TR－SLT－0054

Implementation of EM-IS Algorithm for Machine Translation

Vivien LE POUPON　　Taro Watanabe

December 18, 2003

We want to implement the EM-IS algorithm to build a lexicon model (and then try to use it in conjunction with an IBM model). We followed a Maximum Entropy (ME) approach, but expanded it to Latent ME (because normal ME is limited by scarcity of empirical data). The principle consists in embedding the iterative scaling loop in an EM procedure in order to determine the weighting paramaters associated with the different features: we are then able to make these parameters naturally match the information contained in the corpus. We developped this method using a corpus of English-Japanese pair.

（株）国際電気通信基礎技術研究所
音声言語コミュニケーション研究所
〒619－0288「けいはんな学研都市」光台二丁目2番地2 TEL：0774－95－1301

Advanced Telecommunication Research Institute International
Spoken Language Translation Research Laboratories
2-2-2 Hikaridai "Keihanna Science City" 619-0288,Japan
Telephone:+81-774-95-1301
Fax　　:+81-774-95-1308

1. The ME method

The ME method allows to build probability models. The idea underlying this principle is to make the model conform to some known facts, like statistics, whenever possible, and otherwise to let it be uniform.

For instance, say we want to translate the English word *please* into Japanese. Then, if we know that *please* can translate to either *kudasai*, *onegaishimasu*, or *douzo*, the following applies:

$$p(kudasai) + p(onegaishimasu) + p(douzo) = 1$$

If in addition we know that the translation will be *kudasai* 60% of the time, the ME principle provides us with the following distribution:

$$p(kudasai) = 0.6$$
$$p(onegaishimasu) = 0.2$$
$$p(douzo) = 0.2$$

Therefore, we are interested in extracting relevant statistics from our data, in order to define the constraints that will shape our model. This is done with the help of feature functions, or, more simply, features. They model an aspect of the context. Based on a *condition* representing an information about the sample, with x the context, and y the target word, a feature can be defined by

$$f(x,y) = \begin{cases} 1 & \text{if } condition \\ 0 & \text{otherwise} \end{cases}$$

For example:

$$f(x,y) = \begin{cases} 1 & \text{if } y = douzo \text{ and a comma precedes } please \\ 0 & \text{otherwise} \end{cases}$$

Features are the core of the model, so they have to be chosen with care.

How do we use them to compute the probabilities we need for our lexicon model?

To each feature corresponds a constraint (which is, as we explained previously, what we are looking for). Indeed, the expected values from the model and the data should be the same:

$$\sum_{x,y} \tilde{p}(x)p(y \mid x)f(x,y) = \sum_{x,y} \tilde{p}(x,y)f(x,y)$$

What we have to solve then, is a constrained optimization problem.
In the end, each feature f_i is associated with a weighting parameter λ_i. Then, we can express the probability for an event x as

$$p_\lambda(x) = \Phi_\lambda^{-1}(x) \cdot \exp\left(\sum_i \lambda_i f_i(x)\right)$$

where $\Phi_\lambda^{-1}(x)$ is a normalizing factor.

For a given set of features, we have to find the values of the different λ_i that maximize entropy. In order to do so, we use Improved Iterative Scaling (IIS), an algorithm designed to complete this task.

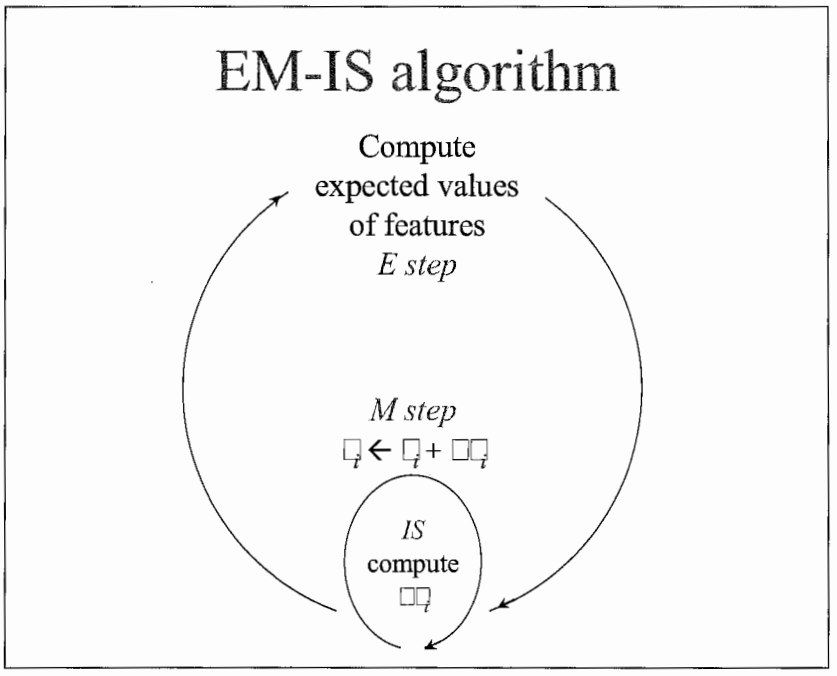
We have just presented what is called the Maximum Entropy (ME) method.
However, constraints used in ME models are subject to errors due to the scarceness of empirical data. Therefore, we will see how we want to try and find several possible "solutions" (i.e. sets of values λ_i) in order to refine the model.

2. The EM-IS algorithm

As we have just explained, a set of values λ_i obtained through IIS algorithm might not be accurate enough because empirical data would be very rough. Therefore, we are going to rely on the knowledge provided by our model to compute expectation, the constraint becoming

$$\sum_{x,y} \tilde{p}(x)p(y \mid x)f(x,y) = \sum_{x,y} p(x,y)f(x,y)$$

However, doing this, we have to "grope around" to make the model converge.
An idea is to restart the iterative procedure at different initial points, generating the several feasible "candidates" we are looking for. By aiming at lowering perplexity each time, we ascertain that the corresponding model should be conforming more and more closely to the training data.
To implement this, we use the Expectation-Maximisation (EM) algorithm in conjunction with the IIS algorithm.
The IS procedure is embedded in an EM loop, thus it is repeted over and over. And each EM loop allows to reevaluate the expected values of the features according to the training data.



3. Our development

FEATURES:

In order to define our features, we simply looked at which aspects of the context we wanted to highlight. We work with English-Japanese pairs of sentences. Indeed, our corpus has a list of couples: source sentence (English) – target sentence (Japanese), one being the translation of the other.
The context we want to model consists of the two words we suppose are translation of each other (E and J), plus the previous word in each sentence (E' and J').
Schematically:

English sentence:	$\dots E_{i-2}$	$E_{i-1} = E'$	$E_i = E$	$E_{i+1} \dots$
Japanese sentence:	$\dots J_{j-2}$	$J_{j-1} = J'$	$J_j = J$	$J_{j+1} \dots$
		context		

We began by using two sets of features:

- *f(J,E)*:

a given source word being translated into a given target word

- ➢ one feature for each possible pair of English-Japanese words
  - ✧ we call this set *trans*

- *f(J)*:

a given target word occuring

- ➢ one feature for each Japanese word
  - ✧ we call this set *targets*

Then, we introduced two more sets, in this order:

- *f(J,E')*:

a given source word preceding the word actually translated into a given target word

- ➢ one feature for each possible pair of English-Japanese words
  - ✧ we call this set *prev*

- *f(J',E)*:

a given source word being translated into a word following a given target word

- ➢ one feature for each possible pair of English-Japanese words
  - ✧ we call this set *zen*

PROGRAMMATION:

The program is written using C++ language. Here is a short explanation of its structure.

The file *train.cc* contains the general procedure. It creates a model, and then calls the main functions for the training of the model, following the EM-IS algorithm: *initialize*, *expectation*, *maximization*, ...

These key functions are found in the files *train_modelX.h*. X currently ranges from 1 to 3, and accounts for the version of the model. The main difference between each of these versions is the number of sets of features used: model 1 has only *trans* and *targets*, while *prev* was included in model 2, and model 3 boasts *zen* in addition to these three sets.

The code in the latter files comprises calls to functions implemented in the group of files *MEModelX.h*, one for each type of model again. These are the base computation functions, like *prob_of* (which returns the probability of a translation, given its context) or *sum_of_joint*. These files also contain the declarations for most of the structures required to build the model: the sets of features of course, but some cache tables and buffers used in our computations as well.

TRYING TO DECREASE RUNNING TIME:

After adding the last set of features, it became obvious that the program could not be used yet to train a model: to hope to do so, several weeks running on a powerful machine would be needed.

Consequently, some ways to reduce this time were required. The ones that we investigated are: draw aside some of the features to lower their number; in our computations, ignore values of limited significance; use alternate algorithmic methods to realize these computations.

We are going to present these ideas and their results.

Selection of features:

If less features are used, the time involved to build a model will eventually decrease, since most of the computations lie on iterating over the whole different sets of features. Therefore, we want to use only features that are "relevant", and erase those that do not hold enough importance in regard to our model.

How do we estimate this "relevance" of a given feature? The solution to which we have access consists of erasing features that have a low count according to the training data. We define a threshold for each set. After initialization, we delete each feature whose count is less than the threshold fixed for its set.

However, this proved to be a rough way of selection, as discarding any feature from the sets *trans* and *prev* results in a deficient model (a decent perplexity cannot be obtained). It is still possible to apply this selection to *targets* and *zen* without altering sensibly the quality or behavior of the model. For these sets, we affected values of 2 and 5 to the thresholds on the count. This caused to dismiss respectively more than 40% and 80% of the features originally collected from our data.

We cannot precisely estimate the gain in running time provided by this method, but, operating over a subset of our training data, it nearly divides this time by three, without inducing a loss in the model's efficiency.

Use of approximation:

Another idea was to limit our computations to significant values. Two different techniques were tested, both trying to ease the computation of *sum_of_joint*, which was very costly in its "original" form.

The first technique consisted in ignoring some groups of contexts (one being represented by two source words and two targets words), based upon the maximum values of *lambdas* for the corresponding groups of features. An approximate value would then be added.

The second technique involved using sortable containers to store the *lambdas*, in order to compute only the significant portion of each sub-sum.

The fact is that neither of these techniques allowed to greatly decreasing the running time. That's why we tried to use alternate methods of computation instead.

Alternate methods of computation:

Here again, we considered modifying *sum_of_joint*.

We can express the return value of this function by factorizing the sum of products in it:

$$\sum_{E,J,E',J'} e^{\lambda_J+\lambda_{E,J}+\lambda_{E',J}+\lambda_{E,J'}} = \sum_{E,J,E',J'} \left(e^{\lambda_J} e^{\lambda_{E,J}+\lambda_{E,J'}} e^{\lambda_{E',J}}\right)$$

$$= \sum_{J} \left[e^{\lambda_J} \left(\sum_{E,J'} e^{\lambda_{E,J}+\lambda_{E,J'}} \right) \left(\sum_{E'} e^{\lambda_{E',J}} \right) \right]$$

$$= \sum_{J} \left[e^{\lambda_J} \underbrace{\left(\sum_{E} \left(e^{\lambda_{E,J}} \underbrace{\sum_{J'} e^{\lambda_{E,J'}}}_{(2)} \right) \right)}_{(3)} \left(\underbrace{\sum_{E'} e^{\lambda_{E',J}}}_{(1)} \right) \right]$$

It is thus possible to compute the sub-sums in this expression ((1) over E', (2) over J', and then (3) over E) beforehand for each source word *J*. Then, the only computation remaining is a sum over all these possible *J*.
Such a trick makes the time required to compute *sum_of_joint* virtually negligible.

Nevertheless, this solves only half of the problem, as we still need to find a way to improve the computation of the maximization loops (the iterative scaling procedure). Our investigation of this matter has led to no positive result yet. The structure of the loops makes it difficult to apply a strategy similar to what we tried for *sum_of_joint*.

MEMORY ISSUES:

In addition, we are faced with another obstacle: after running for a long time, the program might crash because of a lack of memory.
It is probable that finding a way to decrease the running time to a reasonable figure would cause this issue to vanish. However, if such problems are still occuring, it might be useful to consider dropping some of the caches which have been set up.

4. Results

As the procedure for training takes too long a time when we use the model with four sets of features, we could not issue any result for it.

However, we managed to train a model using three sets of features (namely *trans*, *targets*, and *prev*). This model was trained using the file *train.e-j.snt*, while the test file was *valid.e-j.snt* (they were located in */home/pxs130/twatana/CJKE/corpus* when I used them). After three iterative scaling procedures, the model yielded the following perplexities:

over the training file:	44.766 (viterbi: 65.0918)
over the test file:	86.432 (viterbi: 125.166)

References

A. Berger, S. Della Pietra, V. Della Pietra; "A Maximum Entropy Approach to Natural Language Processing", 1996

P. Brown, S. Della Pietra, V. Della Pietra, J. Lai, R. Mercer; "An Estimate of an Upper Bound fro the Entropy of English", 1992

P. Brown, S. Della Pietra, V. Della Pietra, R. Mercer; "The Mathematics of Machine Translation: Parameter Estimation", 1993

R. Rosenfeld; "The EM Algorithm", 1997

S. Wang, D. Schuurmans, F. Peng; "Latent Maximum Entropy Approach for Semantic N-gram Language Modeling", 2002