Machine Translation
Alignment

Haitang Hu
hthu@cs.jhu.edu

March 2, 2015

The implementation is based on the work of Chris Dyer et al.[1]

1 Derivation

Instead of assuming all words are aligned equally, this model will favor on the diagonal word pairs. Also, the normalizing parameters can be computed in $O(1)$ time, which makes the computing process more efficient.

1.1 Marginals

Denote the input sentence pair as **(f,e)**, and correspondingly their length to be *(lf,le)*, and the alignment as a.
For every word e_i in $\mathbf{e}$[1], the joint probability of e_i aligned to a specific location a_i to be some specific translation f_{a_i} can be expressed as:

$$p(e_i, a_i \mid f, le, lf) = \delta(a_i \mid i, le, lf) t(e_i | f_{a_i})$$

We could simply marginalize out a_i, by summing all its possible choices:

$$p(e_i \mid f, le, lf) = \sum_{j=0}^{lf} p(e_i, a_i = j \mid f, le, lf)$$

The marginal is computed in the same way as IBM Model 2.

$$p(e \mid f) = \prod_{i=1}^{le} p(e_i \mid f, le, lf)$$

$$= \prod_{i=1}^{le} \sum_{j=0}^{lf} \delta(a_i \mid i, le, lf) t(e_i | f_{a_i})$$

Since we favored the diagonal pairs, the $\delta(a_i = j \mid i, le, lf)$ can be computed as follow:

$$h(i, j, le, lf) = -\left|\frac{i}{le} - \frac{j}{lf}\right|$$

[1]This **e** does not have to be English, but can be referred to be the source language. Same thing goes to **f**, which can be interpreted as target language.

$$\delta(a_i = j \mid i, le, lf) = \begin{cases} p_0 & \text{if } j = 0 \text{ (null word)} \\ (1-p_0)\frac{e^{\lambda h(i,j,le,lf)}}{Z_\lambda(i,le,lf)} & 0 < j \le lf \end{cases}$$

1.2 Normalization Trick

According to the paper, the normalization constant is allowed to be computed in constant $O(1)$ time. Since the unnormalized probabilities can be extended up and down from the diagonal, so we could compute through its closest point.

$$j_\uparrow = \lfloor \frac{i \times lf}{le} \rfloor, j_\downarrow = j_\uparrow + 1$$

While move one step, the probability will change at the rate:

$$r = e^{\frac{-\lambda}{lf}}$$

Here we introduce the geometric series: a series with a constant ratio between successive terms. In this problem, this ratio is r, and the value at states n could be expressed as:

$$s_j(a, r) = a + ar + ar^2 + \cdots + ar^{n-1} = a\frac{1-r^n}{1-r}$$

Where $a = e^{\lambda h(i,j,le,lf)}$ Then, the normalize term $Z_\lambda(i, le, lf)$ could be computed as

$$s_{j_\uparrow}(a_{j_\uparrow}, r) + s_{n-j_\downarrow}(a_{j_\downarrow}, r)$$

All computation mathematics is described above, where I didn't utilize automatic parameter optimization and gradient descent. Instead, other optimization efforts are described in later section.

2 Implementation

The pseudo code shows below:

Listing 1: fast_align.py

```
def main(args):
  bitext = pair(f_data, e_data)
  revtext = pair(e_data, f_data)
  #Compute alignment in 2 direction
  e2f = trainFastAlign(bitext, max_iter=5)
  f2e = trainFastAlign(revtext, max_iter=5)
  #Use intersection to output
  intersection(e2f, f2e)
  output()

def trainFastAlign(bitext):
  #Initialize translation probability using model 1
  tef = trainModel1(bitext)
  #Do 5 iterations
  for it in range(max_iter):
    cef = array((e_count, f_count))
```

```
likelihood = 0.0
#Enumerate lines
for (n, (e, f)) in enumerate(bitext):
  # Add null word
  en = [None] + e

  #Compute normalization
  prob_e(le)
  for (j, f_word) in enumerate(f):

    for (i, e_word) in enumerate(en):
      if i == 0:
        prob_a_i = p0
      else:
        prob_a_i = prob( j+1, i, lf, le -1, lamb) / Z

      prob_e[i] = tef[e_word][f_word] * prob_a_i
      sum_prob += prob_e[i]

    #Collect counts
    for (i, e_word) in enumerate(en):
      c = prob_e[i] / sum_prob
      cef[e_index[e_word]][f_index[f_word]] += c

#Estimate probabilities(Add counts)
tef += cef
#Normalize tef by row
normalize(tef)
```

3 Optimization

The optimization has be conducted on three aspects, initialization on word translation probability, parameters p_0, λ and alignment Refinement[2].

3.1 Initial Probability

Generally there are two ways to initialize word translation probability. We could either assign a uniform possibility, or use a pre-processing stage, where the translation probability is produced by IBM model 1 with several iterations. After testing the **AER**, a pre-processing method will reduce **AER** comparing to an arbitrary uniform initialization.

3.2 Parameter optimization

The parameters are evaluated on 1000 lines of sentences, and take the parameter that minimize the **AER**.

p_0	λ	AER
0.01	4.0	31.48
0.05	4.0	28.31
0.08	2.0	27.70
0.08	4.0	26.53
0.10	4.0	30.95

Table 1: Parameter optimization

3.3 Alignment Refinement

The input data will be trained twice, from source language to target language and vice versa. Then a simple intersection on the alignment is taken, which improves the **AER** by 3%. A grow diagonal is implemented, but seems buggy which increases the **AER**. So, by far the simple intersection is used.

References

[1] Chris Dyer, Victor Chahuneau, Noah A. Smith. *A Simple, Fast, and Effective Reparameterization of IBM Model 2*, 2013.

[2] Och, Franz Josef and Ney, Hermann. *A Systematic Comparison of Various Statistical Alignment Models*, 2003.

[3] Philipp Koehn. *Advanced Alignment Models*, 2015, `http://mt-class.org/jhu/slides/lecture-advanced-alignment-models.pdf`