INTERSPEECH 2020
October 25–29, 2020, Shanghai, China

g2pM: A Neural Grapheme-to-Phoneme Conversion Package for Mandarin Chinese Based on a New Open Benchmark Dataset

Kyubyong Park[^1*], *Seanie Lee*[#2*]

[^]Kakao Brain, South Korea
[#]KAIST, South Korea

[1]kyubyong.park@kakaobrain.com, [2]lsnfamily02@kaist.ac.kr

Abstract

Conversion of Chinese graphemes to phonemes (G2P) is an essential component in Mandarin Chinese Text-To-Speech (TTS) systems. One of the biggest challenges in Chinese G2P conversion is how to disambiguate the pronunciation of polyphones—characters having multiple pronunciations. Although many academic efforts have been made to address it, there has been no open dataset that can serve as a standard benchmark for a fair comparison to date. In addition, most of the reported systems are hard to employ for researchers or practitioners who want to convert Chinese text into pinyin at their convenience. Motivated by these, in this work, we introduce a new benchmark dataset that consists of 99,000+ sentences for Chinese polyphone disambiguation. We train a simple Bi-LSTM model on it and find that it outperforms other pre-existing G2P systems and slightly underperforms pre-trained Chinese BERT. Finally, we package our project and share it on PyPi.

Index Terms: Grapheme-to-phoneme conversion, Chinese polyphone disambiguation, text-to-speech, Python package

1. Introduction

Chinese grapheme to phoneme (G2P) conversion is a task that changes Chinese text into pinyin, an official Romanization system of Chinese. It is considered essential in Chinese Text-to-Speech (TTS) systems as unlike English alphabets, Chinese characters represent the meanings, not the sounds. A major challenge in Chinese G2P conversion is how to disambiguate the pronunciation of polyphones—characters having more than one pronunciation. In the example below, the first 的 is pronounced *de*, which means the possessive particle "of", while the second one is pronounced *dì*, which denotes the "purpose".

- *Input:* 今天来的目的是什么?
  Translation: What is the purpose of coming today?
  Output: jīn tiān lái **de** mù **dì** shì shén me ?

There have been many academic efforts to tackle this problem [1, 2, 3, 4, 5, 6, 7]. However, we find there exist two main problems with them. First, there are no standard benchmark datasets for Chinese polyphone disambiguation. As shown in Table 1, most past works collect copyright data from the Internet, and annotate themselves. Due to the lack of a public benchmark dataset, they report results on different datasets. This makes it hard to compare different models. Second, all of the reports in Table 1 do not lead to the release of source code or packages where researchers or practitioners can convert Chinese text into pinyin at their convenience.

[*]Equal contribution.

Table 1: *Summary of major past works. Note that most of them source the data from the Internet news articles so it is impossible to access. [4] use a commercial company's internal dataset which is not freely available.*

Work	Year	Data Source	License	Code
[5]	2001	Ren Ming Daily	copyright	N/A
[6]	2002	People Daily	copyright	N/A
[7]	2008	Sinica and China Times	copyright	N/A
[8]	2009	People's Daily	copyright	N/A
[3]	2010	People's Daily	copyright	N/A
[2]	2011	People's Daily	copyright	N/A
[9]	2004	People Daily	copyright	N/A
[1]	2016	the Internet	copyright	N/A
[4]	2019	Data Baker Ltd	copyright	N/A

Motivated by these, we construct and release a new Chinese polyphone dataset and a Chinese G2P library using it. Our contribution is threefold:

- We create a new Chinese polyphonic character dataset, which we call **C**hinese **P**olyphones with **P**inyin (CPP). It is freely available via our GitHub repository[1].
- With the CPP dataset, we train simple neural network models for the Chinese polyphonic character to pinyin task. We find that our best model outperforms other existing G2P systems.
- We build a user-friendly Chinese G2P Python library based on one of our models, and share it on PyPi.

2. Related Work

G2P There are several works for Chinese polyphone disambiguation. They can be categorized into the traditional rule-based approach [5, 6, 7] and the data driven approach [1, 2, 3, 4, 10]. The rule-based approach chooses the pronunciation of the polyphonic character based on predefined complex rules along with a dictionary. However, this requires a substantial amount of linguistic knowledge. The data driven approach, by contrast, adopts statistical methods such as Decision Tree [3] or Maximum Entropy Model [2, 10]. Recently [1, 4] use bidirectional Long Short-Term Memory (LSTM) [11] to extract diverse features on the character, word, and sentence level. However, as they depend on external tools such as a word segmenter and a Part-Of-Speech tagger which are not perfect, they are inherently prone to the cascading errors. Recently, some works [12, 13] consider graphemes to phonemes as sequence transduction and leverage encoder-decoder architecture to generate multilingual phonemes.

[1]https://github.com/kakaobrain/g2pM

http://dx.doi.org/10.21437/Interspeech.2020-1094

Table 2: *Percentage of Chinese polyphones in Wikipedia. A monophone is a character that has a single pronunciation.*

	Total	Monophones	Polyphones
# unique char.	17,720	16,929 (95.60%)	762 (4.30%)
# characters	363M	296M (81.51%)	67M (18.49%)

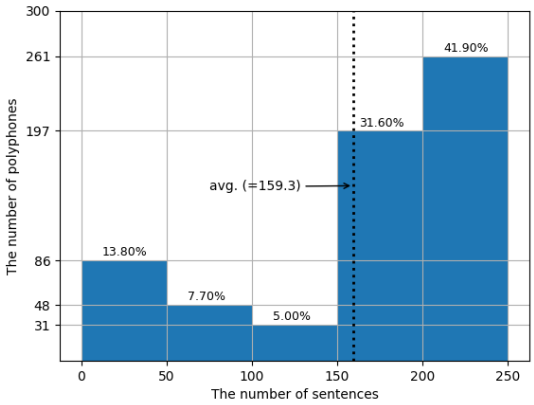

Figure 1: *The number of sentences for each polyphonic characters in CPP dataset. On average, a polyphonic character has about 159 sentences.*

Benchmark Dataset To the best of out knowledge, there are no standard benchmark datasets for Chinese polyphone disambiguation. In contrast, there are several public benchmark datasets for English G2P such as CMUDict, Pronlex and NetTalk.

3. Chinese Characters and Polyphones

We explore what percentage of Chinese characters are polyphones to gauge how important the polyphone disambiguation task is in Chinese.

We download the latest Chinese wiki dump file[2] and extract plain Chinese text with WikiExtractor [3]. All characters including white spaces except Chinese characters are removed. As shown in Table 2, the remaining text consists of 17,720 unique characters, or 363M character instances. Meanwhile, we collect the list of polyphones from the open-source dictionary, CC-CEDICT[4]. According to it, 762 out of the 17,720 characters, which account for only 4.30%, turn out to be polyphones. However, they occur 67M times in the text, accounting for as much as 18.49%. This indicates that disambiguating polyphones is a serious problem in Chinese. The most frequent 100 polyphones and their frequencies are provided in Appendix for reference.

4. The CPP (Chinese Polyphones with Pinyin) Dataset

In this section, we introduce the CPP dataset—a new Chinese polyphonic character dataset for the polyphone disambiguation task.

[2] https://dumps.wikimedia.org/zhwiki/latest/zhwiki-latest-pages-articles.xml.bz2

[3] https://github.com/attardi/wikiextractor

[4] https://cc-cedict.org/wiki/

Table 3: *Basic statistics of CPP dataset*

	Total	Train	Dev.	Test
# sentences	99,264	79,117	9,893	10,254
# characters per sent	31.30	31.29	31.24	31.43
# polyphones	623	623	623	623

Table 4: *The number of polyphones and sentences in the CPP dataset by the number of possible pronunciations*

# Pronunciations	# Polyphones	# Sentences
Total	623 (100%)	99,264 (100%)
2	553 (88.8%)	87,584 (88.2%)
3	60 (9.6%)	10,162 (10.2%)
4-5	10 (1.6%)	1,518 (1.6%)

4.1. Data Collection

We split the aforementioned Chinese text in Wikipedia into sentences. If a sentence contains any traditional Chinese characters, it is filtered out. Also, sentences whose length is more than 50 characters or less than 5 characters are excluded. Then, we leave only the sentences having at least one polyphonic character. A special symbol _ (U+2581) is added to the left and right of a polyphonic character randomly chosen in a sentence to mark the target polyphone. Finally, in order to balance the number of samples across the polyphones, we clip the minimum and maximum number of sentences for any polyphones to 10 and 250, respectively.

4.2. Human Annotation

We have two native Chinese speakers annotate the target polyphonic character in each sentence with appropriate pinyin. To make it easier, we provide them with a set of possible pronunciations extracted from CC-CEDICT for the polyphonic character. Next, we ask the annotators to choose the correct one among those candidates. It is worth noting that we do not split the data in half for assignment. Instead, we assign both of the annotators the same entire sentences. Then, we compare each of their annotation results, and discard the sentence if they do not agree.

4.3. Data Split

As a result, 99,264 sentences, each of which includes a target polyphone with the correct pinyin annotation, remain. Subsequently, we group them by polyphones. For each group, we shuffle and split the sentences into training, development, and test sets at the ratio of 8:1:1. See Table 3 for details. An example whose target polyphone is 角 and its correct pinyin is 'jiao3' is shown below, where the digit denotes Chinese phonetic tone.

- *Sentence:* 即闽粤贛三_角_地带。
  Label: jiao3

4.4. Statistics

Figure 1 shows how many sentence samples each of the polyphones in the CPP dataset has. 73.5% of polyphones (458 of 623) have 150-250 samples, while only 13.8%, i.e., 86 polyphones have less than 50 samples. Obviously, this comes from the differences in the frequency of polyphones.

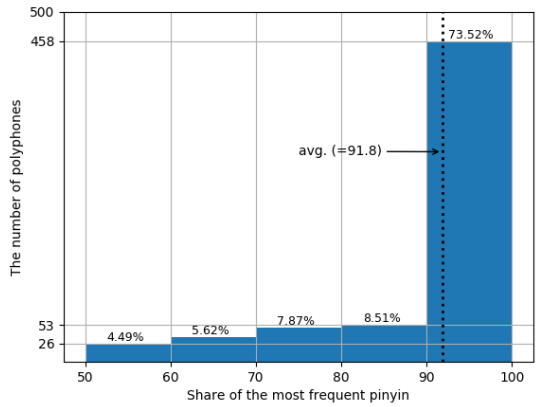

Figure 2: *The number of polyphones by the share of the most frequent pinyin for each polyphonic character.*

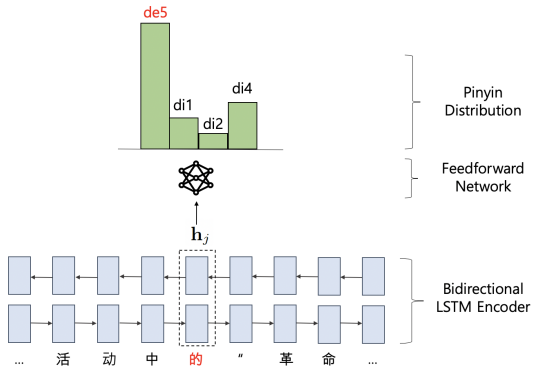

Figure 3: *Conceptual illustration of our models. A sequence of dense character embeddings are encoded with bidirectional LSTMs and the hidden state of the polyphonic character (red-colored) is fed to the feedforward network. It outputs the distribution of the pinyin candidates and finally the most probable one, "de5" here, is decided as the pronunciation of the character* 的.

We also present how many pronunciations the polyphones in the dataset can have in Table 4. Among 623 polyphones in the dataset, 553 (88.8%) have two possible pronunciations. There are 60 (9.6%) polyphones in the dataset that can have three pronunciations, and the rest 10 can have up to five pronunciations. All things being equal, we suppose the more pronunciations a polyphone can have, the more challenging it is for a predictor to disambiguate its correct pronunciation.

Finally, we explore how dominant the most frequent pronunciation in each polyphone is. As shown prominently in Figure 2, 73.52% of polyphones are associated with a single prevalent pronunciation that accounts for more than 90% of all samples. This implies that majority vote—picking up the pronunciation that occurs most frequently in the training set—would be a strong baseline. However, it is also important to remember there are still many that are less inclined to a dominant pronunciation so majority vote is less effective.

5. Method

We consider Chinese polyphone disambiguation as a classification problem and train a function, parameterized by neural networks, which maps a polyphonic character to its pronunciation.

We do not use any external language processing tools such as word segmenter, entity recognizer, or Part-Of-Speech tagger. Instead, we take as input a sequence of characters and train the network in the end-to-end manner.

5.1. Embedding

Let $\mathbf{x} = (x_1, \ldots, x_T) \in \mathbb{R}^T$ a sequence of characters, which represent a sentence. We map each character x_t to the dense embedding vector $\mathbf{e}_t \in \mathbb{R}^d$ with a randomly initialized lookup matrix $\mathbf{E} \in \mathbb{R}^{\mathcal{V} \times d}$, where $\mathcal{V}$ is the number of all characters and d is the dimension of the embedding vectors. We denote a sequence of character embedding vectors by $\mathbf{e} = (\mathbf{e}_1, \ldots, \mathbf{e}_T)$.

5.2. Bidirectional LSTM Encoder

The bidirectional Long Short-Term Memory (Bi-LSTM) [11] network is used to encode the contextual information of the polyphonic character. At any time step t, the representation $\mathbf{h}_t$ is the concatenation of the forward hidden state ($\overrightarrow{\mathbf{h}}_t$) and the backward hidden state ($\overleftarrow{\mathbf{h}}_t$).

$$\overrightarrow{\mathbf{h}}_t = \overrightarrow{\text{LSTM}}(\mathbf{e_t}, \overrightarrow{\mathbf{h}}_{t-1})$$
$$\overleftarrow{\mathbf{h}}_t = \overleftarrow{\text{LSTM}}(\mathbf{e}_t, \overleftarrow{\mathbf{h}}_{t-1})$$
$$\mathbf{h}_t = concat(\overrightarrow{\mathbf{h}}_t, \overleftarrow{\mathbf{h}}_t)$$

5.3. Fully Connected Layers

We use two fully connected layers to transform the encoded information into the classification label. Let j the position index of the polyphonic character in the sentence. The concatenated hidden state $\mathbf{h}_j$ (dotted line in Figure 3) is fed into the two-layered feedforward network followed by the softmax function, yielding the pinyin probability distribution $\hat{\mathbf{y}}$ over all possible pinyin classes as follows:

$$\hat{\mathbf{y}} = (\hat{y}_1, \ldots, \hat{y}_c) = \text{softmax}(g_2(\varphi(g_1(\mathbf{h}_j)))) \quad (1)$$

where g_1 and g_2 are fully connected layers, and φ is a non-linear activation function such as ReLU [14], and c is the number of possible pinyin classes.

5.4. Loss Function

Let $\mathbf{y} = (y_1, \ldots, y_c) \in \mathbb{R}^c$ be a one-hot vector of a true label. We use cross-entropy as a loss function for training. In other words, we minimize the negative log-likelihood to find the optimal parameters θ, which we denote as $\hat{\theta}$.

$$\mathcal{L}(\theta) = -\sum_{j=1}^{c} y_j \log(\hat{y_j}) \quad (2)$$

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\theta) \quad (3)$$

6. Experiments

6.1. Training

We randomly initialize the character embedding matrix and fix its dimension to 64. To find the optimal hyperparameter val-

Table 5: *Development set accuracy of varying models by the hidden size (denoted as **H**) and the number of LSTM layers (denoted as **L**). Note that the model in bold face is the best one.*

H \ L	1	2	3
16	94.34 ± 0.17	92.99 ± 0.11	85.30 ± 4.50
32	96.64 ± 0.04	96.01 ± 0.14	95.75 ± 0.14
64	$\mathbf{97.15 \pm 0.09}$	97.09 ± 0.05	96.58 ± 0.07

Table 6: *Test set accuracy of Chinese g2p systems*

System	Test Accuracy
Majority vote	92.08
xpinyin (0.5.6)	78.56
pypinyin (0.36.0)	86.13
g2pC (0.9.9.3)	84.45
Chinese BERT	**97.85**
Ours	97.31

ues, we vary the hidden size[5] in (16, 32, 64) and the number of layers in the Bi-LSTM encoder in (1, 2, 3). The dimension of the last two fully connected layers is set to 64, and ReLU [14] is used as the activation function.We train all the models with Adam optimizer [15] and batch size 32 for 20 epochs. All the experiments are run five times with different random seeds.

6.2. Evaluation

Hyperparameter Search Table 5 summarizes the development set accuracy of various models according to the hidden size and the number of layers in the Bi-LSTM encoder. We observe that the bigger the hidden size is, the higher the accuracy is, as expected. However, we get the better result when we use the fewer number of layers. The model of a single layer with 64 hidden units shows the best performance.

Baseline & other systems As we mentioned earlier, we take so-called "majority vote" as a baseline. It decides the pronunciation of a polyphonic character by simply choosing the most frequent one in the training set. For example, 咯 can be pronounced *luò*, *gē*, and *lo*, and their frequencies in the CPP training set are 63, 51, and 2, respectively. At test time, the majority vote system always picks up *luò* for 咯, irrespective of the context.

We also compare our model with three open-source Chinese G2P libraries: xpyinin[6], pyinin[7], and g2pC[8]. xpinyin and pypinyin are based on rules, while g2pC uses Conditional Random Fields (CRFs)[16] for polyphone disambiguation. All of them are easily accessible through PyPi.

Finally, we test the pretrained Chinese BERT model [17]. We take a finetuning approach; we attach a fully connected layer to the BERT network and feed the hidden state of the polyphonic character to it. We do not freeze any weights.

Results Our model slightly underperforms Chinese BERT and outperforms all the other systems by large margin. As shown in Table 6, ours reaches 97.31% accuracy on the test set, which is 4.33% point higher than the majority vote and 0.54 lower than Chinese BERT. That our simple neural model shows comparable performance to the heavy BERT model, which has more than 102M parameters, tells us two things. One is that our model is simple but powerful enough. Another is that it is not too simple for the naïve majority vote to beat.

[5]The hidden size in this context refers to the size *after* the concatenation of the forward and backward hidden states.

[6]https://github.com/lxneng/xpinyin

[7]https://github.com/mozillazg/python-pinyin

[8]https://github.com/Kyubyong/g2pC

Table 7: *Breakdown of g2pM. × denotes the number of layers.*

Layer	Size
Embedding	64
LSTM ×1	64
Fully Connected ×2	64
Total # parameters	477,228
Model size	1.7MB
Package size	2.1MB

7. g2pM: a Grapheme-to-Phoneme Conversion Library for Mandarin Chinese

We develop a simple Chinese G2P library in Python, dubbed *g2pM*, using our best Bi-LSTM model. The package provides an easy-to-use interface in which users can convert any Chinese sentence into a list of the corresponding pinyin. We share it on PyPi at `https://pypi.org/project/g2pM/`.

7.1. Packaging

We implement g2pM purely in Python. In order to minimize the number of external libraries that must be pre-installed, we first re-write our Pytorch inference code in NumPy [18]. Our best model is 1.7MB in size, and the package size is a little bigger, 2.1MB, as it includes some contents of CC-CEDICT. Details are shown in Table 7. g2pM works like the following. Given a Chinese text, g2pM checks every character if it is a polyphonic character. If so, the neural network model returns its predicted pronunciation. Otherwise, the pronunciation of the (monophonic) character is retrieved from the dictionary contained in the package.

```
>>> from g2pM import G2PM
>>> model = G2PM()
>>> sentence = "因为脑部手术需剃光头。"
>>> model(sentence)
['yin1', 'wei4', 'nao3', 'bu4', 'shou3'
,'shu4', 'xu1', 'ti4', 'guang1', 'tou2', '。']
```

Figure 4: *Usage example of g2pM.*

7.2. Usage

g2pM provides simple APIs for operation. With a few lines of code, users can convert any Chinese text into a sequence of pinyin. An example is available in Figure 4. More details are on the Github repository.

8. Conclusion

We proposed a new benchmark dataset for Chinese polyphone disambiguation, which is freely and publicly available. We trained simple deep learning models, and created a Python package with one of them. We hope our dataset and library will be helpful for researchers and practitioners.

9. References

[1] C. Shan, L. Xie, and K. Yao, "A bi-directional lstm approach for polyphone disambiguation in mandarin chinese," *2016 10th International Symposium on Chinese Spoken Language Processing (ISCSLP)*, 2016.

[2] F. Z. Liu and Y. Zhou, "Polyphone disambiguation based on maximum entropy model in mandarin grapheme-to-phoneme conversion," *Key Engineering Materials*, 2011.

[3] J. Liu, W. Qu, X. Tang, Y. Zhang, and Y. Sun, "Polyphonic word disambiguation with machine learning approaches," in *2010 Fourth International Conference on Genetic and Evolutionary Computing (ICGEC)*, 2010.

[4] Z. Cai, Y. Yang, C. Zhang, X. Qin, and M. Li, "Polyphone disambiguation for mandarin chinese using conditional neural network with multi-level embedding features," in *INTERSPEECH*, 2019.

[5] Z. Hong, Y. Jiangsheng, Z. Weidong, and Y. Shiwen, "Disambiguation of chinese polyphonic characters," in *The First International Workshop on MultiMedia Annotation (MMA2001)*, 2001.

[6] Z. Zirong, C. Min, and C. Eric, "An efficient way to learn rules for grapheme-to-phoneme conversion in chinese," in *2002 International Symposium on Chinese Spoken Language Processing (ISCSLP)*, 2002.

[7] F.-L. Huang, "Disambiguating effectively chinese polyphonic ambiguity based on unify approach," in *2008 International Conference on Machine Learning and Cybernetics (ICMLC)*, 2008.

[8] L. Yi, L. Jian, H. Jie, and Z. Xiong, "Improved grapheme-to-phoneme conversion for mandarin tts," *Tsinghua Science and Technology*, no. 5, 2009.

[9] H. Dong, J. Tao, and B. Xu, "Grapheme-to-phoneme conversion in chinese tts system," *2004 International Symposium on Chinese Spoken Language Processing*, 2004.

[10] X. Mao, Y. Dong, J. Han, D. Huang, and H. Wang, "Inequality maximum entropy classifier with character features for polyphone disambiguation in mandarin tts systems," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*. IEEE, 2007.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 1997.

[12] M. Yu, H. D. Nguyen, A. Sokolov, J. Lepird, K. M. Sathyendra, S. Choudhary, A. Mouchtaris, and S. Kunzmann, "Multilingual grapheme-to-phoneme conversion with byte representation," 2020.

[13] A. Sokolov, T. Rohlin, and A. Rastrow, "Neural Machine Translation for Multilingual Grapheme-to-Phoneme Conversion," in *Proc. Interspeech 2019*, 2019.

[14] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[16] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[17] Y. Cui, W. Che, T. Liu, B. Qin, Z. Yang, S. Wang, and G. Hu, "Pre-training with whole word masking for chinese bert," *arXiv preprint arXiv:1906.08101*, 2019.

[18] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, 2020.