Transliteration Project

REPORT_lhe_0725

Introduction

Papers

Courses

Data Log

Problems Discussion

Experiments

Pipeline (Appendix I)

Results (Appendix II)

Analysis (en2ch)

Introduction

The project can be divided into four parts, i.e. reading papers, learning deeplearning.ai courses, collecting data, and building/training some models of the English-to-Chinese transliteration.

Papers

Date: 19/06/2018

I mainly focused on the sequence-to-sequence paper from Google, but I could not understand some concepts such as the bidirectional encoder. Also, this paper is rather short and hides many details. What I am currently interested is its discussion and description of the transliteration dataset (which will be discussed below), as it reveals some problems that I am also concerned while gathering my first dataset. Since I was trying to run the Google's model and adjusting my data, I have not finished the reading of the paper concerning the Monolingual Corpora, I will study it after the meeting on Wednesday.

Courses

Date: 19/06/2018

The Coursera website provides me with great materials to learn Machine Learning. I currently finished the first simple project of the binary classification using logistic regression, it is implemented on Jupyter/iPython Notebook. Since two month is a short period of time, I will accelerate the learning process, hopefully will know LSTM/GNN before building the final model.

Date: 10/07/2018

Heading to the hyperparameters section.

Data Log

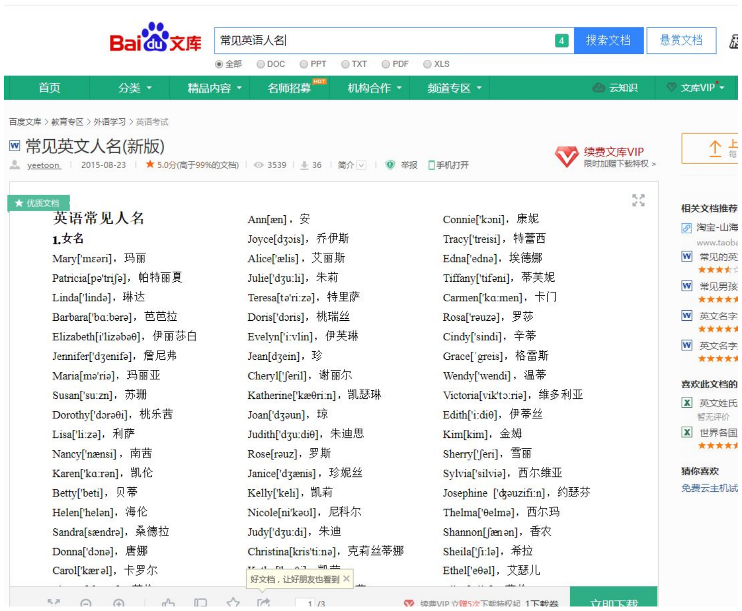
Date: 19/06/2018

Description:

The original en2chi dataset I have created has about 7000 word pairs, including people's names from different origins (particularly there is a small list of the ancient Greek Gods such as Zeus, and a small list of NBA players), places' names from various countries (cities in America/Britain/New Zealand take up the majority), and a list of brands and borrowed words. After some refinement of the data, the real number of word pairs for training (start date: 19/06/2018) is 6828.

Data Sources:

In practice, it is easier to use a Chinese search engine to search for some lists of English-Chinese word pairs. The engine I used for collecting the majority of the data is Baidu, where there is an embedded library called '百度文库' (Baidu's Library). An example is shown in the picture below:



常见英文人名(新版)

英语常见人名

1.女名

Mary['mɛəri]，玛丽
Patricia[pə'triʃə]，帕特丽夏
Linda['lində]，琳达
Barbara['bɑ:bərə]，芭芭拉
Elizabeth[i'lizəbəθ]，伊丽莎白
Jennifer['dʒenifə]，詹尼弗
Maria[mə'riə]，玛丽亚
Susan['su:zn]，苏珊
Dorothy['dɔrəθi]，桃乐茜
Lisa['li:zə]，利萨
Nancy['nænsi]，南茜
Karen['kɑ:rən]，凯伦
Betty['beti]，贝蒂
Helen['helən]，海伦
Sandra[sændrə]，桑德拉
Donna['dɔnə]，唐娜
Carol['kærəl]，卡罗尔
Ann[æn]，安
Joyce[dʒɔis]，乔伊斯
Alice['ælis]，艾丽斯
Julie['dʒu:li]，朱莉
Teresa[tə'ri:zə]，特里萨
Doris['dɔris]，桃瑞丝
Evelyn['i:vlin]，伊芙琳
Jean[dʒein]，珍
Cheryl['ʃeril]，谢丽尔
Katherine['kæθri:n]，凯瑟琳
Joan['dʒəun]，琼
Judith['dʒu:diθ]，朱迪思
Rose[rəuz]，罗斯
Janice['dʒænis]，珍妮丝
Kelly['keli]，凯莉
Nicole[ni'kəʊl]，尼科尔
Judy['dʒu:di]，朱迪
Christina[kris'ti:nə]，克莉丝蒂娜
Connie['kɔni]，康妮
Tracy['treisi]，特蕾西
Edna['ednə]，埃德娜
Tiffany['tifəni]，蒂芙妮
Carmen['kɑ:men]，卡门
Rosa['rəuzə]，罗莎
Cindy['sindi]，辛蒂
Grace[ˈgreis]，格雷斯
Wendy['wendi]，温蒂
Victoria[vik'tɔ:riə]，维多利亚
Edith['i:diθ]，伊蒂丝
Kim[kim]，金姆
Sherry['ʃeri]，雪丽
Sylvia['silviə]，西尔维亚
Josephine ['ʤəuzifi:n]，约瑟芬
Thelma['θelmə]，西尔玛
Shannon[ʃænən]，香农
Sheila['ʃi:lə]，希拉
Ethel['eθəl]，艾瑟儿

In this example, I typed '常见英文人名' (common English names of people) in the textbox beside the logo of Baidu's Library, it actually gave a list of choices. Then I selected the documents whose words are reasonably formatted by the author, and wrote some code (in Python) to extract the relevant information. There are some words coming from other websites, but all searched using Baidu.

Date: 20/06/2018

Description: Collecting a set of names of all Western celebrities' in Chinese.

Date: 23/06/2018

Description:

After finding two giant dictionaries of worldwide names of people and places, with Chinese transliteration aside, I was very excited because this data source is excellent (For one, the **transliteration style is unified** because of the single author; Also, the number of data is guaranteed to be large enough). I wrote some code in Python to extract the relevant information from the dictionary. After the meeting on 20th June, my supervisors and I agreed on **using people's names first** because they are more regular than places' names. I extracted all the people's names from the categories (the author has labelled each word with its related country(ies)) containing the key character '英' or '美', which are abbreviations of **the UK and the US** in Chinese respectively. Overall, there are **57948 distinct word pairs,** I divided them in a ratio of 8:1:1, the 80% part is used for training purposes, the rest are used for development and testing respectively.

Date: 24/06/2018 (Important)

Description:

The next step of data processing is to use some clustering algorithm (e.g. K-means) on the dataset I built. One of my supervisors, Professor Shay Cohen kindly taught me the idea of how to process the data, especially the construction of feature vectors. We decided to consider the left unigram and right unigram as features of each character of a English/Chinese word, and add 'IPA feature' for Chinese characters only. So a feature vector would look like:

[left_unigram ; right_unigram ; left_IPA; right_IPA; character_itself; original_word; class_to_be_determined]

For English character, the dimension would be <u>(26+2)+(26+2)+1+1+1</u> (2 for ^ $).

For Chinese character, the dim would be <u>(435+17+2)+(435+17+2)+(Table_size)^2+1+1+1.</u>

Currently in the **training dataset** there are <u>435</u> distinct Chinese characters (and 17 characters in the original IPA table but not shown in the training dataset), it could be more in the whole dataset. So the dimension shown above is for **inductive setting:** cluster training dataset only. The **transductive learning** will be discussed later.

Today I spent the whole day to construct a 'reasonable' IPA table of which the columns are consonants and the rows are vowels, each cell is filled with zero or more Chinese characters or a mark 'Bigram'. (The 'Bigram' here is used for indicating the cells where some special pairs of Chinese bigram represent a composite sound, since we only consider sound for single character here, these special pairs are removed and marked as 'Bigram') In total, out of 435 characters, there are 150+ character missing in the original table (discussion about these characters will be presented in the Problems Discussion section of this report). I added all of them into the original IPA table from Wikipedia, the rules of insertions I used are as follows:

1. **The top priority is not the original sound of the Chinese character, but its factual usage in English transliteration.**
2. Bear 1. in mind, I did:
    a. Search each of the 150+ Chinese characters in the training dataset first and look at all the English sequences of characters it corresponds to.
    b. Search the pronunciation of the English sequences using Google, if there is an exact IPA presentation, insert the character into the table where **objectively** appropriate, otherwise analyse the sound of the original English word the sequence is attached to, and insert the Chinese character where **subjectively** appropriate.
    c. If the sound information of some particular English words is ambiguous and the Chinese transliteration looks weird and unmatched, I will insert the Chinese character according to its original sound.

The method I used for constructing the IPA table may not be the best, but even my linguistician friend, who knows both English and Chinese very well, cannot resolve the potential risks here. The final IPA table is uploaded to GitHub, compared to the original one, I added two more columns for the composite sound ‘tr’ and ‘di’ in order to deal with some special characters.

Date: 02/07/2018

Below are the recent achievements:

1. Create a csv file storing all the feature vectors generated from the English names dataset, each in a format of [left_unigram ; right_unigram].
2. Create a csv file storing all the feature vectors generated from the Chinese names dataset, each in a format of [left_unigram ; right_unigram ; left_IPA ; right_IPA].
3. It turns out that the training dataset does not include all the Chinese characters used in the whole dataset, But since I am using an inductive setting, the unknown information from the development and test dataset is not added to the unigram features. Only the IPA table is updated accordingly. <u>(Whether this interpretation is right or wrong needs to be confirmed with my supervisors) Answer: do not remove.</u>
4. The smoothing process is applied, so that the features with a frequency no more than three are deleted before the clustering process. Hopefully, this will minimize the negative effect brought by rare characters and special usage.
5. Create a virtual environment to prepare for the final training process.
6. All the data tagged by ‘eva’ are actually belong to the development dataset.
7. The annotated datasets using 2, 5, 10, and 15 clusters are ready for training. The clustering algorithm used is K-means.

Date: 03/07/2018

Description:

After figuring out how to use the openNMT model to train the data (I gave up on Nematus because of the lack of instructions), I tried to use it to train the annotated dataset with 5 clusters first, the training process is ongoing now. However, the tensorflow version of openNMT does not give enough and clear instructions for me to choose the right model type, so I chose the NMT_medium instinctively (the toy examples use the NMT_small), this part along with the specific configurations should be discussed on Wednesday’s meeting.

Date: 17/07/2018

Description:

After some efforts and struggles, everything is prepared for running multiple settings of experiments.

----DATA LOG END----

Relevant URLs:

Baidu: www.baidu.com

Baidu's Library: https://wenku.baidu.com

Transliterated Names of celebrities: http://www.manmankan.com/dy2013/mingxing/oumei/

Full dictionaries of names (people/places):

http://resource.ttplus.cn/publish/app/data/2016/04/04/6754/share1.html

The GitHub address of my project:

https://github.com/Lawhy/En2Chi-Transliteration

OpenNMT: https://github.com/OpenNMT/OpenNMT-tf

OpenNMT_instructions: http://opennmt.net/OpenNMT-tf/

Problems Discussion

Date: 19/06/2018

As mentioned in the Seq2Seq paper, there are many exceptional words whose transliteration probably cannot be learnt at all. Although currently I can erased these exceptions by checking the data line by line, such noisiness cannot be resolved manually when the dataset becomes larger and larger. In my observation, the exceptions occur a lot in the names of places, brand or simply by convention. The reasons behind may result from historical factors or literary manipulation of words. For example, Paris is transliterated to 巴黎, but it actually sounds more like 帕里斯. Nevertheless, 巴黎 has a sense of more 'beauty" than 帕里斯 in Chinese. <u>My idea is that should we divide the dataset into names of peoples and names of others, because exceptions are much less frequent in people's names.</u>

The second issue is about the general noisiness among the places' names. There are dozens of names containing certain parts irrelevant to transliteration, such as 'River', 'Port', 'Mount', 'Sea', 'Island', 'North/East/West/South' , etc., which correspond to '河', '港', '山', '海', '岛, '北/东/西/南' in Chinese. Even though I can erase the irrelevant parts by seeking for some patterns, it will result in a 'collateral damage' in the sense that some words containing these seemingly irrelevant parts as sound components will lose them accidentally. For instance, the English name 'Westbrook' contains 'West', but in this case 'West' does not represent a direction. Instead, 'West' will be transliterated to '韦斯特'. Again, for a small dataset, I can resolve all the conflicts by checking it line by line. But it is impossible for a giant dataset.

The final concern is that Chinese characters are pictograms. There is no way to infer the sound by simply looking at the character itself. Also, Chinese is a many-to-many language in the sense that a single Chinese character may have several sounds and a particular sound may be possessed by several characters, whereas the sound of a single English word is almost fixed (although there exists different accents). Consequently, a model simply dealing with the textual data may treat alternative and other valid transcriptions as errors. (Mihaela Rosca et al., 2016)

Date: 25/06/2018

About the IPA table:

The original table was taken from Wikipedia, which uses a classical set of Chinese characters to represent the IPA sound made up by a consonant component and a vowel component. Most of the cells contain only one Chinese character, some contain a substitute for the alternative gender, a few of them contain a bigram because for some IPA combination, there is no way to use one character to represent that sound.

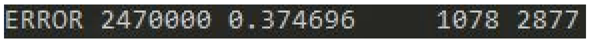
I believed that the original table is trying to give a one-to-one correspondence for the IPA sounds and the Chinese characters, and it actually covers a large portion of characters used in transliteration. In order to make the IPA feature more concrete, I added the missing 150+ Chinese characters into the IPA table in a way described in the Data Collection & Manipulation Log part. Among these 150+ characters, some actually appear multiple times in the dataset and their IPA positions are relatively easier to locate. However, the are some characters used in a special/weird way, the cases are:

1. Characters chosen because of convention, e.g. Athena (雅典娜), the pronunciation of Athena is [əˈθiːnə], whereas 雅典娜 sounds like ‘yudiana’. In this case, I did not put the character 雅 into the cell of its closest sound ([jʌ]), instead, I put it into the cell of [ə]. The same principle is also applied on ‘典“, the second character, whereas the third one ‘娜’ can be located easily because it sounds very similar to [nə]. This is a concrete example of what I mean by **‘Factual usage prior to the original sound’** in the Data Log section.
2. Characters used only one or two times, e.g. ‘典’ in ’雅典娜‘ (Athena), ‘耀’ in ‘Jauchem’. I did not delete them because they are minority.
3. Characters used in some **marginal names** which are used not frequently in English (so that I cannot find a proper IPA from Wikipedia). In this case, I first tried to listen to several sounds of that name from some website offering pronunciation. Moreover, I also took account of the original sound of the Chinese character.

After finishing the table, I kindly asked a linguistician friend to check its validity. He pointed out that the IPA table actually includes some pronunciation not commonly used in English, I believed that the author of the original table also tried to include some exceptional names.

Experiments

Date: 19/06/2018

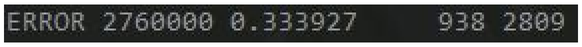


```
ERROR 2470000 0.374696     1078 2877
```

In the first round, I used 7011 word pairs to train the seq2seq model and obtained an error rate of 37.4696% for the training. However, the model behaves rather badly on the test data.

I am now re-training the model using a refined dataset (By refined I mean that I erase some exceptions manually) of the size 6800+. Hopefully it will be finished by tomorrow's meeting.

Date: 21/06/2018



```
ERROR 2760000 0.333927     938 2809
```

In the second round, I removed some fatal errors from the dataset, and the error rate decreases to 33.3927% at the 2,760,000th trail. However, I just found an excellent data source which includes about 820 thousand word pairs (although not all of them are English-to-Chinese). The baseline experiment for the current dataset ends at this point.

Date: 08/07/2018

Using openNMT to train --, -+, +-, ++. First round on tensorflow. Encounter low-level bugs.

(Corresponding log/results not shown here)

Date: 10/07/2018

Using openNMT to train --, -+, +-, ++. Second round on pyTorch with GPU support.

Here is the experiment setup:

https://github.com/Lawhy/En2Chi-Transliteration/blob/master/README.md

Date: 20/07/2018

Finish and record a full **en2ch** experiment with setting 1.

Details in here:
https://docs.google.com/document/d/1vIZGSELcZVQyAs_9dorSxc0_gMqxVu3CCPF3RlYT1lg/edit?usp=sharing

Date: 22/07/2018 (Important)

Add more feature vector choices, prepare for next setting of experiments by changing the features. Available features are:

```
{
"L": [L_unigram],
"R": [R_unigram],
"LR": [L_unigram ; R_unigram],
"bLR": [LR_bigram],
"LRbLR": [L_unigram ; R_unigram ; LR_bigram]
# IPA only for Chinese data
}
```

Details are in the GitHub file: feature_vector.py.

A problem: Do not know how to use spectral clustering algorithm in this particular project.

A reminder: exp1 results were not totally collected. Need to discuss new ideas of processing Chinese data with Shay on Monday.

Date: 24/07/2018

Run experiments using the same feature vector setting ("LR": [L_unigram ; R_unigram]) but with different clusters (4, 7, 9, 12).

Pipeline (Appendix I)

Results (Appendix II)

Setting_1: { K-means

[left_unigram ; right_unigram ; IPA (for Chinese only)] }

En2ch				
Ann/Cls	**2**	**5**	**10**	**15**
++	**WER: 0.29686**	**WER: 0.29496**	**WER: 0.31688**	**WER: 0.32568**
	CER: 0.12105	**CER: 0.12105**	**CER: 0.13031**	**CER: 0.13279**
+-	**WER: 0.30186**	**WER: 0.27874**	**WER: 0.29565**	**WER: 0.28564**
	CER: 0.12466	**CER: 0.11243**	**CER: 0.12175**	**CER: 0.11598**
-+	**WER: 0.29134**	**WER: 0.28875**	**WER: 0.31498**	**WER: 0.32361**
	CER: 0.11819	**CER: 0.11641**	**CER: 0.13107**	**CER: 0.13284**
-- (Baseline)	**WER: 0.29841**			
	CER: 0.12239			

En2jp				
Ann/Cls	**2**	**5**	**10**	**15**
++	**WER: 0.54637**	**WER: 0.56436**	**WER: 0.65037**	**WER: 0.66667**
	CER: 0.23072	**CER: 0.23971**	**CER: 0.29398**	**CER: 0.30622**
+-	**WER: 0.54188**	**WER: 0.54525**	**WER: 0.50815**	**WER: 0.53345**
	CER: 0.23525	**CER: 0.23380**	**CER: 0.22284**	**CER: 0.22798**
-+	**WER: 0.54919**	**WER: 0.57167**	**WER: 0.63969**	**WER: 0.58179**
	CER: 0.23260	**CER: 0.24279**	**CER: 0.29886**	**CER: 0.25674**
-- (Baseline)	**WER: 0.54300**			
	CER: 0.23380			

ar2en				
Ann/Cls	2	5	10	15
++	WER: 0.78057	WER: 0.78407	WER: 0.79455	WER: 0.80014
	CER: 0.23724	CER: 0.23652	CER: 0.23641	CER: 0.26260
+-	WER: 0.774983	WER: 0.77498	WER: 0.75961	WER: 0.77009
	CER: 0.23455	CER: 0.23145	CER: 0.22720	CER: 0.22637
-+	WER: 0.78337	WER: 0.77778	WER: 0.79874	WER: 0.78407
	CER: 0.23797	CER: 0.23621	CER: 0.24294	CER: 0.24024
-- (Baseline)	WER: 0.77638			
	CER: 0.23735			

en2he				
Ann/Cls	2	5	10	15
++	WER: 0.92	WER: 0.9	WER: 0.9	WER: 0.9
	CER: 0.49237	CER: 0.54580	CER: 0.54962	CER: 0.58779
+-	WER: 0.84	WER: 0.84	WER: 0.82	WER: 0.84
	CER: 0.41221	CER: 0.40840	CER: 0.44275	CER: 0.41603
-+	WER: 0.9	WER: 0.9	WER: 0.92	WER: 0.9
	CER: 0.51908	CER: 0.55725	CER: 0.57634	CER: 0.59160
-- (Baseline)	WER: 0.82			
	CER: 0.43130			

Analysis (en2ch)

- Comparing baseline result and the best result (5cls +-):
    - The sequence “ch” could be transliterated into many ways in Chinese, such as ‘奇’ (sounds like ‘chee’), ‘赫’ (sounds like ‘her’), ‘克’ (sounds like ‘ker’) etc. Based on my observations, the machine does not work very well when it tries to predict ‘ch’ in different cases, and this is a general problem that happens in both results.
    - The best result is able to capture ‘th’ as a common English sequence of characters and the transliteration of such sequence is more accurate in various cases, whereas the baseline result **occasionally** breaks ‘th’ down into ‘t’ and ‘h’ and only transliterates ‘t’ into ‘特’(sounds like ter) when ‘th’ appears in the initial positions or the end positions. This might prove that my hypothesis of what clustering does (see below) is on the right track.
- What clustering does:

  I think what the clustering basically does it to assign each character to it ‘popular’ neighbour. For example, in the 5cls English file, most characters that are assigned to cluster 1 happen to have an adjacent character ‘e’ in their right. It indicates that ‘e’ occurs very frequently in the names, and character such as ‘h’ does not have that influence. Certainly, the start symbol ‘^’ and the end symbol ‘$’ should have a significant power because of the way we construct the feature vectors. (This may explain why generally the 2 clusters does not give better results—the ‘^’ and ‘$’ already need 2 clusters in the first place). So in my own understanding , the clustering actually divide characters into groups that **centre on the most frequent characters or their combinations** (if the left-right-bigram is used as a feature). However, this is not a ‘pure’ improvement in the sense that it does not preserve all the right instances in the baseline model, and make progress based on that. **There are 400 words that the best result got right whereas the baseline got wrong. And 288 words that the baseline got right but the best result got wrong.** So if we combine the baseline and the best results, actually we can get a WER=22.9% (There are 5794 words in the development set, 1615 wrong predictions in the best result, if the 288 words are replaced using the baseline result, we can get only 1327 wrong predictions and thus a WER=22.9%).