CS 753: Assignment #1

Preethi Jyothi
pjyothi@cse.iitb.ac.in

IIT Bombay — August 22, 2019

Instructions: This assignment is due on or before 11.55 pm on September 5, 2019. For each additional day after September 5 until September 7, there will be a 10% reduction in marks. The submission portal on Moodle will be closed after 11.55 pm on September 7. For the coding parts of this assignment, follow the exact directory structure specified in the question. Deviations from the specified format will be penalized.

1 Noisy Channel (8 points)

Let C be a channel which behaves as follows, when a sequence of symbols are sent through it: A symbol is erased (i.e. replaced with a symbol $\square$) with probability ϵ_0, unless the previous symbol was already erased, in which case the current symbol is erased with probability ϵ_1.

To make this channel more robust, an encoding scheme is devised: each symbol is encoded by repeating it twice, and while decoding, if at least one of the symbols is not erased, the original symbol can be recovered. If both symbols are erased, then the decoder also outputs the erasure symbol, $\square$. For example, the input sequence `iitb` is encoded as `iiiittbb`; if the output from the channel C is $\square$`i`$\square\square$`t`$\square$`bb` it is decoded as `i`$\square$`tb`.

Consider the system which takes as input a sequence of symbols $x_1 \ldots x_n$, encodes them as above into $2n$ symbols, passes the encoded string through C, decodes the output of C and outputs the resulting n-symbol string $y_1 \ldots y_n$. Note that for each i, $y_i \in \{x_i, \square\}$.

For the following questions, assume $\epsilon_0 = 1 - \epsilon_1$.

Question 1

1. What is the probability that $y_1 = \square$? **[2 points]**
2. What is the probability that $y_2 = \square$? **[2 points]**
3. Suppose the system has been running for a long time. What is the probability that the next decoded symbol is $\square$? More precisely, what is $\lim_{n\to\infty} \Pr[y_n = \square]$? **[4 points]**

2 Hidden Markov Models (16 points)

Consider an HMM $\lambda = (A, B)$ with a sequence of hidden states Q, a sequence of observations O, transition probabilities $a_{ij} = \Pr(q_t = j | q_{t-1} = i)$ and emission probabilities $b_j(o_t) = \Pr(o_t | q_t = j)$. The forward probability, $\alpha_t(j)$ and backward probability $\beta_t(j)$ for an observation sequence $\{o_1, \ldots, o_T\}$ of length T are defined as follows:

$$\alpha_t(j) = \Pr(o_1, \ldots, o_t, q_t = j) \quad (1)$$

$$\beta_t(j) = \Pr(o_{t+1}, \ldots, o_T | q_t = j) \quad (2)$$

Question 2

(a) Compute the following posterior probabilities using $\alpha_t(j)$, $\beta_t(j)$, a_{ij} and $b_j(o_t)$ **[4 points]**

(i) $\Pr(q_{t+1} = k | q_t = j, o_1, \ldots, o_T)$

(ii) $\Pr(q_{t-1} = i, q_t = j, q_{t+1} = k | o_1, \ldots, o_T)$

(b) Given an HMM and a sequence of observations $O = o_1, \ldots, o_T$, the most probable sequence of states $q_1, \ldots, q_T$ can be efficiently computed using the Viterbi algorithm in $O(N^2T)$ time where N is the size of the state space. Let us consider a specific HMM where $a_{ii} = p$, $a_{ij} = q \; \forall j \neq i$ and $p > q$. How can we modify the original Viterbi algorithm such that it runs in $O(NT)$ time? **[5 points]**

c) Modify the recurrence (for $v_t(j)$) in the original Viterbi algorithm so that it finds the best state sequence among all sequences in which there is at least one run of k consecutive occurences of the same state. **[7 points]**

3 FSTs for Pronunciation Modelling (26 points)

Note: You will use the OpenFst Toolkit for this question. You can refer to the OpenFst Quick Tour for a brief introduction to the OpenFst libraries.

A pronunciation dictionary or lexicon provides a mapping between words and their canonical pronunciations. When represented using weighted finite state transducers (WFSTs), lexicons are FSTs that transduce words to sequences of phones. Figure 1 shows a trivial FST that accepts two words, "hello → [h eh l ow]" and "world → [w er l d]".

Recall from the lectures that such a lexicon FST can be composed with other FSTs to create an ASR system. In this problem we shall use FSTs in a creative way for a slightly different purpose.

Download the following archive file into your working directory and untar it:

Command Line

```
$ tar -xzf assgmt1.tgz
```

Create a text file named `ANSWERS` within `assgmt1`. All the subsequent questions which require answers in text should be added to `ANSWERS`. Your final submission directory, `assgmt1.tgz`, should contain:

- `lex.txt`
- The `ANSWERS` file.
- All the scripts asked for in the subsequent questions.
- A `README` file describing any usage specifications for the scripts.

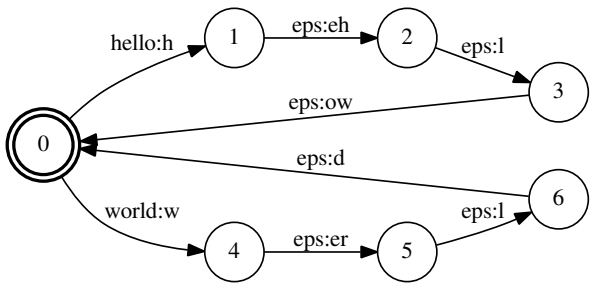

Figure 1: Simple lexicon FST.

3.1 Dictionary FST, L **[4 points]**

Create a dictionary FST (as in Figure 1) using all the words and pronounciations in `assgmt1/lex.txt`. Write a script that traverses through the word list in `assgmt1/lex.txt` and constructs an FST (say L) mapping each word to a sequence of phones. Given a word represented as an acceptor, `A`, `A` $\circ$ `L` should produce as output the phone sequence corresponding to the word (if the word exists in the dictionary; otherwise the output is a special symbol "<OOV>").

Command Line

```
$ chmod +x create-dict.sh
$ ./create-dict.sh lex.txt > L.fst
        $ ./lookup.sh L.fst ALICE
"a l I s
```

3.2 Letters instead of words **[4 points]**

Create another FST S which maps each word to its spelling, as a sequence of letters. Now, appropriately combine it with the FST L from the previous part (using OpenFST commands) to create a new FST Q which takes a sequence of letters of a single word as input and produces as output its pronunciation as a sequence of phones.

Command Line

```
$ chmod +x create-let-dict.sh
$ ./create-let-dict.sh lex.txt L.fst > Q.fst
```

3.3 Pronunciations for partial words **[8 points]**

When you manually inspect the file `lex.txt` you can often *align* the letters in a word with the phones in the pronunciation. This would allow you to produce the pronunciation for a partial word (i.e., the prefix of a word in the dictionary). For instance, given a prefix PRES, the following words match it:

Command Line

```
$ grep "^PRES" lex.txt
PRESENT p r e "s E n t
PRESENTERA p r e s e n "t e: r a
PRESIDENT p r e s I "d E n t
PRESIDENTEN p r e s I "d E n t e n
```

By inspection, you could assign the pronunciation `p r e s` to PRES. However, you are probably relying on your familiarity with the input and output alphabets for this.

Now, you should implement this heuristic as an FST QPrefix, by appropriately modifying the FST Q using OpenFST commands (possibly after other minor modifications). More precisely, given a prefix string w, QPrefix should find a pronunciation π so that there is a word of the form wz in the vocabulary with $\pi\rho$ as the pronunciation (assuming such a word exists). If there are multiple such pronunciations, QPrefix need not find all of them, and you may arbitrarily choose one that is maximally long. Here is an expected behaviour.

Command Line

```
$ ./create-prefix-dict.sh Q.fst > QPrefix.fst
        $ ./lookup.sh QPrefix.fst PRES
p r e s
```

3.4 Pronunciations for new words **[10 points]**

Now, suppose you are given a new word that is not in the vocabulary. Here is the outline of a heuristic-based algorithm to assign a pronunciation for this new word from existing in-vocabulary words (if possible):

1. Split the new word into every possible prefix+suffix split.
2. For each split:
   - Use the previous part to find a pronunciation for the prefix, if possible.
   - Analogously, find a pronunciation for the suffix, if possible.
   - If both the above steps succeed, merge the two pronunciations to derive a candidate pronunciation for the complete word.
3. If multiple pronunciations are derived for the new word, return one using any heuristic that you prefer.
4. If no pronunciations are found, return an appropriate message.

Here's a possible session with your algorithm (implemented in two stages, for setup and lookup).

Command Line

```
$ ./setup-merge.sh lex.txt fstdir
created FSTs QPrefix and QSuffix in fstdir/
        $ ./lookup-merge.sh fstdir PRESAKTIG
p r e s %a k t I g
```

3.5 Extra Credit: Improve it! **[up to 5 points]**

The heuristic from the previous problem can be very brittle. Can you identify instances where it does poorly, analyze how they arise, and come up with (WFST based) strategies to handle such issues?