# CS224N Homework 1

**Jiyue Wang**
jiyue@stanford.edu

**Ye Tian**
yetian1@stanford.edu

## 1 Word Alignment Algorithm and Implememtation

Our MT system is divided into language model and the translation model. The language model is to learn $p(e)$, which could be learned quite easily given the target language corpus. However, for the translation model, we need to learn $p(f|e)$, which is hard to estimate directly. In order to estimate $p(f|e)$, we introduce the alignment variable for each word in source sentence and try to estimate $p(f, a|e)$. In the assignment, we try to learn the distribution of the alignment variable from parallel corpus and give the best alignment for the test sentence pair using

$$a^* = \underset{a}{\operatorname{argmax}}\, p(a|f, e) \tag{1}$$

For PMI, IBM1 and IBM2 model, we assume that the alignment variable $a_1, ..a_m$ are independent.

### 1.1 PMI Model

PMI Model assumes

$$\begin{aligned} a_i^* &= \underset{j}{\operatorname{argmax}}\, \frac{p(f_i, e_j)}{p(f_i)p(e_j)} \\ &= \underset{j}{\operatorname{argmax}}\, p(f_i, e_j)/p(e_j) \end{aligned} \tag{2}$$

The maximum likelihood estimation for $p(f_i, e_j)/p(e_j)$ is $count(f_i, e_j)/count(e_j)$. Thus, in the training step, all we need to maintain is a source-targe word count and a source word count and save them for later use. In the aligning step, we find the alignment for $a_i$ by using the above equation. In PMI Model, we go through the corpus for only once.

### 1.2 IBM1 Model

IBM1 Model assumes that given the target sentence length $l$, the alignment variable $a_i$ has a uniform distribution $q(a_i|i, l, m) = \frac{1}{l+1}$.

$$\begin{aligned} p(f, a|e, l, m) &= \Pi_{i=1}^m q(a_i|i, l, m)t(f_i|e_{a_i}) \\ &= \Pi_{i=1}^m \frac{1}{l+1} t(f_i|e_{a_i}) \end{aligned} \tag{3}$$

Thus

$$\begin{aligned} a_i^* &= \underset{j}{\operatorname{argmax}}\, p(a_i = j|f, e) \\ &= \underset{j}{\operatorname{argmax}}\, p(f_i|e_j) \\ &= \underset{j}{\operatorname{argmax}}\, t(f_i|e_j) \end{aligned} \tag{4}$$

**During training.** We need two local variables for source-target count, and target count during each iteration. We have a static variable $t(f_i|e_j)$, which is initiallized uniformly.

**Stopping criteria.** We choose to set an iteration limit as the stopping criteria. We set the maximum iteration empirically. After some experiments, we found that five iterations over the whole corpus is sufficient to give a good result for all the three languages.

**Multithreading.** Considering that Corn has 4 cores, we decided to implemented the EM training part in a multithreaded fashion. The maximum threads are set to the number of cores as the code is computationally expensive. Each child thread will take one pair of training sentence and compute $p(a_i = j|f, e)$. Finally it will update the source-target count and target count.

**After training.** We get the estimated $t(f_i|e_j)$, which will be used in (4) during the alignment

### 1.3 IBM2 Model

In IBM2 Model, the alignment variables are still assumed to be dependent of each other, but no

longer uniformly distributed. The maximum likelihood estimation is given by

$$q(a_i = j|i, l, m) = \frac{count(j, i, l, m)}{count(i, l, m)} \quad (5)$$

Thus

$$a_i^* = \underset{j}{\operatorname{argmax}}\, q(a_i = j|i, l, m) * t(f_i|e_j) \quad (6)$$

**During training.** We need four local variables for source-target count, targe count, source positiong count and source-target position count. We have a static variable for $q(a_i = j|i, l, m)$ and $t(f_i|e_j)$.

**Initialization.** $t(f_i|e_j)$ is initiallized by the result of IBM Model1 while $q(a_i = j|i, l, m)$ is initiallized uniformly. We tested a uniform initiallization for both two variables and as expected, got a quite bad result. This is because, unlike IBM Model1, EM for IBM Model2 is not guaranteed to find the global optimum. Also, as IBM Model1 is a special case of IBM Model2, we could expect that the global optimal for IBM Model1 is close to IBM Model2. Thus, the result of IBM Model1 gives a better initiallization for IBM Model2.

**Stopping criteria.** We just set the same iteration number for IBM Model2 as IBM Model1.

**After training.** We get the estimated $q(a_i = j|i, l, m)$ and $t(f_i|e_j)$, which will be used in (6) during the alignment.

### 1.4 Result

| Dev | French | Hindi | Chinese |
|---|---|---|---|
| PMI | 0.6856 | 0.8452 | 0.8376 |
| IBMModel1 | 0.3975 | 0.6243 | 0.5862 |
| IBMModel2 | 0.3494 | 0.6127 | 0.5617 |

Table 1: Training Set

| Test | French | Hindi | Chinese |
|---|---|---|---|
| PMI | 0.6215 | 0.8515 | 0.8280 |
| IBMModel1 | 0.3761 | 0.6547 | 0.5539 |
| IBMModel2 | 0.3116 | 0.6405 | 0.5539 |

Table 2: Test Set

### 1.5 Error Analysis

As we speak Chinese, we did some bilingual analysis of the Chinese corpus. Though our IBM Model 2 is not significantly bettern than Model 1, we did find some interesting difference between the two model.

**Garbage collection.** The IBM models show garbage collection for words like *the, of, etc*.

**Rare target word alignment error.** The PMI model performs pretty bad on Chinese corpus. The PMI model tends to assign many source words to one rare target word, of which $p(targetWord)$ is too small. This situtation is alleviated in IBM model 1 and 2.

**No many-to-many alignment** In all the three models, one word cannot be aligned to multiple target words, which causes some alignment errors. Further more, it's hard to catch the structural features, i.e. the ordering of phrases.

**Comparison.** The AER of IBM model1 and model2 are quite similar; however, the alignments do differ. For IBM model1, the same source word, appearing multiple times, will be aligned to only one target word. Model1 simply ignore the position of source. However, there are more chances for Model2 to align the same source word to different target word. But unfortunately, Model2 might aligns some word to wrong word, which is correctly aligned by Model1.

## 2 MT System Training and Feature Engineering

In our concept, we classify statistical machine translation features into two classes, and we name them as language neutral features and language concerning features. One the one hand, by language neutral, we mean that those features are not attributes only belong to language, but also other sequences such as bit sequence, audio sequence and so on. For example, the length, position of a certain subsequence (phrase in language) or the relative position between a subsequence in the source and its corresponding subsequence in the target are all language neutral features. On the other hand, by language concerning features, we mean those features unique to (specific) languages. Now we talk on these two features separately. We should mention that all the features talked about below are rule features.

### 2.1 Language neutral features

In this section, we mainly utilize some features concerning the length of the phrases.

#### 2.1.1 Target dimension

By target dimension(TD), we mean the number of words in the target phrase. As noticed in the assignment, we presume the target dimension should either be too large or too small. So we try a categorical feature as follows:

$$TD = \begin{cases} 1 & \text{if tsize} \in \text{range} \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

Where tsize is target phrase size. Here we try two ranges which fires the feature Tar. The first is $\{2, 3\}$, and the second is $\{2, 3, 4\}$. The performance is as the table below:

| Feature Name | Average BLEU |
|---|---|
| TD(2, 3) | 15.277 |
| TD(2, 3, 4) | 15.262 |

Table 3: TD performance

We can see the average performances of these two ranges are about the same. One the other hand, the baseline performance is 15.290, and thus this feature by itself does not make a lot of progress.

#### 2.1.2 Source Target Difference

We use Source Target Difference(STD) to represent the difference between the number of words of the source phrase and that of the target phrase. We assume it should not be too much larger than 0. Again, we use multiple kinds of definitions of STD as shown below.

$$\begin{cases} \text{STD} = |\text{tsize - ssize}| \\ \text{STDLog} = \log(1+|\text{tsize - ssize}|) \\ \text{STDSqr} = |\text{tsize - ssize}|^2 \end{cases} \quad (8)$$

The first one is just the absolute value of the difference, which is natural. But we think this difference should not influence the performance linearly, so we propose two scaled versions of it. The first one is a log-scaled version of the first one, and the second one is a square-scaled version. Performance is as the table below:
We notice that STDLog by itself significantly improve the average BLUE. The other two does not did well, especially STDSqr.

| Feature Name | Average BLEU |
|---|---|
| STD | 15.243 |
| STDLog | 15.369 |
| STDSqr | 15.183 |

Table 4: STD performance

### 2.2 Language concerning features

In this section, since its very hard to take account of grammar, we mainly consider the correspondence of some frequent words. Moreover, some phrase's correspondences are also very potential, but we haven't tried it here.

#### 2.2.1 The and La/Le/Les

This is probably the most common words correspondence in English and French. Firstly we define a function:

$$InTarget(string) = \begin{cases} true & \text{if string} \in \text{Target Phrase} \\ false & \text{otherwise} \end{cases} \quad (9)$$

InSource can be defined similarly. Then we can define the feature as below:

$$TL = \begin{cases} 1 & \text{if InTarget(”the”) ==} \\ & \text{InSource(”la” || ”le” || ”les”)} \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

With this feature only, average BLEU = 15.372. It is quite a large progress.

#### 2.2.2 A/An/One and Un/Une

This is probably the most common words correspondence in English and French. This feature's accurate definition is as below.

$$AU = \begin{cases} 1 & \text{if InTarget(”a” || ”an” || ”one”)} \\ & \text{== InSource(”un” || ”une”)} \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

With this feature only, the average performance is 15.30. It makes a little progress.

#### 2.2.3 Same punctuation

Punctuations are also a typical characteristic of languages, and apparently, same punctuations

should appear in corresponding phrase pairs. So we define the feature below.

$$SP = \begin{cases} 1 & \text{if InTarget(”,”) == InSource(”,”)} \\ & \&\& \text{ InTarget(”.”) == InSource(”.”)} \\ & \&\& \text{ InTarget(”?”) == InSource(”?”)} \\ & \&\& \text{ InTarget(”!”) == InSource(”!”)} \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

With this feature only, the average performance is 15.273.

### 2.3 Composite features

As we can see, single features usually cannot give very good results. In this section, we mainly utilize some features concerning the length of the phrases. We demonstrate the result in the following table.

| Composite Features | Average BLEU |
|---|---|
| TD+TL | 15.36 |
| TD+TL+AU | 15.33 |
| TL+AU | 15.39 |
| TL+AU+SP | 15.42 |
| STDLog+TL+AU+SP | 15.46 |
| TD+STDLog+TL+AU+SP | 15.38 |
| STDLog+TL+AU+SP+AD | 15.30 |

Table 5: STD performance

We can see even with composite features, the BLEU progress is not very significant. The best one is STDLog+TL+AU+SP. If only consider the best single hit, TD+TL ever achieved 15.806, Which is made by TD+TL. After a number of experiments, we found that those combinations which provide high single hits do not ensure a high average performance because the distributions of their performance are of different shape, as shown in Figure 1.

We can see, for TD+TL, TL+AU+SP and STDLog+TL+AU+SP+Abs, they all have a central peak, but some concentrate more while others spread more. But for STDLog+TL+AU+SP, which has the best average performance, it doesn't have a peak and it has a lot of high records. As a result, we adopt **STDLog+TL+AU+SP** as our

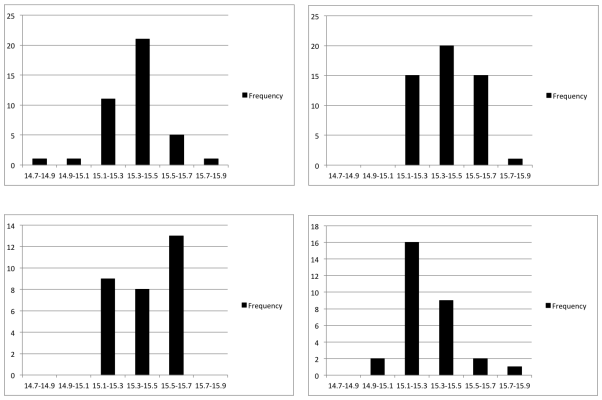


Figure 1: Performance distribution:
TD+TL, TL+AU+SP,
STDLog+TL+AU+SP,
STDLog+TL+AU+SP+Abs

feature combo. It's highest record is 15.644 (STDLog+TL+AU+SP17). We utilize it as our system.

| Feature Name | baseline | STDLog_TL_AU_SP17 |
|---|---|---|
| LM | .01880 | 0.2028 |
| LinearDistortion | 0.0707 | 0.0701 |
| PhrasePenalty | 0.0058 | 0.0171 |
| TM:FPT.0 | 0.1177 | 0.1224 |
| TM:FPT.1 | 0.0730 | 0.0928 |
| TM:FPT.2 | 0.1600 | 0.1335 |
| TM:FPT.3 | 0.1016 | 0.1064 |
| TM:FPT.4 | 0.0129 | 0.0184 |
| TM:FPT.5 | 0.1131 | 0.0950 |
| WordPenalty | -0.3061 | -0.2995 |
| STDLog | | -0.0209 |
| SP | | -0.0427 |
| TL | | -0.0513 |
| AU | | -0.0356 |

Table 6: Font guide.

We can see except for PhrasePenalty, all those old features have similar weight. This is because the added feature STDLog also penalize the phrase size. Now we list a examples of translations in the order baseline/Our System/Google Translate:
B: where the site is enriched uranium .
O: where is enriched uranium .
G: which is enriched uranium.
We can see the sequence of our system is similar to that of google translate, and it may be due to the article ”the”. There are a lot of similar examples. Moreover, the phrase pairs in our system have more similar length.

## 3 Derivation Feature (Extra Credit)

We implement some derivation features.These features are related to the context of the phrases, including their positions and the information of its prior phrase.

### 3.1 Absolute Distortion

In a lot of cases, we want the corresponding phrases appear at the same position of the source and the target. So we defined the feature absolute distortion(AD) as below:

$$AD = log(1+|source_position - target_position|) \tag{13}$$

Again, we curve it using log. It alone makes an average BLEU of

### 3.2 Two Phrase Difference

Similar to the rule feature :

$$AD = log(1+|source_position - target_position|) \tag{14}$$

However, this feature takes too long to compute.