Natural Language Processing, Homework 4 (15%)

Prof. Liang Huang

Due Monday Nov 11 at 11:59pm on Canvas (to HW4 groups)

This homework is about unsupervised learning (using the EM algorithm) of the alignment between English and Japanese phonemes in English-Katakana pairs. Recall from HW2/3:

```
AE K T ER          ;; English phoneme sequence for 'actor'
A K U T A A        ;; Same word, loaned into Japanese
1 2 2 3 4 4        ;; e.g., Japanese T maps to the 3rd English sound
```

But in this homework, you need to learn the alignments yourself. In fact, those alignments we gave you in HWs 2–3 were also learned by EM, and thus are *not* 100% correct. You will reuse the following files:

`epron-jpron.data`	a database of aligned English/Japanese phoneme sequence pairs ignore the aligments – your job is to learn them!
`eword.wfsa`	a unigram WFSA of English word sequences
`eword-epron.wfst`	a WFST from English words to English phoneme sequences
`jprons.txt`	a short list of Japanese Katakana sounds to decode

1 Complete Data, Incomplete Model (10 pts)

If we're given the following manually aligned pairs (English "boat" and "test"):

```
B OW T          T EH S T
B O O T O       T E S U T O
1 2 2 3 3       1 2 3 3 4 4
```

What is the maximum likelihood estimate for $p(\texttt{T} \mid \texttt{T})$ and $p(\texttt{T O} \mid \texttt{T})$?

2 Complete Model, Incomplete Data (10 pts)

Given the following conditional probabilities,

```
p(B | B) = 1
p(O O | OW) = 0.8   p(O | OW) = 0.2
p(T | T) = 0.3      p(T O | T) = 0.5   p(O | T) = 0.1   p(O T O | T) = 0.1
```

Enumerate (by hand) all legal alignments for the "boat" example above, and compute the probability and normalized probability of each alignment. Which one is the Viterbi alignment?

3 EM, Implementation I: Enumerating All Alignments (70 pts)

Now implement the EM algorithm:

```
1. initialize the conditional probabilities to uniform
2. repeat:
3.   E-step:
4.     for each English-Katakana pair:
5.       enumerate all legal alignments for that E-K pair, along with their respective probabilities
6.       renormalize the probabilities to get a distribution over these alignments
```

```
7.        collect the fractional counts for this E-K pair
8.   M-step: count-and-divide based on the collected fractional counts from all pairs to get new prob. table
9. until reached maximum iteration or converged
```

You should proceed with the following steps:

1. (15 pts) to start with, you should work on the simplest scenario ("wine" example): `W AY N / W A I N`.

   Print the following information in each iteration: 1) the corpus probability, 2) the new prob. table (>= 0.01). 3) the number of nonzero entries in the prob. table (>= 0.01). **Note:** the corpus probibility should increase in each iteration (we have the proof), and the number of nonzero entries should decrease (as the model gets peakier).

   For example, you should run `echo -e "W AY N\nW A I N\n" | ./em.py 5` and get the following (note that the data should be read in from standard input and 5 is the number of iterations):

   ```
   iteration 0  ----- corpus prob=  0.00411522633745
   AY|->   A: 0.33  A I: 0.33  I: 0.33
   W|->    W: 0.67  W A: 0.33
   N|->    N: 0.67  I N: 0.33
   nonzeros = 7

   iteration 1  ----- corpus prob=  0.296296296296
   AY|->   A I: 0.50  A: 0.25  I: 0.25
   W|->    W: 0.75  W A: 0.25
   N|->    N: 0.75  I N: 0.25
   nonzeros = 7
   ...
   iteration 4  ----- corpus prob=  0.912659654395
   AY|->   A I: 1.00
   W|->    W: 1.00
   N|->    N: 1.00
   nonzeros = 3
   ```

   For your debugging convenience you can also print all possible alignments and their unnormalized and normalized probabilities in each iteration. Include the result of 5 iterations in your report.

2. (5 pts) If you add `N AY N / N A I N` to it you should see it converge faster. Show the results.

3. (5 pts) Now that you have passed "wine", try the "test" example above (`T EH S T / T E S U T O`) and you'll see that EM converges to something wrong. Show me the result of 5 iterations.

   Now come up with a minimal second example so that when combined with the "test" example, EM will learn the correct alignments. This second pair has to be a real English word, and should be choosen from `epron-jpron.data`.

4. (8 pts) Now come up with an example dataset where EM does not learn anything and is still ambivalent at the end (i.e., the final probability table is (almost) the same as the initial one). In the lecture we gave `B IY / B I I` as one such example, but can you come up with something bigger, i.e., longer sequences and multiple, interdependent pairs?

5. (15 pts) If you have passed all previous questions, it's now time to work on the whole dataset with the following command-line:

   ```
   cat epron-jpron.data | ./em.py 15 >epron-jpron.probs 2>epron-jpron.logs
   ```

   (Ignore the alignment lines in the input – your job is to learn them yourself!)

   The logs file contains the logging info for all iterations (see above), and print the final prob. table to the `.probs` file. Theis file should follow the same format as in HW3 (ignore < 0.01 entries).

   Note: Do not reestimate probs from the final Viterbi alignments – you'll get HW3 probs that way. **Hint:** The corpus probs are likely to underflow (beyond the range of Python `float`). Try fix it.

6. (7 pts) Decode `jprons.txt` from HW3 using your newly learned `epron-jpron.probs`. Compare with your results from HW3: did you see any differences?

7. (5 pts) Generate Viterbi alignments from `epron-jpron.probs`:

   ```
   cat epron-jpron.data | ./viterbi_align.py epron-jpron.probs >epron-jpron.viterbi
   ```

   The result file should follow the same format `as epron-jpron.data` so that we can compare them.

8. (5 pts) Calculate the number of different viterbi alignments between your `epron-jpron.viterbi` and `epron-jpron.data`. It should be less than 10 (out of 2684 examples).

4 EM, Implementation II: Forward-Backward (DP) (30 pts)

The enumeration of all alignments is not the best idea, since in the worst case, there are exponentially many such alignments per example.

1. (5 pts) BTW how many possible alignments are there for a pair of n English phonemes and m Japanese phonemes ($n \leq m$)? What pair(s) in `epron-jpron.data` give the highest number of alignments?

2. (20 pts) Implement the forward-backward algorithm (replacing lines 5–7 in the pseudocode above) so that you can do dynamic programming to collect fractional counts without enumerating all alignments. Make sure the two approaches match on their results.

   Include a detailed pseudocode of your implementation. What is the complexity of this forward-backward algorithm for a (n, m) pair? Try optimize your program (`em2.py`). How does it compare to the enumeration approach in terms of speed? Part of the grade will depend on efficiency. (5 pts pseudocode/complexity, 10 pts correctness of implementation, 5 pts efficiency).

3. (5 pts) Try to construct an example so that `em2.py` is substantially faster than `em.py` and explain why.

5 EM for Decipherment (30 pts)

Can you decode this? Hint: letter substitution cipher (i.e., permutation of 27 chars).

```
gjkgcbkycnjpkovryrbgkcrvsczbkyhkahqvykgjvjkcpekrbkjxdjayrpmkyhkmhkyhkyvrcukrpkbjfjvcukihpygb
oqykcykujcbykhpjkejihavcyrakvjmrijkjxrbybkrpkjfjvzkiclhvkarfrurtcyrhpkhvkaquyqvjkrpkygjkshvue
auqobkrpkvjajpykzjcvbkgcfjkwhqpekohvcbkbhkerwwraquykyhkpjmhyrcyjksrygkygcykicpzkbgqpkgrbkaurjpyb
gjkbcrekygjkoqvjcqkiqbykgcfjkygjkerbdqyjkvjbhufjekozkwjovqcvzkrpkhvejvkyhkdjvwhvikygjkajpbqb
oqykygjkdqouraryzkbqvvhqperpmkygjkejocaujkajvycrpuzkbjvfjekyhkwhaqbkckbdhyurmgykhpkyghbjkjwwhvyb
```

(Wait: how come there is no space at all, given that space also participated in the permutation??)

Using a bigram model trained on Ex2's `train.txt` with 50 iterations should be good enough.
Hint: as shown in slides, start with a uniform model $p(c \mid e) = 1/27$ for all c, e: (the LM does not change!)

$$
\begin{aligned}
p(c_1 \dots c_n) &= \sum_{e_1 \dots e_n} p(c_1 \dots c_n, e_1 \dots e_n) = \sum_{e_1 \dots e_n} p(e_1 \dots e_n) \cdot p(c_1 \dots c_n \mid e_1 \dots e_n) && (1) \\
&= \sum_{e_1 \dots e_n} p(e_1 \dots e_n) \cdot p(c_1 \mid e_1) \cdots p(c_n \mid e_n) && (2) \\
&= \sum_{e_1 \dots e_n} p(e_1 \mid \texttt{<s>}) \cdots p(e_i \mid e_{i-1}) \cdots p(\texttt{</s>} \mid e_n) \cdot p(c_1 \mid e_1) \cdots p(c_n \mid e_n) && (3)
\end{aligned}
$$

After each iteration, print some debugging information like these (probs $\leq$ 0.01 are considered zeros):

```
$ cat cipher.txt | ./decipher2.py train.txt 50
epoch   1 logp(corpus)= -2270.05  entropy= 4.79 nonzeros= 567
...
epoch  50 logp(corpus)= -1606.14  entropy= 3.39 nonzeros= 76
```

notice the initial entropy is similar to 0-gram's, and the final entropy similar to 2-gram's (see LM slides). why?

Optional: Can you do trigram? (should be better: the stronger your LM is, the better you can decipher).