K$, $15K$, and $30K$ sentences, respectively) and with three values of n in the BLEU score (i.e., $n = 1, 2, 3$). You should therefore have $25 \times 4 \times 3$ BLEU scores in your evaluation. Write a short subjective analysis of how the different references differ from each other, and whether using more than 2 of them might be better (or worse).

In all cases, you can use the MLE language model (i.e., specify `lmtype = ''`). Optionally, you can try additional alignment models, smoothed language model with varying δ, or other test data from other files in `/u/cs401/A2_SMT/data/Hansard/Testing/`.

Submit your evaluation procedure, `evalAlign.py`, along with a report, `Task5.txt`, which summarizes your findings. If you make any changes to any other files, submit those files as well.

Bonus [up to 15 marks]

We will give bonus marks for innovative work going substantially beyond the minimal requirements. Your overall mark for this assignment cannot exceed 100%.

You may decide to pursue any number of tasks of your own design related to this assignment, although you should consult with the instructor or the TA before embarking on such exploration. Certainly, the rest of the assignment takes higher priority. Some ideas:

- Try additional smoothing methods (e.g., Good-Turing, Knesser-Ney) and re-run the experiments in Task 3, above. Submit your code and an associated discussion.
- Implement the IBM-2 model of word-alignment, otherwise replicating Task 4 above. Ideally, translate the test data using this model and compute the error, as you did for Task 5. How does this model compare to IBM-1? Submit your code and an associated discussion.
- We have not considered the null word when performing alignments. Re-implement the IBM-1 alignment model to include null words and the possibility that no English word aligns with a French word (or vice versa). Submit your code and an associated discussion.
- Perform substantial data analysis of the error trends observed in each method you implement. This must go well beyond the basic discussion already included in the assignment. Submit a report.
- The decoder we use here is extremely simple and incomplete. You can write your own decoder that attempts to find $\hat{e} = \arg\max_e P(e \mid f)$ using a heuristic A^* search, for example. Alternatively, what happens if you weight the contributions of the alignment and the language model to the overall probability? Section 25.8 of the Jurafsky & Martin textbook offers some ideas on how to improve the decoder. Submit your code and an associated discussion, comparing the decoded results to those performed with the default decoder.

[2] See `https://developers.google.com/api-client-library/python/apis/translate/v2`, but be prepared to pay.

- Read the sequence-to-sequence tutorial at `https://www.tensorflow.org/tutorials/seq2seq` and apply it to these data. Is the performance significantly better (or different) than IBM-1 on these data?

- The website `https://www.allsides.com/unbiased-balanced-news` curates news articles on particular events or stories according to their perceived political bias, using the spectrum used in Assignment 1 (less 'alternative' news). We sampled a considerable amount of these data; they are available at `/s/course/csc401/A2/allSides` (1.6GB) for you to examine. Unfortunately, since a right-leaning report on a story is not strictly a translation of a right-leaning report (or vice versa), the normal approach to sentence alignment (or SMT generally) does not apply; in our experiments, performance was unacceptably random. You, however, may be more fortunate...

4 General specification

We will test your code on different training and testing documents in addition to those specified above. Where possible, do not hardwire directory names into your code. As part of grading your assignment, the grader will run your programs using test scripts. It is therefore important that each of your programs precisely meets all the specifications and formatting requirements, including program arguments and file names.

If a program uses a file or helper script name that is specified within the program, it must read it either from the directory in which the program is being executed, or it must read it from a subdirectory of `/u/cs401/` whose path is completely specified in the program. Do **not** hardwire the absolute address of your home directory within the program; the grader does not have access to this directory.

All your programs must contain adequate internal documentation to be clear to the graders. External documentation is not required.

4.1 Submission requirements

This assignment is submitted electronically. Submit your assignment on MarkUs. Do **not** `tar` or `compress` your files, and do not place your files in subdirectories. Do not format your discussion as a PDF or Word document — use plain text only. You should submit:

1. All your code for `preprocess.py`, `lm_train.py`, `lm_prob.py`, `align_ibm1.py`, and `evalAlign.py`, along with any other source files to which you made changes or which are necessary to run your code in Python3 on teach.cs.
2. Your alignment model trained on 1k sentences from `/u/cs401/A2_SMT/data/Hansard/Training/`, dumped in file `am.pickle`.
3. Your reports `Task3.txt` and `Task5.txt`.
4. Any material submitted towards a bonus mark. This should ideally be limited to code, results, and reports as text files.
5. Your `ID` file as described in Assignment 1. A template of `ID` is available on the course web page.

You do not need to hand in your language models or other temporary files.

5 Using your own computer

If you want to do some or all of this assignment on your laptop or other computer, you will have to do the extra work of downloading and installing the requisite software and data. You take on the risk that your computer might not be adequate for the task. You are strongly advised to upload regular backups of your work to teach.cs, so that if your machine fails or proves to be inadequate, you can immediately continue working on the assignment at teach.cs. When you have completed the assignment, you should try your programs out on teach.cs to make sure that they run correctly there. **A submission that does not work on teach.cs will get zero marks.**

6 Suggestions

This assignment uses a simplified version of an alignment model which itself makes several major simplifying assumptions and, as such, the results of the decoder will not be representative of the state-of-the-art in statistical machine translation. You will generally be marked on how well you understand the underlying concepts and algorithms. This approach was chosen for this assignment in order to give you a relative reprieve in the mid-term workload. However, if you have the time you are highly encouraged to pursue bonus work as indicated above. Exploring more complex models is not only interesting, but will give you a fuller perspective on the techniques used in machine translation.

The following dates are suggestions as to how to spread out the work for this assignment. These dates may not be applicable to you personally and they are not required deadlines. However, it's a good idea to try to spread things out so you don't have to rush at the end.

Task 1 18 February

Task 2 25 February

Task 3 1 March

Task 4 4 March

Task 5 8 March