

目录

1 课程总结	- 2 -
1.1 课程内容	
1.2 课程形式	
1.3 课程收获	
2 说明	- 3 -
3 实验一：实现 IBM Model 1	- 3 -
3.1 实验要求	
3.2 IBM Model 1 简介	
3.3 具体实现	
3.4 结果展示	
4 实验二：基于短语的模型	- 6 -
4.1 实验要求	
4.2 基于短语的模型简介	
4.3 具体实现	
4.4 结果展示	
附录 A: 实验 1 部分代码	- 8 -
附录 B: 实验 2 部分代码	- 10 -

1 课程总结

1.1 课程内容

课程主要讲解了以下内容：

- 1、 机器翻译的几个层次；
- 2、 基于单词的模型；
- 3、 基于短语的模型；
- 4、 解码过程。

1.2 课程形式

课程采取“阅读教材+老师提问”的形式。个人认为这种课程形式对于小班授课来说，真真是极好的。概括的来说，有两个优点：

- 1、 课程的节奏非常慢，能够使学生对知识的掌握非常细致。同时，老师可以很好的根据学生的掌握，调整上课的速度。
- 2、 这种课程形式其实极大地激发了学生的主动性，有助于提高学生的自学能力。

我个人是非常喜欢这种课程形式的。大部分的课程主要采取的是，单纯的“教师讲授”这种方法。我每每遇到这种课程，往往昏昏欲睡，没有上课的劲头；待到考试将近，才着急地开始“真正的”学习。

1.3 课程收获

学习本课程，我有以下的收获：

- 1、 **EM 算法**。很多算法虽学习过，但如不结合具体的例子，我还是懵懵懂懂、一脑袋浆糊。此番花了很多时间在 EM 算法上，我不仅对它的原理和实施过程有了进一步的了解，同时我相信以后我学习这类算法也会比以前更加轻松。
- 2、 **阅读英文书籍**。以前，阅读英文书籍给我的印象是，阴森恐怖、困难重重。如今读了这一遭，我发现英语书籍也不是那么的令人读不下去。语言障碍并没有想象的大。只是需要花些时间和耐心。
- 3、 **锻炼代码能力**。说实话，自从大一大二以后，代码写的非常少；而且非常多的一部分是在调 API。实验虽未完全实现，但是我的代码能力还是有所提高。
- 4、 **入门机器翻译**。虽不打算从事机器翻译这个方向，但是作为一个自然语言方向的学生，我觉得还是有必要对这个领域有所了解的。

2 说明

2.1 符号说明

为了以下内容的严谨性，我认为有必要对一些符号进行说明。

表 1 符号说明

符号	意义
\mathbf{e}, \mathbf{f}	句子
e, f	单词
\bar{e}, \bar{f}	短语
a, A	对齐
$t(e f), \phi(\bar{e} \bar{f})$	翻译概率

2.2 其他说明

详细代码见我的 github 仓库。¹

3 实验一：实现 IBM Model 1

3.1 实验要求

实现 IBM Model 1，并在给出的双语语料库 fbis.en.10k 和 fbis.zh.10k 上进行实验，估计单词的翻译概率。

3.2 IBM Model 1 简介

给定一系列双语语句对，IBM Model 1 将句子之间的对应分解为单词之间的对应。对于具体的句对，单词之间的对应通过对齐 a 进行表示。如对于双语句对 $\langle e_1 e_2 e_3, f_1 f_2 f_3 \rangle$ ，一个可行的对齐为

$$a : \{1 \rightarrow 1, 2 \rightarrow 2, 3 \rightarrow 3\}. \quad (1)$$

而对于所有双语句对，单词之间的对应则体现在英语单词 e 和外语单词 f 的翻译概率 $t(e | f)$ 上。

对齐 a 是针对具体的某个句子而言的，而单词间翻译概率则是全局的。如果已知二者中的一个，则可以对另一个进行估计。

¹ <https://github.com/1140310118/2018-course-Machine-Translation/>

3.2.1 由对齐到翻译概率

假设有大量的双语句对及它们的对齐 a_i ，我们可以使用计数的方法对 $t(e | f)$ 进行估计。如给定如下双语句对及对齐

表 2 双语句对及它们的对齐

双语句对 $\langle e, f \rangle$	对齐 a
$\langle a \text{ pen, 一根 钢笔} \rangle$	$a1 : \{1 \rightarrow 1, 2 \rightarrow 2\}$
$\langle a \text{ book, 一本 书} \rangle$	$a2 : \{1 \rightarrow 1, 2 \rightarrow 2\}$

那么可以得到翻译概率：

表 3 翻译概率 $t(e|f)$

e	f	$t(e f)$
a	一根	0.5
a	一本	0.5
pen	钢笔	1.0
book	书	1.0

3.2.2 由翻译概率到对齐

由翻译概率到对齐则复杂些。对于具体的某个句对，假设已知单词之间的翻译概率 $t(e | f)$ ，为了选择一个最佳的对齐，IBM Model 1 将对齐出现的概率定义为

$$p(a | \mathbf{e}, \mathbf{f}) = \prod_j \frac{t(e_j | f_{a(j)})}{\sum_i t(e_j | f_i)}. \quad (2)$$

公式(2)的推导过程在这里不再赘述。此公式表明，句子 \mathbf{e} 中单词 e_j 应该与使得翻译概率 $t(e_j | f_i)$ 最大的单词 f_i 对齐。

3.2.3 IBM Model 1 的训练

如果已知对齐 a 和翻译概率 $t(e | f)$ 二者中的一个，我们就能推得另一个。但不幸的是，二者都是未知的。

为了解决这个问题，IBM Model 1 采取了 EM 算法：

1. 使用均匀分布，对 $t(e | f)$ 进行初始化；
2. 使用 $t(e | f)$ 去估计对齐 a ；
3. 使用对齐 a 更新 $t(e | f)$ ；
4. 迭代步骤 2、3 直至收敛。

3.3 具体实现

3.3.1 算法伪代码

```
Input: set of sentence pairs (e, f)      14: // collect counts
Output: translation prob. t(e|f)         15: for all words e in e do
1: initialize t(e|f) uniformly           16: for all words f in f do
2: while not converged do                17: count(e|f) +=  $\frac{t(e|f)}{s - \text{total}(e)}$ 
3: // initialize                         18: total(f) +=  $\frac{t(e|f)}{s - \text{total}(e)}$ 
4: count(e|f) = 0 for all e, f           19: end for
5: total(f) = 0 for all f                 20: end for
6: for all sentence pairs (e, f) do      21: end for
7: // compute normalization              22: // estimate probabilities
8: for all words e in e do                23: for all foreign words f do
9: s-total(e) = 0                         24: for all English words e do
10: for all words f in f do               25: t(e|f) =  $\frac{\text{count}(e|f)}{\text{total}(f)}$ 
11: s-total(e) += t(e|f)                  26: end for
12: end for                               27: end for
13: end for                               28: end while
```

图 1 使用 EM 算法训练 IBM Model 1 的伪代码

3.3.3 实现细节

(1) 使用数字代替单词²

为了减少程序的内存占用，采取的策略是建立单词与数字之间的一一映射，训练时直接使用数字代替单词。

(2) 初始化

假设单词 e 可能的翻译选项为 $\{f_1, f_2, \dots, f_n\}$ ，那么 $t(e | f_i) = 1 / n$ 。

我的另一想法是，不进行平均初始化，而是根据 f_i 与 e 的共现频率以及 f_i 本身出现的频率对 $t(e | f_i)$ 进行初始化。即采取类似 TF_IDF 的策略进行初始化。这种初始化策略有可能加快后续的收敛。但实际代码中未采取此策略。

(3) 加速训练

每次迭代结束，淘汰掉那些翻译概率接近 0 的单词对 $\langle e, f \rangle$ 。实验发现，采取此策略，每次迭代的时间将逐渐减少。

(4) 终止条件

最初程序的终止条件是设定固定的迭代次数。但这是不科学的。

改进后的终止条件如下。由于每次迭代结束后会淘汰一部分单词对，因此单词对的总数将慢慢减少。这里设定的终止条件是：当单词对减少的比例小于一个阈值，则迭代终止。

² 此策略是从课程中的某个学长那学习到的。

3.4 结果展示

序号	e	f	t(e f)
1	introduction	引言	1.000000
2	other	其他	0.997382
3	advanced	先进	0.987018
4	1997	1997	0.979666
5	wuhan	武汉	0.977492
6	or	或	0.977167
7	1998	1998	0.972462
8	military	军事	0.962291
9	authorities	当局	0.952336
10	1999	1999	0.948960

图 2 实验一的部分结果

4 实验二：基于短语的模型

4.1 实验要求

实现基于短语的模型，输出的短语对及其概率。

4.2 基于短语的模型简介

IBM Model 属于基于单词的模型。与之相比，基于短语的模型有如下优点：

1. 由于单词往往存在着一对多的情况，因此单词不是最佳的翻译单元。
2. 基于短语进行翻译，有利于解决二义性的问题。
3. 如果语料充足，我们可以学习到非常长的短语，甚至可以记住整个句子的翻译。
4. 从概念上来看，基于短语的模型更简单。

训练基于短语的模型的前提，我们通过 IBM Model 1 或者其他模型得到了单词间的翻译概率。接下来，通过翻译概率获得双语句对间的对齐 A 。然后，通过对齐 A 发现对应的短语对。接着，简单地通过计数的方式计算短语对的翻译概率。

4.2.1 由对齐 a 到对齐 A

对齐实际上就是句对中单词间的映射。在 IBM Model 1 中，一个句对中的对齐 a 是一个单射，而在基于短语的模型中对齐 A 则不是。对齐 A 允许 $i \rightarrow j_1, i \rightarrow j_2$ 或 $i_1 \rightarrow j, i_2 \rightarrow j$ 同时在 A 中出现；但不允许 $i_1 \rightarrow j_1, i_1 \rightarrow j_2, j_2 \rightarrow j_1$ 同时在 A 中出现。下图是一个对齐 A 的一个例子。

我们通过调换双语句对的顺序，运行 IBM Model 1 两次，获得两

个对齐 a_{e2f} 和 a_{f2e} ，然后通过算法 GROW-DIAG 获得 A 。具体算法这里不进行赘述。

	michael	geht	davon	aus	,	dass	er	im	haus	bleibt
michael	■									
assumes		■	■	■						
that						■				
he							■			
will										■
stay										■
in								■		
the								■		
house									■	

图 3 对齐 A 举例

4.2.2 从对齐 A 中学习短语对

对于对齐 A 而言，如果短语对 $\langle \bar{e}, \bar{f} \rangle$ 满足如下条件，那么短语对 $\langle \bar{e}, \bar{f} \rangle$ 是一致的：

1. $\forall e_i \in \bar{e} : (e_i, f_j) \in A \Rightarrow f_j \in \bar{f}$;
2. $\forall f_j \in \bar{f} : (e_i, f_j) \in A \Rightarrow e_i \in \bar{e}$;
3. $\exists e_i \in \bar{e}, f_j \in \bar{f} : (e_i, f_j) \in A$ 。

4.2.3 计算短语间的翻译概率

使用列表 P 保存双语句对中所有一致的短语对， $\text{count}(\bar{e}, \bar{f})$ 则表示短语对 $\langle \bar{e}, \bar{f} \rangle$ 在列表 P 中出现的次数。于是，短语间的翻译概率为

$$\phi(\bar{f} | \bar{e}) = \frac{\text{count}(\bar{e}, \bar{f})}{\sum_{\bar{f}_i} \text{count}(\bar{e}, \bar{f}_i)} \quad (3)$$

4.3 具体实现

```

GROW-DIAG-FINAL(e2f, f2e):
    neighboring = {(-1,0), (0,-1), (1,0), (0,1),
                  (-1,-1), (-1,1), (1,-1), (1,1)}
    alignment = intersect(e2f, f2e);
    GROW-DIAG(); FINAL(e2f); FINAL(f2e);

GROW-DIAG():
    iterate until no new points added
    for english word e = 0 ... en
        for foreign word f = 0 ... fn
            if ( e aligned with f )
                for each neighboring point ( e-new, f-new ):
                    if ( ( e-new not aligned or f-new not aligned ) and
                        ( e-new, f-new ) in union( e2f, f2e ) )
                        add alignment point ( e-new, f-new )

FINAL(a):
    for english word e-new = 0 ... en
        for foreign word f-new = 0 ... fn
            if ( ( e-new not aligned or f-new not aligned ) and
                ( e-new, f-new ) in union( e2f, f2e ) )
                add alignment point ( e-new, f-new )

```

图 4 探索法对齐的伪代码

```

Input: word alignment A for sentence pair (e, f)
Output: set of phrase pairs BP
1: for e_start = 1 ... length(e) do
2:   for e_end = e_start ... length(e) do
3:     // find the minimally matching foreign phrase
4:     (f_start, f_end) = ( length(f), 0 )
5:     for all (e, f) ∈ A do
6:       if e_start ≤ e ≤ e_end then
7:         f_start = min( f, f_start )
8:         f_end = max( f, f_end )
9:       end if
10:    end for
11:    add extract(f_start, f_end, e_start, e_end) to set BP
12:  end for
13: end for
function extract(f_start, f_end, e_start, e_end)
1: return {} if f_end == 0 // check if at least one alignment point
2: // check if alignment points violate consistency
3: for all (e, f) ∈ A do
4:   return {} if e < e_start or e > e_end
5: end for
6: // add phrase pairs (incl. additional unaligned f)
7: E = {}
8: f_s = f_start
9: repeat
10:  f_e = f_end
11:  repeat
12:    add phrase pair (e_start .. e_end, f_s .. f_e) to set E
13:    f_e++
14:  until f_e aligned
15:  f_s--
16: until f_s aligned
17: return E

```

图 5 一致短语对抽取的伪代码

4.4 结果展示

可以看出，实现了一致短语对的抽取，但翻译概率的估计仍不准确。可能的原因有 3 个：(a) 数据量太少；(b) 翻译概率的计算方法不恰当；(c) 代码中存在错误。

表 4 实验 2 的部分结果

序号	\bar{e}	\bar{f}	$\phi(\bar{f} \bar{e})$
1	zhongxing 189 dual-band mobile telephone characterized by	自主 知识 产权 的 中兴 一八九双频 手机	0.1
2	zhongxing 189 dual-band mobile telephone characterized by	自主 知识 产权 的 中兴 一八九双频	0.1
3	zhongxing 189 dual-band mobile telephone characterized by	知识 产权 的 中兴 一八 九双频 手机 仔细	0.1
4	zhongxing 189 dual-band mobile telephone characterized by	知识 产权 的 中兴 一八 九双频 手机	0.1

附录 A: 实验 1 部分代码

1、单词转数字 word2num(sentences)


```

1 def word2num(sentences):
2     sentences_new = []
3     n2w = {}
4     w2n = {}
5     n = 0
6     for sentence in sentences:
7         sent_new = []
8         for w in sentence.split():
9             if w in _w2n:
10                sent_new.append(_w2n[w])
11            else:
12                sent_new.append(n)
13                _w2n[w] = n
14                n2w[n] = w
15                n += 1
16        sentences_new.append(sent_new)
17    return sentences_new, n2w

```

2、初始化 init_t(E, F)

```

1 def init_t(E, F):
2     option = defaultdict(set)
3     t = defaultdict(int)
4     for p in range(len(E)):
5         for e in E:
6             for f in F:
7                 t[(e, f)] += 1
8                 option[f].add(e)
9
10    for f in option:
11        option_of_f = len(option[f])
12        for e in option[f]:
13            t[(e, f)] = 1/option_of_f
14    return t

```

3、一次迭代 one_loop_for_train(E, F, t)

```

1 def one_loop_for_train(E, F, t, zero=0.001):
2     count = defaultdict(int)
3     total = defaultdict(int)
4     s_total = defaultdict(int)
5     for p in range(len(E)):
6         s_total.clear()
7         for e in E[p]:
8             for f in F[p]:
9                 if (e, f) in t:
10                    s_total[e] += t[(e, f)]
11
12        for e in E[p]:
13            for f in F[p]:
14                if (e, f) in t:
15                    count[(e, f)] += t[(e, f)]/s_total[e]
16                    total[f] += t[(e, f)]/s_total[e]
17
18    non_zero_num = 0
19    total_num = 0
20    for e, f in list(t):
21        t_ef = count[(e, f)]/total[f]
22        if t_ef > zero:
23            t[(e, f)] = t_ef
24            non_zero_num += 1
25        else:
26            t.pop((e, f))
27        total_num += 1
28    return non_zero_num, total_num

```

4、主循环 mainloop (E, F, t)

```
1 def mainloop(E,F,t,threshold=0.05):
2     print ("第1次迭代")
3     _,last_total_num = one_loop_for_train(E,F,t)
4     i = 1
5     while True:
6         i += 1
7         print ("第%d次迭代: t 中减少了"%i,end=' ')
8         _,total_num = one_loop_for_train(E,F,t)
9         _rate = (last_total_num-total_num)/last_total_num
10        if _rate<threshold:
11            break
12        last_total_num = total_num
```

附录 B: 实验 2 部分代码

1、探索法对齐 GROW_DIAG (e2f, f2e, ee_len, ff_len)

```
1 def GROW_DIAG(e2f, f2e, ee_len, ff_len):
2     alignment = e2f & f2e
3     union_ali = e2f | f2e
4     for i in range(ee_len):
5         for j in range(ff_len):
6             for ni,nj in ((-1,0),(0,-1),(1,0),(0,1),\
7                           (-1,-1),(-1,1),(1,-1),(1,1)):
8                 inew = max(min(0,i+ni),ee_len-1)
9                 jnew = max(min(0,j+nj),ff_len-1)
10                if (inew,jnew) in union_ali:
11                    if noaligned(inew,alignment,0)\
12                        or noaligned(jnew,alignment,1):
13                        alignment.add((inew,jnew))
14     return alignment
```

2、一致短语对抽取 extract (f_start, f_end, e_start, e_end, A, ff_len)

```
1 def extract(f_start,f_end,e_start,e_end,A,ff_len):
2     if f_end == -1:
3         return set()
4     for e,f in A:
5         if f_start<=f<=f_end:
6             if not e_start<=e<=e_end:
7                 return set()
8     E = set()
9     f_s = f_start
10    while f_s>=0:
11        f_e = f_end
12        while f_e<ff_len:
13            if 0<f_e-f_s<=6 and 0<e_end-e_start<=6:
14                E.add((e_start,e_end,f_s,f_e))
15            f_e += 1
16        f_s -= 1
17    return E
```

3、计算短语间的翻译概率 cal_translation_probability_for_parse (phrases_dic)

```
1 def cal_translation_probability_for_parse(phrases_dic):
2     count = defaultdict(lambda : defaultdict(int))
3     t_pha = {}
4     for pe,pf in phrases_dic:
5         count[pf][pe] += 1
```

```

6         for pf in count:
7             sum_ = len(count[pf])
8             if sum_ < 10:
9                 continue
10            for pe in count[pf]:
11                p = count[pf][pe]/sum_
12                t_pha[(pe,pf)] = p
13        return t_pha

```

4、主循环

```

1  zh=txt_to_lst('data/fbis.zh.10k')
2  en=txt_to_lst('data/fbis.en.10k')
3  t1 = read_translation_probability_for_word('data/parse_\
      base_model_p_result1.txt')
4  t2 = read_translation_probability_for_word('data/parse_\
      base_model_p_result1.txt')
5  pharses_dic = defaultdict(int)
6  for i in range(len(zh)):
7      ee = en[i].lower().split()
8      ff = zh[i].lower().split()
9      ee_len,ff_len = len(ee),len(ff)
10
11     e2f = get_aligment_for_sentence(t1,ee,ff,ee_len,\
      ff_len)
12     f2e = get_aligment_for_sentence(t2,ff,ee,ff_len,\
      ee_len)
13     A = GROW_DIAG(e2f,f2e,ee_len,ff_len)
14
15     for e_start in range(ee_len):
16         for e_end in range(e_start,ee_len):
17             f_start,f_end = ff_len-1,-1
18             for e,f in A:
19                 if e_start<=e<=e_end:
20                     f_start = min(f,f_start)
21                     f_end = max(f,f_end )
22             E = extract(f_start,f_end,e_start,e_end,A,\
23                 ff_len)
24             read_parse(E,ee,ff,pharses_dic)
25
26  t_pha=cal_translation_probability_for_parse(pharses_dic)

```