CSME 206A – Natural Language & Speech Processing **Spring Semester**

Lecture 18: Statistical machine Translation-III

Lecturer: K.R. Chowdhary *: Professor of CS*

18.1 Training for Lexical Selection and Reordering

The Statistical machine translation can be formulated as a search for the best target sequence that maximizes $P(T|S)$, where S is the source sentence and T is the target sentence. Ideally, $P(T|S)$ should be estimated directly to maximize the conditional likelihood on the training data (i.e., discriminant model). However, T corresponds to a sequence with an exponentially large combination of possible labels, and traditional classification approaches cannot be used directly.

18.1.1 Global Lexical Selection Model

The *global lexical selection model* (GLSM) is based on discriminatively trained classification techniques. This modeling technique is a successful method for resolving ambiguity in speech and natural language processing, and it outperforms the *generative models.* This model also provide a framework for incorporating rich *morpho-syntactic* information.

The GLSM does not require the sentences to be word-aligned in order to learn the local associations. The sentence-aligned corpus is taken as before, but the target sentence is treated as a *bag of words* (BOWs) assigned to the source sentence. The goal is: given a source sentence S, estimate the probability of finding a given word (t_j) in its translation T, i.e., we need to estimate the probabilities $P(true|t_j, S)$ and $P(false|t_j, S)$ (see Fig. 18.1).

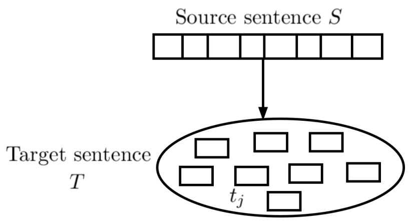

Figure 18.1: Bags of words: estimate probability of Word (i.e., t_j) given any source sentence S

To train such a model, we need to build *binary classifiers* for all the words in the target language vocabulary. The probability distributions of these binary classifiers are learned using *maximum entropy* model. For the word t_j, those training sentence pairs are considered as positive examples, where the word appears in the target, and negative other-wise. Thus, the number of training examples for each binary classifier equals the total number of training source sentences. The classifiers are trained using word n-grams of the source sentence as features, we call it as *BOG(S)* (Bag of grams, i.e., some n-grams).

18.1.2 Using Global Lexical Selection Model

For example, consider the following training data to train the GLSM of the kind, $S \rightarrow T$:

$$(1) s_1\ s_2\ s_3\ s_4\ s_5\ s_6\ s_7\ s_8 \Rightarrow\ t_1\ t_2\ t_{14}\ t_3\ t_5\ t_2\ t_{10}\ t_5\ t_{18}\ t_{14} \tag{18.1}$$
$$(2) s_3\ s_7\ s_9\ s_{10}\ s_{11}\ s_1\ s_{12} \Rightarrow t_4\ t_6\ t_7\ t_9\ t_8\ t_{17}\ t_{12}\ t_8 \tag{18.2}$$
$$(3) s_{13}\ s_{14}\ s_{15}\ s_6\ s_8\ s_{10} s_2 \Rightarrow t_{20}\ t_{11}\ t_2\ t_1\ t_8\ t_{14} t_6\ t_{18} \tag{18.3}$$

Let us consider training the maximum entropy model for the target word t_{14}, the training data will be modified as shown below. Note that RHS (Right Hand Side) is true where t_{14} is present in the RHS.

$$(S_1) s_1\ s_2\ s_3\ s_4\ s_5\ s_6\ s_7\ s_8 \Rightarrow\ \text{true}$$
$$(S_2) s_3\ s_7\ s_9\ s_{10}\ s_{11}\ s_1\ s_{12} \Rightarrow\ \text{false}$$
$$(S_3) s_{13}\ s_{14}\ s_{15}\ s_6\ s_8\ s_{10} s_2 \Rightarrow\ \text{true}$$

This data is used to train the probabilities $p(true|t_{14}, BOG(S))$ and $p(false|t_{14},\ BOG(S))$. The $BOG(S)$ are the n-gram features (like unigram, bigram and trigram) in the source sentence S, as shown in three equations. For the source sentence S_1 above, the features $(BOG(S_1))$ for uni-gram, bi-gram, and tri-grams are, respectively:

$$\text{(Unigram)} : s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, \tag{18.4}$$
$$\text{(Bigram)} : s_1\text{-}s_2, s_2\text{-}s_3, s_3\text{-}s_4, s_4\text{-}s_5, s_5\text{-}s_6, s_6\text{-}s_7, s_7\text{-}s_8, \tag{18.5}$$
$$\text{(Trigram)} : s_1\text{-}s_2\text{-}s_3, s_2\text{-}s_3\text{-}s_4\ s_3\text{-}s_4\text{-}s_5, s_4\text{-}s_5\text{-}s_6, s_5\text{-}s_6\text{-}s_7, s_6\text{-}s_7\text{-}s_8. \tag{18.6}$$

The maximum entropy model combines feature weights in a *log-linear* fashion as shown in Equation 18.7:

$$P(X|t, BOG(S)) = \frac{e^{W(t).F(X.BOG(S))}}{\mathbf{Z}}, \tag{18.7}$$

where, $W(t)$ represents the weight vector learned by training a *maxent* model for a target word t, and *F(X , BOG(S))* represents the feature vector that associates the input features with the class label (X).

To handle repetition of words in the target side, every word in the target sentences is associated with its repetition index. In the training data (18.1) above, the target word t_2 has occurred twice, the first occurrence represented as t_2^1 and the second occurrence as t_2^2. The t_2^1 models the occurrence of t_2 once in the target sentence given a source sentence, while t_2^2 models the occurrence of t_2 the second time in the target sentence given a source sentence. The training data for modeling repetition of t_2 is shown below. Here, t_2 has occurred twice in sentence (18.1).

$(S_1)\ s_1\ s_2\ s_3\ s_4\ s_5\ s_6\ s_7\ s_8 \Rightarrow\ true$
$(S_2)\ s_3\ s_7\ s_9\ s_{10}\ s_{11}\ s_1\ s_{12} \Rightarrow false$

(S_3) s_{13} s_{14} s_{15} s_6 s_8 $s_{10}s_2 \Rightarrow false$

By contrast, the training data for modeling the occurrence of t_2 will be, as t_2 appears in both 18.1 and 18.3.

(S_1) s_1 s_2 s_3 s_4 s_5 s_6 s_7 $s_8 \Rightarrow$ $true$
(S_2) s_3 s_7 s_9 s_{10} s_{11} s_1 $s_{12} \Rightarrow false$
(S_3) s_{13} s_{14} s_{15} s_6 s_8 $s_{10}s_2 \Rightarrow true$

For lexical choice, instead of producing the target sentence directly, what is initially obtained is BOWs, such that each word in the target bag is detected independently, so there is a simple use of binary classifiers. Given the sentence S, the bag of words $BOW(T)$ contains those words whose distribution have positive probability greater than a threshold (τ).

$$BOW(T) = \{t|P(true|t, BOG(S)) > \tau\}. \tag{18.8}$$

In order to obtain the target BOWs, the iteration is performed through target vocabulary, and the words that satisfy equation 18.8 are included in the predicted target language bag associated with the source sentence.

In the above discussion, we have not considered any lexical ordering. In order to incorporate word order, we could: (a) start with a random order of words, or (b) a word order is chosen based on their lexical choice probability. Note that the latter does not contain any sequence information. To obtain the correct ordering of the target sentence words using this word order, permutations of the target words is generated, and then for such permutations, an n-gram algorithm is applied to select the order that has the maximum probability. To implement the process, a finite automata is created, with states equal to $n+1$, where n is number of words in the target sentence. In generating various permutations, the words which are not needed in sequences, are substituted with ε-strings.

18.2 Pre- and Post-processing

Often, the MT models are trained on a *pre-processed* version of an original corpus. Pre-processing provides a simpler representation of the training corpus which makes token or word forms more homogeneous. This helps in boosting the automatic training of the MT model, and reduces the amount of computation.

The pre-processing steps are: *tokenization*, *filtering* (removing unnecessary case information), and tagging of some special tokens, like: numerical sequences, e-mail addresses, and URLs ("categorization"). In the translation from a source to a target language, some times, there are words which are translated identically, e.g., bus is same in English and modern German, hence they have the same spelling. Such words need to be identified in the corpus and are replaced with some generic tags to help the translation system.

Post-processing takes place after the translation, in order to hide the internal representation of the text from the user. Thus, the user will only work with an output which is very similar to human-generated texts. The post-processing steps are: *de-tokenization*, *true-casing*, and replacing the tags with their corresponding words.

In an *interactive-predictive machine translation* (IPMT) scenario, the pre-/post-processing must run in real-time (with respect to the translation) and should be reversible as much as possible. In each human – machine interaction, the current prefix needs to be pre-processed for the interactive-predictive *engine* and then the generated completion has to be post-processed for the *user*. It is important that the pre-processing of prefixes is fully compatible with the training corpus.

18.3 Finite-state Translation Models

Broadly speaking, a model is simply the set of all rules employed by an MT system to transform a source sentence into a target sentence. In principle these rules may come from anywhere. In most SMT systems they are automatically extracted from a parallel corpus, discussed in the previous sections. The most popular models can be described by one of two formalisms: *finite-state trans-ducers* (FST) or synchronous context-free grammars (SCFG). These formalisms are generalizations of finite-state automata (FSA) and context-free grammar (CFG), respectively. Rather than producing single output strings as in those formalisms, they produce two output strings, and define an alignment between them, in the case of FST. the translational equivalence models are important in decoding, where they constrain the search space in which we will attempt to find translations.

Finite-state transducers are extensions of finite-state automata (FSA). Recall that we can define a finite-state automaton (S, L, D), as a set of states S, a set of states S, set of labels L, labels Σ, and a set of transitions D. Each transition in $D \subseteq \{S \times S \times L\}$ is defined as a pair of states and a label that must be output (or read, depending on the use of the FSA) as we move from the first state to the second.

Finite-state transducers extend FSAs with a second label set. Each transition includes a label from both sets. We can imagine that the transducer operates on an input string and an output string. When a label from the first set is read from the input while traversing the transition, the label for the other is written to the output. A transition labeled with x from set L_1 and y from set L_2 therefore signifies a correspondence between x and y. Additionally, either or both label may consist of the empty string ε, which indicates that there is no change in that output for that particular transition.

We can compose FSTs by making the output of one FST the input to another, giving us a useful modeling tool. For each of the model types that we describe in this section, we will show how it can be implemented with composed FSTs. We first describe word-based models, which introduce many of the common problems in translation modeling.

18.3.1 Word-based Models

1. Each target word chooses the number of source words that it will generate. We call this number phi_i the *fertility* of e_i. One way of thinking about fertility is that when the transducer encounters the word e_i in the input, it outputs phi_i copies of e_i in the output. The length J of the source sentence is determined at this step since $J = \sum_{i=0}^{I} \phi_i$. This enables us to define a translational equivalence between source and target sequences of different lengths.

2. Each copy of each target word produces a single source word. This represents the translation of individual words.

3. The translated words are permuted into their final order.

These steps are also applied to a special empty token ε, called the *null word* – or more simply *null*, and denoted by e_0. Null translation accounts for target words that are dropped in translation, as is often the case with function words.

Note that the this alignment is asymmetric, i.e., each source word can align to exactly one target word or the null word. However, a target word can link to an arbitrary number of source words, as defined by its fertility. This is important in word alignment.

The reordering step exposes a key difficulty of finite-state transducers for translation. There is no efficient way to represent reordering in a standard finite-state transducer. They are designed to represent relationships between strings with a monotonic alignment – in other words, if an input label at position i is aligned to an output label at position j, then an input label at position $i' > i$ will be aligned to an output label at position $j' > j$. This is fine for problems such as automatic speech recognition and part-of-speech tagging, where monotonicity is the natural relationship between two sequences. However, as we have seen, words are typically reordered in real translation.

One solution to this discrepancy is to simply ignore the reordering problem and require monotonic alignments. This enables very fast decoding, but generally leads to less accurate translation, particularly for languages with naturally different word orders. At the other end of the spectrum is full permutation of the J source words. An FST to do this contains $O(J!)$ paths and $O(2^J)$ states. However, search for the best permutation is *NP-complete*, and the problem is reduced to the *Traveling Salesman Problem* (TSP).

Most models take a middle ground, using a mostly monotonic approach but allowing any of the k leftmost uncovered words to be translated. The setting $k = 4$ is some-times called the IBM constraint. This method enables local reordering, which accounts for much language variation. However, it still prevents long-distance reordering. Accounting for long-distance reordering without resorting to arbitrary permutation requires additional modeling.

18.3.2 Phrase-based Models

In real translation, it is common for contiguous sequences of words to translate as a unit. For instance, the sequence "front-row" is translated as "Aage ki pankti" (written in Hndi).

However, in word-based translation models, a substitution and reordering decision is made separately for each individual word. For instance, the model would have to translate "front" as "Aage ki" and "row" as "pankti" and then order them monotonically, with no intervening words.

Phrase-based translation addresses this problem. In phrase-based models the unit of translation may be any contiguous sequence of words, called a phrase. Null translation and fertility are gone. Each source phrase is nonempty and translates to exactly one nonempty target phrase. However, we do not require the phrases to have equal length, as the example above states, so model can still produce translations of different length. Now we can characterize the substitution of "front-row" as an atomic operation. If in the parallel corpus we have only ever seen "front" and "row" separately, we can still translate them using one-word phrases.

The translation process takes three steps:

1. The sentence is first split into phrases.
2. Each phrase is translated.
3. The translated phrases are permuted into their final order. The permutation problem and its solutions are identical to those in word-based translation.

Phrase-based models produce better translations than word-based models, and they are widely used. They successfully model many local reordering, and individual passages are often fluent. However, they cannot easily model long-distance reordering without invoking the expense of arbitrary permutation.

18.4 Synchronous context-free Grammar Model

Compared with the regular languages generated by FSAs, context-free grammars (CFG) confer a couple of potential benefits that relate to our modeling problem. First, they are closely tied to some linguistic representations of syntax. Second, in their synchronous form, they can easily represent long-distance reordering without the exponential complexity of permutation. However, added modeling power comes with added modeling challenges, and meeting these challenges is currently an area of much active research.

Indeed, there is some skepticism over the superiority of synchronous grammar models over finite-state methods. For the remainder of this section, we describe translation models based on *synchronous context-free grammars* (SCFG), and describe some of their modeling challenges.

SCFGs are known in different guises as syntax-directed translation, inversion transduction, head transducers, and a number of other names. A formalism that generalizes these is multitext grammar. In the following discussion, we will use the notation of SCFG, which is relatively easy to understand.

SCFG is a generalization of CFG to the case of two output strings. Recall that a CFG $=(N, T, D)$ consists of a set of nonterminal symbols N, terminal symbols T, and productions $D = \{N \rightarrow \{N \cup T\}^*$. Each production describes the in-place replacement of the productions right-hand side nonterminal with the left-hand side sequence of terminals and nonterminals. We begin by writing a special root nonterminal symbol to the output. This symbol is rewritten using a rule $d \in D$. Rewriting of nonterminal symbols continues recursively until the output contains only terminal symbols.

CFG is popular in natural language parsing, where terminal symbols are words (i.e., $T = VE$) and nonterminal symbols represe....

18.5 Language translation tools in Python

The NLTK is standard tool for natural language processing, which can be installed as basic Python library, which can be extended by installed more libraries. For language translation, the libraries are "translate" and "langdetect".

```
$ sudo pip install translate
```

Below is an example of translating a simple sentence from English to German:

```
from translate import Translator
translator= Translator(to_lang="German")
trans1 = translator.translate("Good Morning!")
trans2 = translator.translate("How are you?")
trans3 = translator.translate("What are you doing now?")
trans4 = translator.translate("I am reading")
print(trans1)
print(trans2)
print(trans3)
print(trans4)
```

When we run the above program, we get the following output:

```
Guten Morgen!
Wie geht es dir?
Was machst du gerade?
ich lese
```

To install *langdetect*, run the following command in your terminal.

```
$ pip install langdetect
```

Following is the code for detecting the language of sentence ”Guten Morgen!”:

```
from langdetect import detect
sent="Guten Morgen!"
print(detect(sent))
```

Which produces the output:

```
de
```

Here “de” stands for “Deutschland” (i.e., German).