Pinyin-BART: An End-to-End Chinese Input Method

Anonymous ACL submission

Abstract

A Chinese Input Method Engine helps users convert a keystroke sequence into the desired Chinese character sequence. It is usually a cascaded process in which the original input sequence is firstly corrected to remove typos, then segmented into the pinyin token sequence, and finally converted into a Chinese character sequence. Errors are prone to accumulate and propagate in that pipeline. This paper summarizes that process as a Key-to-Character (K2C) conversion task and solves it in a unified end-to-end way. Pinyin-bart is proposed which can effectively solve the error propagation problem and improve the IME engine performance significantly in experiments. Moreover, we model the user real input behaviors and design a method to generate the training corpus with typos for the K2C task. It further improves the robustness of Pinyin-bart. Finally, we design a non-autoregressive (NAR) decoder for Pinyin-bart and obtain 9x+ acceleration with limited performance degradation, which makes the deployment possible on the commercial input software.

1 Introduction

Some of languages, such as Chinese, Japanese and Thai language, can not be input directly through the standard keyboard. Users type in these languages via some commercial input software, such as Microsoft Input Method (Gao et al., 2002), Google Chinese Input Method1, Sogou Input Method2, Baidu Input method3, Huawei Celia Keyboard4, and so on. Pinyin is the official romanization representation for Chinese language. It’s natural for a user to type in pinyin through the keyboard. And the input software converts the pinyin into the character sequence. As the figure 1 shows, a user inputs a keystroke sequence of “woainizongguo”, and the software segments it into the pinyin sequence “wo’ai’ni’zong’guo” then converts it into the Chinese character sequence that user desires “我爱你中国 (I love you China)”.

Specifically, as the figure 2 shows, the IME engine takes it as a cascaded process. Firstly, the correction module corrects the typos in the original keystroke sequence. In the example of the figure 1, the blade-alveolar sound of ‘zong’ is corrected into the cacuminal sound of ‘zhong’. It is usually implemented by some rule system for efficiency. Secondly, the modified keystroke sequence is segmented into the pinyin token sequence. For example, “woainizongguo” is segmented into “wo’ai’ni’zhong’guo”. The tokenizer is usually implemented by some Chinese word segmentation algorithm, i.e. the Maximum Matching (MM) algorithm. Lastly, the pinyin sequence is converted into the character sequence, which is called the Pinyin to Character (P2C) conversion task (Zhang et al., 2019a; Yao et al., 2018; Xiao et al., 2007). It is usually resolved as a sequence labeling task by the Ngram language model (Goodman, 2001) together with the Viterbi search algorithm (Viterbi, 2006).

In the above process, the error in the previous step is prone to accumulate and propagate to the later step, which hurts the IME engine performance badly as presented in the later experiments. In this paper, we summarizes those steps into a unified end-to-end process named the Key-to-Character (K2C) task and proposes Pinyin-bart to solve it. As

1https://www.google.com/inputtools/
2https://pinyin.sogou.com/
3https://shurufa.baidu.com/
4https://consumer.huawei.com/uk/community/details/App-Gallery-Celia-Keyboard-is-now-available/topicId_48409/

The screenshot is from Sogou Input Method software.
far as we know, it’s the first work to build the IME engine in an end-to-end way. We summarize the main contributions of this paper as follows:

- We propose Pinyin-bart to solve the K2C task and build the IME engine in an end-to-end way, which effectively resolves the error propagation problem in the cascaded IME engine. As far as we know, it’s the first end-to-end IME engine.

- We model the user input behavior and design a method to generate the massive corpus with typos automatically for the K2C task, which further improve the robustness of Pinyin-bart.

- We design the NAR decoder for Pinyin-bart and boost the inference speed significantly with only little performance degradation.

2 Method

In this section, we describe the details about Pinyin-bart. Firstly, we introduce the K2C task formally in the section 2.1. Then we present how the Pinyin-bart is implemented in the section 2.2. Lastly, we describe the method that models user input behavior and generates the massive corpus with typos in the section 2.3.

2.1 The K2C Conversion Task

As illustrated in the figure 2, the K2C conversion task is to convert the user keystroke sequence from keyboard directly into the Chinese sentence. Formally, $k_1, k_2, ..., k_n$ is the keystroke sequence. They are converted into the character sequence of $c_1, c_2, ..., c_m$ in the K2C conversion task. Usually the value of $m$ is smaller than $n$ since one Chinese character corresponds one pinyin token which is composed of multiple letters. The task can be resolved in a cascaded way as most of the commercial input software does, or in an end-to-end way by Pinyin-bart in this paper.

2.2 Pinyin-bart

We build Pinyin-bart based on the standard bart model (Lewis et al., 2020). To fit for the K2C task, we do some modifications in several aspects, including the training paradigm as described in the section 2.2.1, the embedding layer as described in the section 2.2.2 and the NAR decoder described in the section 2.2.3.

2.2.1 The Training Paradigm

The standard bart adopts the pretrain-then-finetune paradigm, like most of the transformer model does. It firstly pretrains the model on the massive unlabeled corpus by some self-supervised learning tasks, such as Masked Language Model (MLM),
Text Infilling, Sentence Permutation and so on. Then the model is finetuned on the labelled corpus on the target task, such as SQuAD (Rajpurkar et al., 2016), MNLI (Williams et al., 2018), XSum (Narayan et al., 2018), and so on. The pretrain process leverages the general knowledge contained in the unlabeled corpus, which boosts the performance significantly on the target tasks. As described in the section 2.3 later, we design the method to create the massive labelled corpus for the K2C task automatically. Therefore, we train the Pinyin-bart directly on the target K2C task instead of the pretrain-then-finetune paradigm.

2.2.2 The Embedding Layer

Different from the standard bart, there is no segment embedding in the embedding layer of Pinyin-bart as illustrated in the figure 3. It is because there is no pretrain process in Pinyin-bart and it no longer needs the self-supervised learning task, especially the sentence-level pretrain task. Besides, Pinyin-bart takes the keystroke sequence as input rather than the subword sequence as the standard bart dose. There are only 26 individual letters which is 3x order of magnitude smaller than the number of subword (more than 50,000) in the standard bart. Thus the size of embedding layer of Pinyin-bart is much smaller than the standard bart.

2.2.3 The NAR Decoder

The standard bart adopts the autoregressive decoder like GPT which predicts the current token based on the previous one. The advantage is to leverage the dependency between tokens in sequence, whereas it’s pretty slow during the inference, which hinders its deployment in the commercial input software. The NAR decoder is proposed in the machine translation domain (Gu et al., 2018). It makes the independent assumption on the tokens in sequence, which makes the inference process parallel so that accelerates the inference significantly. In this paper, we design the NAR decoder for Pinyin-bart.

As described in the figure 3, we firstly replace the GPT-like decoder of single direction attention in the standard bart with the Bert-like decoder of bidirectional attention in Pinyin-bart, which can leverage the parallel computation in GPU. Secondly, we add a predictor to predict the length of target sequence. Specifically, we add a mean pooling layer stacked with a regression layer on the top of the encoder. Thirdly, to train the Pinyin-bart model, the cross-entropy (ce) loss is adopted for the target sequence prediction task, and the mean square error (mse) loss is adopted for the length prediction task. They are weighted combined together, as shown in the formula 1.

\[
    \text{loss}_{\text{total}} = \lambda_1 \times \text{loss}_{\text{ce}} + \lambda_2 \times \text{loss}_{\text{mse}}
\]

During the inference, the tokens in the target sequence are generated parallel, and the target length is predicted as well. The length is rounded off from float to the integer value. Then the target sequence is simply truncated by that length.

2.3 Generating Massive Labelled Corpus

We generate the massive labelled corpus for the K2C task. The whole process is described in the figure 4.

Firstly, the text in Chinese corpus, i.e. the sentence of “我爱你中国 (I love you China)”, is converted into the pinyin token sequence, i.e. “wo’ai’ni’zong’guo”. This task is called Text-to-Pinyin conversion which can achieve more than 99.9% accuracy (Zhang and Laprie, 2003). In this way, we can get the massive pinyin corpus automatically. Secondly, user does not type in any separator to split the pinyin token explicitly during its input process in reality, so we combine the pinyin tokens in a sequence together into the keystroke sequence. The “wo’ai’ni’zong’guo” is then combined into ‘woainizhongguo’. Thirdly, some kind of noise is added into the keystroke sequence so as to simulate user’s typos. Finally we get the parallel corpus with the Chinese character sequence as well as the keystroke sequence with typos.

To add noise to the keystroke sequence, we select some positions randomly from the original sequence. Then three operators are applied on the letters of these positions with equal probability, including ’Delete’, ’Insert’ and ’Replace’. Some probability distribution is required to guide the ’Insert’ and ’Replace’ operator, i.e. to insert which letter before the current position. The uniform distribution is the most straightforward choice. However, it’s sub-optimal because it does not take the consideration of the keyboard layout and the user’s behavior in reality. For example, when user types in the letter of ‘z’ in ‘zong’, it is prone to mistype it as ’x’ instead of ’p’ because the position of ’x’ is much closer to ’x’ on the keyboard layout than ’p’ dose. Besides, the typos of one user are usually different from another user due to their different input habits. In this paper, we collect the user type-in
behaviors in reality. Some of them are visualized as the points cloud shown at the top left of figure 4. Based on these data, we build the Gaussian model for each key on the keyboard layout, as the formula shows below:

\[ f(x; \mu, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x - \mu)^2}{2\sigma^2}} \]  

(2)

According to the Gaussian model, we can calculate the probability that the current key is mis-typed to any other key. And we finally generate the typo noise according to that mis-type probability matrix, as illustrated at the upper half part of figure 4.

### 3 Experiment

#### 3.1 Data Set Preparation

As far as we know, there is no public benchmark for the Chinese Pinyin input method. So we build our own data set and will make it public to the community later. More than 2.6 million articles are collected from the Chinese news websites. We firstly segment them into sentence by the punctuation list including comma, period, and so on. Then we filter the characters which there are no pinyin corresponded to. Thirdly, these sentences are further segmented by a max length (i.e. 16 in our experiment) because user only types in a few Chinese characters at one time. Lastly, we make them as the labelled corpus as described in the section 2.3. Most of the corpus are taken as the training corpus, and another one thousand disjoint articles are taken as the test corpus, as described in the table 1.

Besides, to evaluate the performance of the cascade IME engine, we build several test corpus with different degree of noise:

- **No Typos and No Segment Errors.** In the first one, we assume that there is no typo from user’s input and the pinyin tokenizer in the figure 2 works perfectly. It looks like “我爱你中国 (wo’ai’ni’zong’guo)”. It is a total clean environment and the only factor that matters the IME performance is language model. It can be taken as the upper bound of the IME engine performance in reality. We get this corpus by processing only the first step of figure 4.

- **No Typos BUT Segment Errors.** In the second corpus, we assume that there is no typo but the pinyin tokenizer works probably with errors. It looks like “我爱你中国 (wo’a’in’zong’guo)”. It is a possible situation if the user types carefully and precisely. We can get it by re-segmenting the combined keystroke sequence automatically after the second step of figure 4 by some real tokenizer, i.e. the MM algorithm.

- **Typos and Segment Errors.** In the last one, we assume both the typo and the segmenting error, which is the situation in the real world. It might look like “我爱你中国 (wo’ao’ni’zong’guo)”. We can get it by re-segmenting the sequence containing noises after the third (last) step of of the figure 4.

During evaluating, we apply language model directly on these kinds of corpus to simulate the

---

Table 1: The Detailed Information of Corpus

<table>
<thead>
<tr>
<th>Corpus</th>
<th>#Articles</th>
<th>#Chars</th>
<th>#Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>2,603,869</td>
<td>2,432,585,138</td>
<td>9.7G</td>
</tr>
<tr>
<td>Test</td>
<td>1000</td>
<td>926,792</td>
<td>3.7M</td>
</tr>
</tbody>
</table>

---

6 We get these data under the users’ authorization.
performance of the cascaded IME engine in various noisy environment.

### 3.2 Evaluation Metrics

We use the character-based precision to evaluate the performance of the IME engine. It is defined as the ratio that the IME engine converts to the Chinese character correctly, as described in the formula 3.

\[
\text{Precision}_{\text{char-based}} = \frac{\# \text{correct converted char}}{\# \text{total converted char}} \quad (3)
\]

### 3.3 Baseline Models and Experiment Settings

The cascaded IME engine is taken as the baseline model, and is evaluated on the corpus with different degree of noise as described in the section 3.1. Several kinds of language models are integrated respectively into the cascaded IME engine:

- **Bigram.** Bigram is the De facto model adopted widely in the commercial IME engine. We build the bigram model on the lexicon of the Table of General Standard Chinese Characters which contains more than 6 thousand Chinese frequent characters. No pruning strategy is adopted since the scale of training corpus is large enough.

- **LSTM.** LSTM is reported to obtains better performance than the bigram model (Zhang et al., 2019b; Yao et al., 2018; Malhotra et al., 2015). In out implement of the LSTM model, both the embedding size and the hidden size are 256, and the learning rate is $5e^{-4}$. The batch size is $2k$ and the epoch number is 10.

- **Bart.** We use the standard bart in the sequence-to-sequence way. The pinyin token sequence is taken as input, and the Chinese character sequence is taken as output. It is trained from scratch directly on the P2C task. We follow most of the specifications in the paper (Lewis et al., 2020), except that the max sequence length is set to 16 instead of 512. The epoch number is 10.

For the Pinyin-bart model, the keystroke sequence is taken as input. It is trained directly on the K2C task as described in the section 2.2.1. The experimental settings are exactly the same as the standard bart model. In the formula 1, the value of $\lambda_1$ is 1 and the value of $\lambda_2$ is 0.01.

### 3.4 Experimental Results on the K2C Task

The experimental results are presented in the table 2. Two ratios of typo noises (1% and 5%) are added into the corpus.

Firstly, let’s take a quick look at the results under the clean environment (no typo and no segment error). The bigram model obtains 84.56% precision and the LSTM model gets a better result of 89.71% (5.15% ↑) as reported in the previous articles (Zhang et al., 2019b). The standard bart model achieves 96.97% which outperforms the above two models (12.41% ↑ and 7.26% ↑) significantly. It proves that language model plays a crucial role in the cascaded IME engine and its capacity can improve the performance greatly.

Secondly, the performance of the cascaded IME engine decreases badly in the noisy environment. Taking the bigram model as an example, the precision decreases from 84.56% to 79.30% (5.26% ↓) under the segment errors, and further to 66.87% (17.69% ↓) under the typo errors as well, and lastly to 37.75% (46.81% ↓) as the typo ratio increases. The similar results can be observed in the LSTM model and even in the powerful bart model.

Thirdly, Pinyin-bart achieves much higher precision than the standard bart in the cascade IME engine under the same noisy environment. For example, with the 1% typos and the segment error, Pinyin-bart gets 94.86% precision which is much higher (11.13% ↑) than 83.73% of the standard bart model. The improvement is further expanded to 35.10% ↑ as the typo ratio increase to 5%. These results are also significantly higher than the bigram model and the LSTM model. It proves that Pinyin-bart can effectively avoid the error propagation problem and performs much more robust than the cascaded IME engine.

### 3.5 Effectiveness of Modeling User Behaviors

In the section 2.3, we model user’s input behavior and generate the typos for the training corpus of the K2C task. In this section, we compare it with the method that adds typos by the uniform distribution. The results are presented in the table 3.

As we can see, Pinyin-bart achieves better performances. As the typo ratio increases from 1% to 5%, the improvement rises from 2.29% to 5.49% accordingly. It proves that our method can generate
Table 2: The Experimental Results on the K2C Task

<table>
<thead>
<tr>
<th>Model</th>
<th>Typo Error</th>
<th>Segment Error</th>
<th>Precision</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bigram no</td>
<td>no</td>
<td>no</td>
<td>84.56%</td>
<td>NV</td>
</tr>
<tr>
<td>Bigram no</td>
<td>yes</td>
<td>yes</td>
<td>79.30%</td>
<td>5.26%↓</td>
</tr>
<tr>
<td>Bigram 1%</td>
<td>yes</td>
<td>yes</td>
<td>66.87%</td>
<td>17.69%↓</td>
</tr>
<tr>
<td>Bigram 5%</td>
<td>yes</td>
<td>yes</td>
<td>37.75%</td>
<td>46.81%↓</td>
</tr>
<tr>
<td>LSTM no</td>
<td>no</td>
<td>no</td>
<td>89.71%</td>
<td>5.15%↑</td>
</tr>
<tr>
<td>LSTM no</td>
<td>yes</td>
<td>yes</td>
<td>84.96%</td>
<td>4.75%↓</td>
</tr>
<tr>
<td>LSTM 1%</td>
<td>yes</td>
<td>yes</td>
<td>66.87%</td>
<td>22.84%↓</td>
</tr>
<tr>
<td>LSTM 5%</td>
<td>yes</td>
<td>yes</td>
<td>51.75%</td>
<td>37.96%↓</td>
</tr>
<tr>
<td>Bart no</td>
<td>no</td>
<td>no</td>
<td>96.97%</td>
<td>12.41%↑</td>
</tr>
<tr>
<td>Bart no</td>
<td>yes</td>
<td>yes</td>
<td>93.05%</td>
<td>3.92%↓</td>
</tr>
<tr>
<td>Bart 1%</td>
<td>yes</td>
<td>yes</td>
<td>83.73%</td>
<td>13.24%↓</td>
</tr>
<tr>
<td>Bart 5%</td>
<td>yes</td>
<td>yes</td>
<td>57.39%</td>
<td>39.58%↓</td>
</tr>
<tr>
<td>Pinyin-bart 1%</td>
<td>yes</td>
<td>yes</td>
<td>94.86%</td>
<td>11.13%↑</td>
</tr>
<tr>
<td>Pinyin-bart 5%</td>
<td>yes</td>
<td>yes</td>
<td>92.49%</td>
<td>35.10%↑</td>
</tr>
</tbody>
</table>

Table 3: Effectiveness of Modeling User Behaviors. Pinyin-bart-uni is trained with the typos generated by the uniform distribution. Pinyin-bart is trained with the typos generated by the Gaussian model.

<table>
<thead>
<tr>
<th>Model</th>
<th>Typo Error</th>
<th>Segment Error</th>
<th>Precision</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pinyin-bart-uni 1%</td>
<td>yes</td>
<td>yes</td>
<td>92.57%</td>
<td>NV</td>
</tr>
<tr>
<td>Pinyin-bart-uni 5%</td>
<td>yes</td>
<td>yes</td>
<td>87.00%</td>
<td>NV</td>
</tr>
<tr>
<td>Pinyin-bart 1%</td>
<td>yes</td>
<td>yes</td>
<td>94.86%</td>
<td>2.29%↑</td>
</tr>
<tr>
<td>Pinyin-bart 5%</td>
<td>yes</td>
<td>yes</td>
<td>92.49%</td>
<td>5.49%↑</td>
</tr>
</tbody>
</table>

the typos closer to the reality, and boost the IME engine’s performance.

3.6 Effectiveness of the NAR Decoder

In this section, we compare the NAR decoder with the AR decoder in Pinyin-bart on both the performance and the inference speed. The experimental results are presented in the table 4.

Compared to the AR-model, there is 0.03% performance drop from the NAR-model under the 1% typo ratio, and further 0.91% drop under the 5% typo ratio. Considering the fact that the precision of Pinyin-bart has already exceeded 90%, that degradation is very slightly. However, the inference process is accelerated greatly. The time to infer one token drops from 15.66ms to 1.60ms under the 1% typo ratio, and drops from 16.09ms to 1.73ms under the 5% typo ratio, which is reduced by more than 9 times. It makes the deployment of Pinyin-bart possible to the commercial input method software.

4 Related Works

4.1 Language model

Language model predicts the current word probability by its previous words. It plays an essential role in the P2C task in the IME engine. The dominant model is the Ngram model (Bahl et al., 1983). However, its simplicity and low capacity limits its performance. In recent years, RNN is proposed to improve the performance by modeling longer history information (Kalchbrenner and Blunsom, 2013). Variant network architectures are proposed to solve the vanishing gradient problem and the exploding gradient problem, such as LSTM (Malhotra et al., 2015; Graves et al., 2013), GRU (Cho et al., 2014), and so on. Yao et al. (2018) replaces the Ngram model with the LSTM model in the IME engine and get performance improvement both in the candidate prompt task and in the P2C task. It further proposes an incremental selective softmax method to solve the LSTM efficiency problem in the Viterbi algorithm. Zhang et al. (2019b) applies LSTM in a sequence-to-sequence way in the P2C task, and verify it in a smart sliding input method. Zhang et al. (2019a) designs a novel online learn-
Table 4: Comparison between autoregressive Pinyin-bart and non-autoregressive Pinyin-bart. AR-model is the Pinyin-bart with the autoregressive decoder as the standard bart does. NAR-model is the Pinyin-bart with the non-autoregressive decoder as described in the section 2.2.3

<table>
<thead>
<tr>
<th>Model</th>
<th>Typo Error</th>
<th>Segment Error</th>
<th>Precision</th>
<th>Improvement</th>
<th>ms/token</th>
<th>Speedup Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-model</td>
<td>1%</td>
<td>yes</td>
<td>94.86%</td>
<td>NV</td>
<td>15.66</td>
<td>NV</td>
</tr>
<tr>
<td>AR-model</td>
<td>5%</td>
<td>yes</td>
<td>92.49%</td>
<td>NV</td>
<td>15.66</td>
<td>NV</td>
</tr>
<tr>
<td>NAR-model</td>
<td>1%</td>
<td>yes</td>
<td>94.83%</td>
<td>0.03%↓</td>
<td>1.60</td>
<td>9.78↑</td>
</tr>
<tr>
<td>NAR-model</td>
<td>5%</td>
<td>yes</td>
<td>91.58%</td>
<td>0.91%↓</td>
<td>1.73</td>
<td>9.30↑</td>
</tr>
</tbody>
</table>

References


Zhuosheng Zhang, Zhen Meng, and Hai Zhao. 2019b. A smart sliding chinese pinyin input method editor on touchscreen.