Spelling Correction for NLP final project

Su Tongtong Liu Menglin

1 Background

Misspelling is a very common phenomena. For example, when a user enters 'carot', he may actually want to return a document containing the word 'carrot'. Google's report points out that when users enter 'british spears', 'britney's spears', 'brand spears' or 'prittany spears'[1], in fact, search engines will treat it as a misspelling of 'britney spears'. In daily life, it is relatively simple for people to correct some misspelling problem, but for machines, there must be an error correction program, otherwise wrong results may appear. Spelling correction mainly focuses on the spelling correction of queries. Compared with Enlish words correction task, it is more difficult for Chinese words.

2 Introduction

We focus on two specific forms of spelling correction that we refer to as term-level correction and context-level correction. In term-level correction, we simply calculate the **distance** between isolated words. We will discuss how to define distance in Section 3.1. Sometimes we have multiple choices, for example, *appls* have same distance from apple and *apply*. Moreover, this type of correction would fail to detect In context-level correction, each term in the query is correctly spelled but the whole sentence is wrong. We need also take context into consideration, for instance, which need pre-trained models of word embeddings. We will discuss how to use it in Section 3.2.

We focus on Chinese spelling correction, which is much more difficult than English, because Chinese characters have more properties like shape and pronounce.

3 Prior knowledge

3.1 distance

We consider 2 kinds of term distance which we have learnt in our class: **Min Edit Distance** and **K-gram index distance**.

3.1.1 Min Edit Distance

Min Edit Distance, which is also called Levenshtein Distance, refers to the minimum number of editing operations required to convert one string to another string. The larger the MED, the more different they are. Permitted editing operations include replacing one character with another, inserting a character, and deleting a character.

3.1.2 K-gram index distance

To calculate the K-gram index distance between two words, first we need to get the K-gram set. For example, 2-gram set for word "world" is wor, orl, rld. Next we calculate the Jaccard coefficient between the two sets(A and B) by the following formula:

$$K - gram\_index\_distance = \frac{|A| \cap |B|}{|A| \cup |B|}$$

$k-gram$ index are used to retrieve vocabulary terms that have many $k-grams$ in common with the query.

3.1.3 Chinese character distance

Chinese spelling correction task is difficult to accomplish, partly because no word delimiters exist among Chinese words and a Chinese word can contain only a single character or multiple characters. Furthermore, there are more than 13 thousand Chinese characters, instead of only 26 letters in English, and each with its own context to constitute a meaningful Chinese word. All these make Chinese spell checking a challengeable task.

3.2 pre-trained bert model

We choose **bert4keras** as our pre-trained package. Bert4keras aims to provide keras users with a way to quickly load common pre-training language models. It's ease for users who are familiar with keras. Also, Bert has developed Chinese pre-trained resource: $chinese_L - 12_H - 768_A - 12$. https://github.com/google-research/bert. This package includes 3 files:

- bert_config.json: It include hyperparameters of bert. We do not use it.

- bert_model.ckpt: This is the checkpoints of pre-trained model. We load it to initialize our model, and use our dataset to start training based on our specific task.

- vocab.txt: This is the dictionary of the model, which is frequently used in our downstream task. We will introduce it in detail later.

3.3 seq2seq task main idea

The seq2seq model is used when the length of the output is uncertain. This situation usually occurs in the task of machine translation. If a Chinese sentence is translated into English, the length of the English sentence may be shorter or longer than Chinese, so the length of the output is uncertain.
Seq2seq is a kind of encoder-decoder structure. Here we look at the common encoder decoder structure. The basic idea is to use two RNN, one as encoder and the other as decoder. The encoder is responsible for compressing the input sequence into a vector of specified length, which can be regarded as the semantics of the sequence. This process is called coding. You can also transform the last hidden state to get the semantic vector, and you can also transform all the hidden states of the input sequence to get the semantic variables.

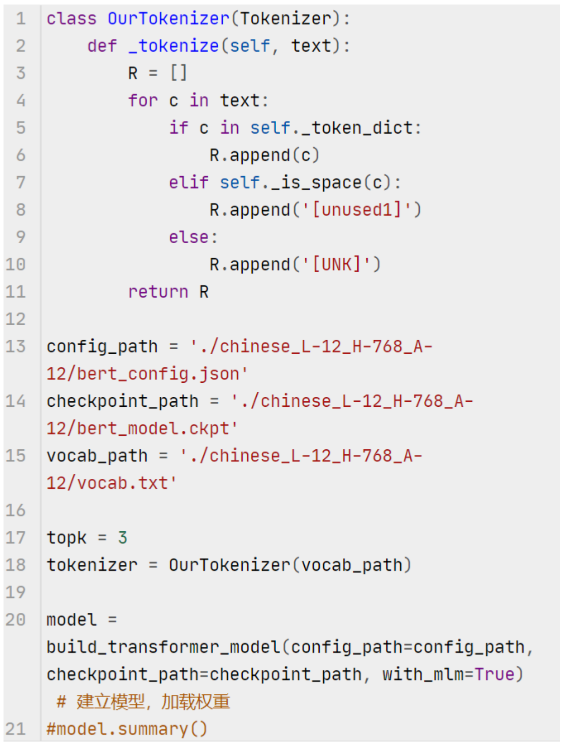
4 Method 1: Directly load Bert together with character similarity

4.1 Step I: Load pre-trained

According to official use documents, we should first set three path, whih is $config\_path$, $checkpoint\_path$, $dict_path$, for each is used to load $bert\_config.json$, $bert\_model.ckpt$ and $vocab.txt$. Then we build a word splitter through function $Tokenizer(dict\_path, do\_lower\_case = True)$. Next we load the model through function $build\_transformer\_model$ $(config\_path, checkpoint\_path)$.The code are shown in figure 1.

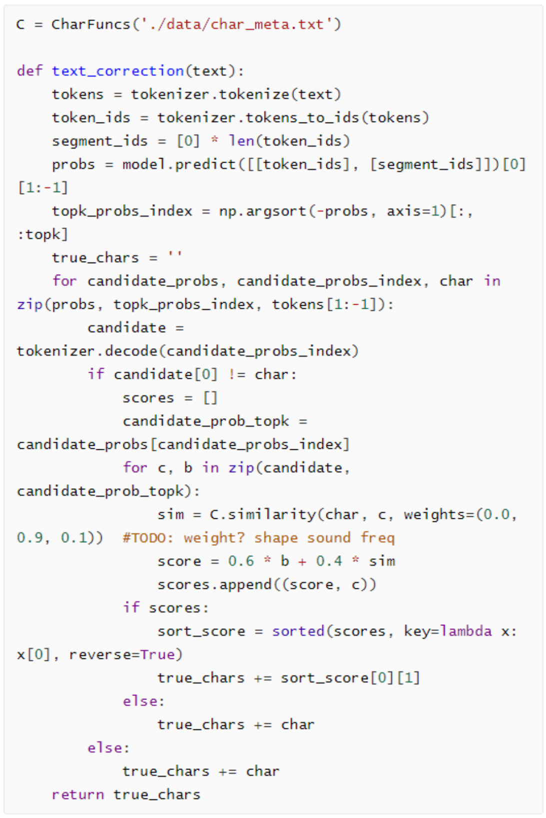
4.2 Step II: Chinese Character

We consider properties of Chinese character: shape(偏旁部首), sound and frequency. We define a class **CharFuncs** in charsim.py. It has functions to calculate the shape_similarity, sound_similarity and frequency_similarity between two chinese characters. In all, it can calculate the total similarity by weighted summation and the weight is confirmed by us. Based on CharFuncs, we define a text correction function **text_correction**. The code are shown in figure 2.



```python
class OurTokenizer(Tokenizer):
    def _tokenize(self, text):
        R = []
        for c in text:
            if c in self._token_dict:
                R.append(c)
            elif self._is_space(c):
                R.append('[unused1]')
            else:
                R.append('[UNK]')
        return R

config_path = './chinese_L-12_H-768_A-
12/bert_config.json'
checkpoint_path = './chinese_L-12_H-768_A-
12/bert_model.ckpt'
vocab_path = './chinese_L-12_H-768_A-
12/vocab.txt'

topk = 3
tokenizer = OurTokenizer(vocab_path)

model =
build_transformer_model(config_path=config_path,
checkpoint_path=checkpoint_path, with_mlm=True)
 # 建立模型，加载权重
#model.summary()
```

Figure 1: Load pre-trained model



```python
C = CharFuncs('./data/char_meta.txt')

def text_correction(text):
    tokens = tokenizer.tokenize(text)
    token_ids = tokenizer.tokens_to_ids(tokens)
    segment_ids = [0] * len(token_ids)
    probs = model.predict([[token_ids], [segment_ids]])[0]
[1:-1]
    topk_probs_index = np.argsort(-probs, axis=1)[:,
:topk]
    true_chars = ''
    for candidate_probs, candidate_probs_index, char in
zip(probs, topk_probs_index, tokens[1:-1]):
        candidate =
tokenizer.decode(candidate_probs_index)
        if candidate[0] != char:
            scores = []
            candidate_prob_topk =
candidate_probs[candidate_probs_index]
            for c, b in zip(candidate,
candidate_prob_topk):
                sim = C.similarity(char, c, weights=(0.0,
0.9, 0.1))  #TODO: weight? shape sound freq
                score = 0.6 * b + 0.4 * sim
                scores.append((score, c))
            if scores:
                sort_score = sorted(scores, key=lambda x:
x[0], reverse=True)
                true_chars += sort_score[0][1]
            else:
                true_chars += char
        else:
            true_chars += char
    return true_chars
```

Figure 2: text correction

4.3 Code analysis

For the first 100 data, we mainly use them to do detection task by inputing the right sentence and observe the output. If the output is the same as input, the detection task is successful ($d\_num + 1$), otherwise it is failed. For the remaining test data, we input the wrong sentence and observe the output. If the output is the same, it means the detection task failed ($fn\_num + 1$) and if the output is the correct sentence, it means the correction task is successful ($c\_num + 1$). In other case, it means the detection task is successful but the correction task is failed. We record these numbers to calculate $detection\_prec$, $detection\_recall$, $detection\_f1$, $correction\_acc$. In the same time, we try different parameter groups to achieve best test effect.

5 Method 2: Fine-tuning using MLM

The problem of directly using pre-trained model is that, it only focus on the character-level similarity, cannot focus on specific knowledge from our designed training set. For example, '自然语言处理' and '自然预言处理' are all regarded as correct for basic model. Can we introduce method that can continue training model based on our prepared training set? Actually, it is the mainstream of nowaday most of NLP task. For different concrete task, like sentiment analysis, translation or parsing, the first step is always download pre-trained embedding from Internet, which is well-trained by big company like google. The complete workflow is shown as figure3

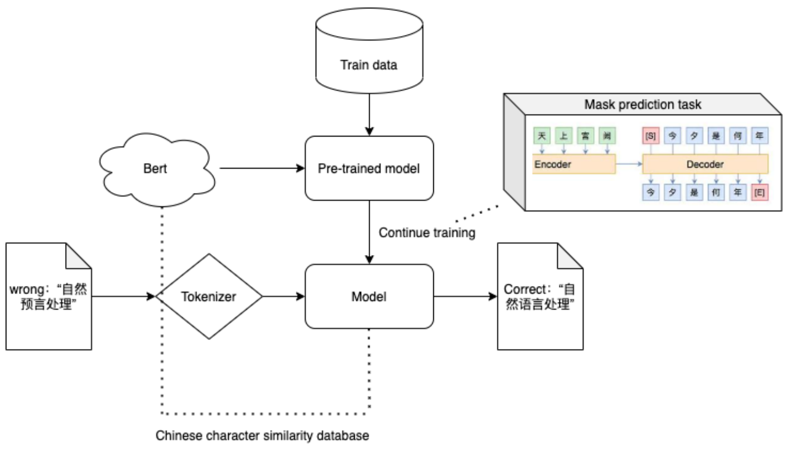


Figure 3: MLM model workflow

5.1 Step I: Train using MLM

First, we introduce 5 special token:

- [PAD] Padding the sentence with [PAD] tokens so that the total length equals to the maximum length.

- [UNK] For tokens which do not appear in the original vocabulary, it is designed that they should be replaced with token [UNK], which stands for unknown token.

- [CLS] Adding the [CLS] token at the beginning of the sentence.

- [SEP] Adding the [SEP] token at the end of the sentence.

- [MASK] Mask and prediction is the pseudo-task to get embedding. In our task, we replace every character in wrong sentence as [MASK].

5.2 Seq2Seq architecture

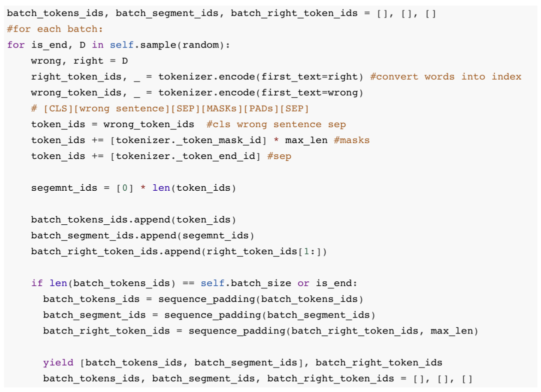
Spelling correction task can also be regarded as a seq2seq task like translation and tagging: the input sequence is wrong sentence, and the output sequence is the right sentence.

5.3 Generating dataset

Since our specific task is to correct mistakes, the format of our sequence is: [cls] wrong sentence [sep] [mask][mask].. [sep] The format of our training files is like: [[right_0, wrong_0],[right_1,wrong_1]...[right_n, wrong_n]] Wrong sentences and corresponding right sentences are in pairs.

5.4 Code analysis

We introduce the class **MyDataGenerator**, which can convert original sentence pairs into fixed-length array. Available function **encode** from package bert4keras can convert characters into ids.

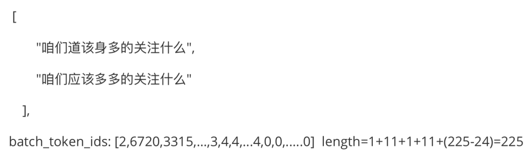


```
batch_tokens_ids, batch_segment_ids, batch_right_token_ids = [], [], []
#for each batch:
for is_end, D in self.sample(random):
    wrong, right = D
    right_token_ids, _ = tokenizer.encode(first_text=right) #convert words into index
    wrong_token_ids, _ = tokenizer.encode(first_text=wrong)
    # [CLS][wrong sentence][SEP][MASKs][PADs][SEP]
    token_ids = wrong_token_ids  #cls wrong sentence sep
    token_ids += [tokenizer._token_mask_id] * max_len #masks
    token_ids += [tokenizer._token_end_id] #sep

    segemnt_ids = [0] * len(token_ids)

    batch_tokens_ids.append(token_ids)
    batch_segment_ids.append(segemnt_ids)
    batch_right_token_ids.append(right_token_ids[1:])

    if len(batch_tokens_ids) == self.batch_size or is_end:
      batch_tokens_ids = sequence_padding(batch_tokens_ids)
      batch_segment_ids = sequence_padding(batch_segment_ids)
      batch_right_token_ids = sequence_padding(batch_right_token_ids, max_len)

      yield [batch_tokens_ids, batch_segment_ids], batch_right_token_ids
      batch_tokens_ids, batch_segment_ids, batch_right_token_ids = [], [], []
```

Figure 4: generator

Batchsize equals to 8, so after every 8 train pairs, we will do a **padding**, which means add [PAD] to ensure every sequence having same length(225).

Here is one case (Figure7):

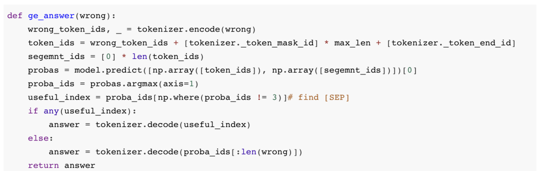


```
[
    "咱们道该身多的关注什么",
    "咱们应该多多的关注什么"
  ],
batch_token_ids: [2,6720,3315,...,3,4,4,...4,0,0,.....0]  length=1+11+1+11+(225-24)=225
```

Figure 5: demo

The length of mask is the same as the wrong sentence, and our task is to predict [mask] as the correct sentence.

Then we introduce decoder. Available function **predict** from package bert4keras can calculate a probabilistic matrix in shape(lens(tokens)*sizeof(totalwords), here is 64*13585). Then we use argmax to find the most likely sequence ids. At last, we use function **decode** to transform id into character as our correcting result.



```python
def ge_answer(wrong):
    wrong_token_ids, _ = tokenizer.encode(wrong)
    token_ids = wrong_token_ids + [tokenizer._token_mask_id] * max_len + [tokenizer._token_end_id]
    segemnt_ids = [0] * len(token_ids)
    probas = model.predict([np.array([token_ids]), np.array([segemnt_ids])])[0]
    proba_ids = probas.argmax(axis=1)
    useful_index = proba_ids[np.where(proba_ids != 3)]# find [SEP]
    if any(useful_index):
        answer = tokenizer.decode(useful_index)
    else:
        answer = tokenizer.decode(proba_ids[:len(wrong)])
    return answer
```

Figure 6: predict and decode to get answer

6 Experiment

6.1 training set and metric

We generate our own dataset in the format of json. For each pair, the first sentence is wrong, and the second is correct. We totally have 223 sentences, and after our pre-processing, the first 100 sentences are correct, while the left have 1 or 2 character mistakes. We also have the ground-truth correction result.

Actually, our correction task has a relative task: detection, which is a 2-class classification problem. Of course we can first design a classification model to do this task first, and if the prediction is True(exist mistake), we can put it into our correction model. However, we can generalize detection to correction task: if the correction result is the same as origin sentence, we can say the classification result is False(no mistake).
For detection task, we use typical 2-class classification metrics: precision, recall and F1 score. For correction task, we only concern about precision(if the correction result is correct).

6.2 Experiment Settings

6.2.1 Basic model

We load the pre-trained model, each time we choose a new set of parameter group. The parameter group is $weight = \{shape, sound, frequency\}$. However, the result shows that no matter how we modify the parameters, the results will not change.

6.2.2 MLM model

We set training epoch as 10. Every 2 epochs we save a checkpoints(.h5, only the weights of the model). We use MaskedCrossEntropy(MCE) as our loss. This loss introduce a padding matrix to avoid calculating the loss of [PAD], which is meaningless. The loss during training is shown as Tabel1.

Table 1: MLM model result(Max epochs= 10)

Epochs	**Loss**
2	1.1156
4	0.6665
6	0.3851
8	0.1047
10	0.0732

shape,sound,freq	detection prec	detection recall	detection F1	correction acc
0.8, 0.2, 0.0	0.6731	0.3500	0.4605	0.0813
0.9, 0.1, 0.0	0.6731	0.3500	0.4605	0.0813
0.8, 0.1, 0.1	0.6731	0.3500	0.4605	0.0813
0.5, 0.2, 0.3	0.6731	0.3500	0.4605	0.0813
0.5, 0.4, 0.1	0.6731	0.3500	0.4605	0.0813

: Table 3: Basic model result

6.2.3 Evaluation

For method 1, we can see that the four metric do not vary with the parameters no matter how we change the parameters. We tried several parameter group, the results are showed in Tabel 3. Also, the results is not good especially for the correction task. We analysis that the reason why the basic model performs so bad in correction task is maybe it lacks training step, so the training set has no effect on the model. For method 2, from Table 4, we can see that correction task perform much poorer than detection task, even though it is our direct task. We find that for detection task, the performance is good enough when epoch is only 2, while for correction task, the performance is keeping on improving.

Epochs	detection prec	detection recall	detection F1	correction prec
2	**0.9444**	0.8292	0.8831	0.2124
4	0.7593	0.8211	0.7890	0.3170
6	0.6686	0.9024	0.7681	0.2439
8	0.7762	0.9024	0.8345	0.3495
10	0.8106	0.8699	**0.8392**	0.5759
12	0.7044	**0.9105**	0.7943	**0.5872**

: Table 4:MLM Correction and detection result

6.3 Experiment result

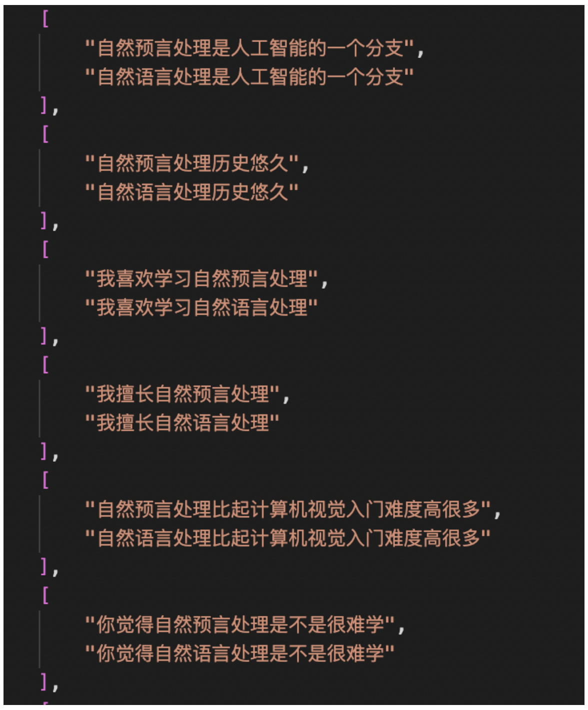
6.3.1 case study

For the wrong sentence: '追风少俊年王俊凯', we show for every 2 epochs the correcting result and loss change(Tabel 2). We can see that the loss is decreasing substantially and the correcting results is eventually right.
If we use bacis model to correct the sentence: '追风少俊年王俊凯', the result is '。风少俊年王俊凯'. If we enter '。风少俊年王俊凯' to the bacis model, the result is still '。风少俊年王俊凯'. Thus we can see that the basic model shows relatively poor result in correction task.

Table 2: MCE during training(epochs= 10)

Epochs	correction	Loss
2	追风少年年王俊凯	1.1156
4	追风少年王王俊	0.6665
6	追风少年王俊凯凯凯	0.3851
8	追逐少年王俊凯	0.1047
10	追风少年王凯凯	0.0732
12	追风少年王俊凯	0.0419

We also interest in the scenario when a sentence contains of correct **phrases** but when we combine them together, the whole sentence is wrong, e.g., '自然预言处理真太是有趣啦'. The correction result of this sentence generrated by methos 2 is ’自然预言处理真的是有趣啦’, it do not consider ’自然语言处理’ as a whole, since it is not included in the training dataset. If we add many sentence consist of ’自然语言处理’, can this problem be solved? The result shows that if we add just 6 training pairs including ’自然语言处理’, the sentence can be corrected successfully!



```
[
    "自然预言处理是人工智能的一个分支",
    "自然语言处理是人工智能的一个分支"
],
[
    "自然预言处理历史悠久",
    "自然语言处理历史悠久"
],
[
    "我喜欢学习自然预言处理",
    "我喜欢学习自然语言处理"
],
[
    "我擅长自然预言处理",
    "我擅长自然语言处理"
],
[
    "自然预言处理比起计算机视觉入门难度高很多",
    "自然语言处理比起计算机视觉入门难度高很多"
],
[
    "你觉得自然预言处理是不是很难学",
    "你觉得自然语言处理是不是很难学"
],
```

Figure 7: New-added train pairs including key phrase :自然语言处理/自然预言处理. The total pairs number is 6.

7 Summary

In addition, we can see that MLM model performs much better than basic model in completing both dectetion task and correction task. Despite this, spelling correction task for chinese is still not easy. The main problem in our task is recognizing fixed nouns. Also, we hope to try more algorithms in this task in the future.

References

[1] Yuen-Hsien Tseng, Lung-Hao Lee, Li-Ping Chang, and Hsin-Hsi Chen. Introduction to sighan 2015 bake-off for chinese spelling check. In *Proceedings of the Eighth SIGHAN Workshop on Chinese Language Processing*, pages 32–37, 2015.