Neural Input Method Engine

Jingchen Hu and **Qingda Wen** and **Jianyang Zhang**

{jingchen, qw9, jianyz}@cs.washington.edu

Abstract

Input Method Engine (IME) is a program that facilitates the input of non-english languages into digital devices. This work improves upon traditional n-gram based Chinese Pinyin IMEs by incorporating previous context and using a Seq2Seq neural network model with end-to-end training. Our model simplifies the NLP pipeline, while maintaining some tolerance for Pinyin abbreviations and typos. Our evaluation shows that it significantly outperforms the baseline bigram model in terms of prediction accuracies. We also built a Chrome extension frontend to help users type Chinese in any web pages.

1 Introduction

1.1 background

Input Method Engine (IME) is a program that facilitates the input of language symbols, especially for languages where there's no direct character mapping from the keyboard to the character set, such as Chinese. Since such IMEs are obligatory for many languages, improvement in their performance could potentially improve the daily productivity for users.

Pinyin Input Methods are by far the most popular Chinese IMEs. Pinyin is the standard pronunciation transcription system of Mandarin characters, understood by a vast majority of Chinese speakers [1]. Each character (and thus its Pinyin) corresponds to one syllable and/or morpheme. There are around 410 valid pinyin tokens in the language, but literate individuals normally use between 3,000 and 4,000 characters (Norman, 1988). Therefore the task of mapping Pinyin tokens to Chinese characters is often ambiguous and highly context dependent. Some characters can also have several pronunciations. Below are some examples of pinyin-character mapping (the numbers represent pronunciation tones):

ha=铪,鉿,蛤,畣,哈,紦,蝦,虾
落=luo4,la4,lao4

A typical Chinese Pinyin IME works interactively with the user, where a collection of phrase suggestions are listed according to the current input pinyin string (the tones are usually dropped for convenience).

The goal of the IME would be to rank the most likely prediction at the top of the suggestion list, since the top items in the suggestions requires the least amount of keystrokes. If an item is ranked low, the user would need to hit next page several times on a computer, or scroll all the way down on a smart phone. Thus more accurate predictions means less keystrokes and faster typing.

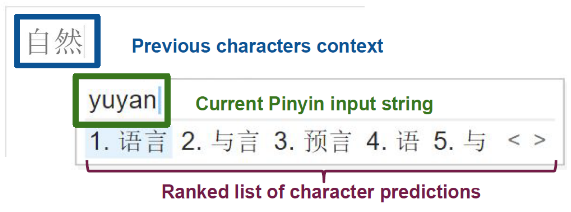

自然
Previous characters context
yuyan
Current Pinyin input string
1. 语言 2. 与言 3. 预言 4. 语 5. 与 < >
Ranked list of character predictions

Figure 1: Chinese Pinyin IME Interface

[1] In this work we limited the scope to only considering Simplified Mandarin, but the same reasoning holds for traditional Chinese and other dialects or languages as well.

1.2 Previous work

Traditionally IME implementations are usually based on n-gram models, framed as a decoding problem in HMM, and variants of the Viterbi Algorithm are often used. For instance, Chen and Lee used a statistical segmentation and a trigram model to convert Pinyin tokens to Chinese characters (Chen and Lee, 2000). They also proposed a Hidden Markov Model based approach for single Pinyin spelling correction. To correct multi-character typo errors, Zheng, Li and Sun proposed a way by keeping the top-k correct spellings that are most similar to the sequence of spellings that has typos and converting those correctly spelled Pinyin tokens to Chinese characters (Zheng et al., 2011). However, this approach depends on the assumption that input Pinyins are correctly segmented and a large knowledge table is required for a limited contextual window.

We realized that n-gram models, despite their simplicity, fail to incorporate enough previous long-term context, which may be very useful to disambiguate potential decoding candidates and make more accurate predictions. Furthermore, such models often require a multi-step NLP pipeline, where upstream errors can be easily propagated down. For instance, pinyin string segmentation is a necessary step for filtering bigram prediction candidates, but the segmentation task itself is inherently ambiguous and can't be solve perfectly with current tools.

Recent researches show that Neural Network models using the seq2seq architecture with attention mechanism significantly outperform previous state of the art in various NLP tasks such as Machine Translation and dialogue generation (Cho et al., 2014; Bahdanau et al., 2014). In this work we explored the possibility of applying the seq2seq model to the IME problem, and incorporating the previous context, in order to improve the performance of Chinese Pinyin IME.

1.3 Contribution

The major contribution of this work is that we framed the training and inference scheme of an IME on top of the seq2seq model, and applied techniques to make the learned model more usable. Our model predicts users' intended phrases based on context and user input, while maintaining some level of tolerance for pinyin abbreviations and typos.

At a high level, we used tuples extracted from parallel corpora as training data, added noise to augment the dataset and emulate user input, used encoder-decoder model with beam search to generate the ranked list of predictions per output length, and finally merged the results of different lengths as the suggestion list. The core of our model is the encoder-decoder model, where the encoder takes in the previous context as well as the input pinyin tokens.

Preliminary evaluation shows that our model significant outperforms the baseline bigram model. Multiple variants of the encoder-decoder architecture were attempted, and most of them achieved comparable performance.

2 Data

2.1 Data sources

We used a mixture of four different datasets to train our models. The Lancaster Corpus of Mandarin Chinese (LCMC) is a parallel corpus of Mandarin texts in diverse genres along with their annotated Pinyin(McEnery and Xiao, 2004); NUS Short Messages Corpus consists of about 10,000 Chinese SMS messages (Chen and Kan, 2013); Open Weibo contains Sina Weibo posts, which are Twitter-like microblog posts in Chinese, as well as a subset of Wikipedia article abstracts (snapshotted at 2016-04) (Auer et al., 2007).

Since the latter three datasets are not pinyin-annotated, we added the corresponding pinyin for each character and built parallel corpora using a dictionary-based character-pinyin conversion tool as part of the HanLP package (Hankcs, 2017). Although the conversion is not perfectly accurate (greater than 90% accuracy), the mis-labeled Pinyins are likely to correspond to the characters that are easily mispronounced in real life, and thus can be viewed as a type of noise injection similar to the ones described in 2.3.

Moreover, we used the pinyin and character dictionary in the HanLP package to implement tasks such as pinyin segmentation.

We trained all of our models initially on the combination of LCMC, Weibo and SMS data. Due to the time constraints, we only trained the RNN seq2seq model on the wikipedia dataset, and the set of parameters we used could be far from optimal.

2.2 Data extraction

2.2.1 Tuple extraction, sliding window

We extracted data entry tuples from the original datasets in the form of (Context, Pinyin, Label), where Context is a window of characters previous to the current "cursor". Pinyin is the current input pinyin string, and Label is the ground truth output phrase. Such triplets are extracted from the original parallel corpus in a sliding window fashion, where we move the "cursor" position along each sentence and then vary the input lengths.

In our experiments we used a context window size of 10, and a max Pinyin input window of 5. And we used character-level vocabulary for most of our models.

2.2.2 Word boundary-aligned sliding window

Initially, we allowed the cursor to be at any positions in a sentence. This results in many word fragments in the training set, and thus lots of predictions consist of combinations of common characters that don't make sense when put together.

As an improvement, we added the restriction that each "cursor position" in the sliding window must align with actual word boundaries. The word boundaries were precomputed using the HanLP package's segmentation tool. The improved sliding window both reduced the size of the generated dataset, and increased the data quality and naturalness, decreasing the validation cross entropy loss by around 50%.

2.3 Noise Injection

For an IME to be convenient to use, it needs to support various kinds of abbreviations and fuzzy matches in input handling. To support these, we added noise to the extracted tuples to emulate actual user input. These noisy mutations on a tuple (Context_i, Pinyin_i, Label_i) can be thought as a pipeline, where in each step we randomly fork a copy (Context_i, Pinyin_i', Label_i) and add abbreviations and/or typos to $\text{pinyin}_{i'}$. Note that some mispronunciations were implicitly injected via the pinyin labeling process in 2.1.

The intuition here is that although recovering from abbreviations and typos are hard, the reverse is much simpler. If we push the fuzzy match logic all the way down to the dataset, the seq2seq model will learn the mapping end-to-end. In our evaluation, we will show the effects of different dataset noise levels.

2.3.1 Abbreviation generation

We have a probability of 0.5 to generate a noisy copy of original pinyin tokens. If the program decides to do so, then each valid pinyin tokens in Pinyin_i' has a probability of 0.8 to be replaced by its abbreviation as an unambiguous prefix and usually dropping the vowel.

2.3.2 Typo generation

Similarly for typos, given we are generating a noisy copy, each pinyin token has a probability of 0.1 to be mutated by one of letter transpose, addition and deletion, chosen uniformly.

2.4 Result Tuples

Table 1 shows the number of tuples we extracted from each corpus. The datasets grow larger as we add more noisy copies.

Finally the extracted tuples are split into training, development and test set at a ratio of 7:1:2. For efficiency, we only used 50k samples of the development set during training.

3 Methods

This section describes the basic models and algorithms we used for building the IME.

3.1 Bigram baseline

We used character level bigram model as our baseline. We calculated the bigram counts on the first three corpora combined (except Wiki abstract), using add-1 smoothing to handle unknown words. The top k results, filtered by the input pinyin tokens, are generated using beam search.

The Pinyin input by the user is a contiguous string containing only alphabetic letters. Therefore for

Dataset	Clean	With Abbreviation	With Abbreviation + Typo
LCMC	2 Million	3 Million	3.2 Million
SMS	522 K	759 K	811 K
Weibo	5.8 Million	8.5 Million	9.1 Million
Wikipedia	35.7 Million	52.4 Million	55.9 Million

Table 1: Number of extracted tuples

bigram we need to first segment the whole string into syllables (whole pinyin strings or their abbreviated prefixes). To do so, we implemented a simple heuristic that tries to minimize the number of syllables segmented, using a dynamic programming algorithm. In contrast, our Seq2Seq model takes the whole input string as part of the input without segmentation. Section 5.3.1 shows the error analysis for bigram with pinyin segmentation.

3.2 Seq2Seq with Attention

The core of our model is the seq2seq, or encoder-decoder architecture with attention mechanism, introduced in Sequence to Sequence Learning with Neural Networks and Neural Machine Translation by Jointly Learning to Align and Translate (Sutskever et al., 2014; Bahdanau et al., 2014). The RNN learns to capture long-term memory, while the attention learns to examine the most relevant encoder states at each decoding step. In our case, we feed the concatenation of context characters and pinyin input tokens [context; input] into the encoder, and use beam search with beam width 100 to extract the best predictions for each output length during decoding.

The best RNN seq2seq model we had uses a 256 dimension embedding, 3 layer bidirectional GRU cells and a one-layer perceptron based Attention mechanism (all 512 dimensions). The learning was performed using Adam Optimizer with initial learning rate of 0.0001. We clipped the result gradient at 5 maximum. For regularization, we added dropout keep rate of 0.6. We also tried several variants of the basic seq2seq model, presented In section 4, with similar configurations.

3.3 Merging predictions

For a input of L pinyin tokens, we used beam search to generate the top K predictions for length 1 to L, and merge the results as the final suggestion list. While merging the L lists, the predictions are first ranked by their length, and then by their probability scores.

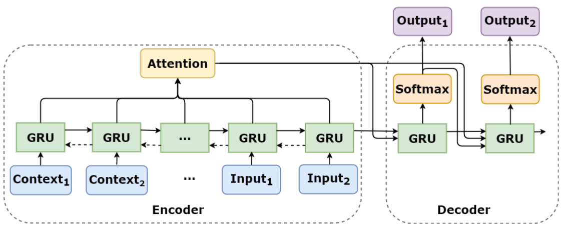


Figure 2: Seq2Seq with Attention

Moreover, we applied a cutoff probability value C_i for each prediction length, and only allow predictions with score greater than C_i to be returned in the final predictions list.

In order to choose C_i, we plotted and studied the distribution of the probability score of the truth prediction for each input length. Another potentially more flexible scheme would be delta based cutoff. That is, only keep prediction$_\mathrm{i}$ if $P(prediction_\mathrm{i}) - P(prediction_{\mathrm{i-1}}) > C_i$.

3.4 Unigram Fallback

It's important for an IME to be able to allow users to input any word they want, even it's a extremely rare one. To support this, we append after the prediction list all the candidate characters for the next pinyin input syllable, ranked by their smoothed unigram counts.

4 Model Variants

This section describes our attempts to adjust the model for our purpose and alternative seq2seq models we tried. The effects of these changes will be presented in the next section.

4.1 Word level vocabulary

We found that with character level vocabulary, the model's power of predicting less frequent whole words is limited, especially for longer named entities and idioms. One obvious solution is to use

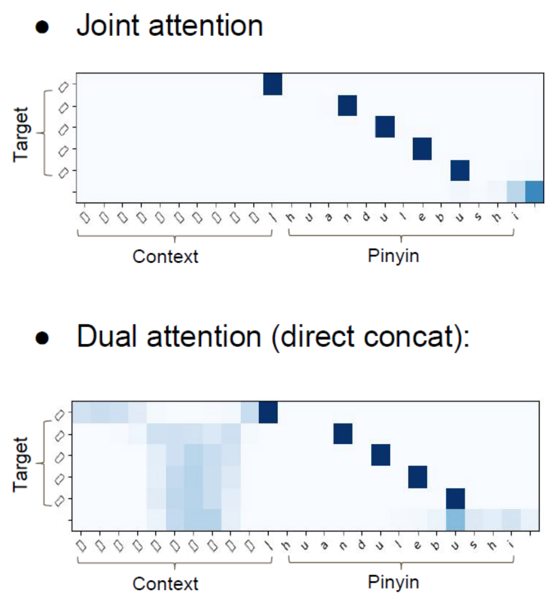


Figure 3: Attention scores

word level vocabulary in the decoder side, instead of one character at a time (the source vocabulary remains the same on the encoder side for easier prediction). However switching to word level target vocab does increase the vocabulary size by an order of magnitude (7k to 80k), making the training much harder.

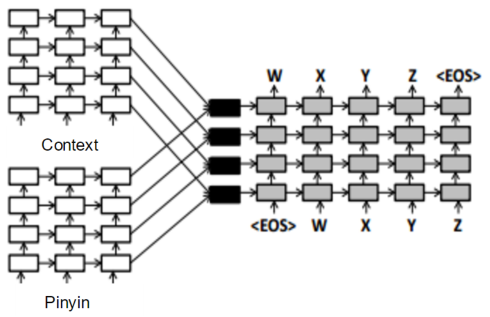
We didn't implement word-level n-gram models, but we believe they can easily outperform character-level models. With word-level n-gram, there won't be a training problem as in Seq2Seq, as the corpus will always be used efficiently to compute the n-gram count statistics.

4.2 Separate attention

When plotting the attention over the whole [context; pinyin] input, we found that little attention ended up placed in the context parts, as shown in Figure 3 (top). Each horizontal line in the heat map shows what encoder states get attended at each decoding step. This motivates us to try different ways to get the context attended as well.

We first tried using two separate softmax and concatenate the results. That is, we changed $softmax(att\_scores)$ into

$$[softmax(att\_scores_{\text{context}});$$
$$softmax(att\_scores_{\text{pinyin}})]$$



$$h = \tanh(W[h_c; h_p] + b)$$

Figure 4: Multiple encoders

and then renormalizes. On the resulting attention heat map, we see very soft and "homogenous" attention scores on the context part. (Each horizontal line represents the attention scores for one decoding step/output symbol).

4.3 Multiple encoders

Previously, on the encoder side, we feed in the concatenation of previous Chinese characters and currently pinyin tokens as the input to our encoder. However, since Chinese characters and pinyin are really different languages, we decide to try using two separate encoders for them. This idea is inspired by the paper Multi-Source Neural Translation (Zoph and Knight, 2016). As shown in the diagram below, our original input is split into two parts: previous Chinese character, and current pinyin tokens, each part has its own encoder (each is 3-layer and bidirectional). At the end, we concatenate the outputs of the two encoders, pass the concatenated result into a tanh function and feed the final output to the decoder. However, the new architecture caused a lot more numeric unstableness during training.

4.4 Non-RNN encoders

With the recent release of Facebook's CNN based NMT paper (Gehring et al., 2017), we turned our attention to non-RNN encoders. The end-to-end CNN training is too much work for us, and therefore we tried pooling and CNN in the encoder part only (the library we used already has a base implementation), as described in A Convolutional Encoder Model for Neural Machine Translation (Gehring et al., 2016). For both approaches, the context characters are front-padded to the same length and then concatenated with the pinyin input.

Pooling encoder is the baseline for non-RNN encoders, where the encoder layers simply average the embeddings of k consecutive words. To preserve the absolute position of each input, position embedding is calculated as a constant function, and added on top of the word embeddings to form the source embeddings. We used three layers with 256 dimensions, and kernel size 2.

The CNN encoder contains two stacked convolutional networks, one for computing the encoder output, and one for computing the conditional input that will be fed into the decoder later on. We followed the settings mentioned in the paper with 5 convolutional layers and kernel size 3, which results in an input field of length 11.

Both models achieved comparable cross entropy losses with the RNN approach, maintained faster training speed, but are a little bit more prone to overfitting.

4.5 Beam search results re-weighting

Pure log probability based beam search sometimes generated sequence that does not make very natural sense. To increase the naturalness of predictions, we incorporate an alternative scoring function into the beam search. In Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, a scoring function with length normalization and coverage penalty was used as a refinement in addition to the max log-probability based beam search (Wu et al., 2016). Since we want to improve the local naturalness of the predictions, we use a weighted sum of bigram score of each token and its log-probability. We then use the score to re-rank the prediction list as the final result. The scoring function works especially well for longer tokens.

5 Model Evaluation

5.1 Evaluation Metrics

For each output length $L \in \{1, 3, 5\}$, we calculated the frequency for the desired phrase to be in the top one and top ten of the prediction list (before merging results from different lengths). All of the following models are trained on the combined dataset without wiki data, with abbreviations but not typo added.

5.2 Evaluation Results

- Seq2Seq
  - The standard Seq2Seq model, with context added (RNN), significantly outperforms the bigram baseline. Also, all seq2seq based models we tried have comparable performances.
  - Only "RNN (wiki)" was trained on the larger wiki dataset, and we can see the accuracies improved about 5% for longer inputs.
- Re-weighted Attention on RNN encoder (Dual-Att)
  - We are happy to see that the improved attention mechanism does improve the accuracies a little bit over the standard attentive RNN seq2seq model, especially for longer inputs.
- CNN and Pooling encoders
  - CNN and Pooling encoders performed only slightly worse than the RNN encoders, but they both saw a roughly 2 times improvement in training speed, since the encoder can process all the input tokens without the temporal dependency. One can imagine that a full CNN approach will be even faster.
- Word level vocabulary
  - It surprised us that the accuracies for word level vocabulary are actually worse than those for character level vocabulary, or even bigram.
  - After examining the prediction samples, we found out that the main cause of the poor performance is that often words containing wrong number of characters get predicted. Since tokens in the word-level vocabulary have no notion of their internal character compositions, we think it's probably way harder for the model to learn the mapping from long words to long sequences of pinyin

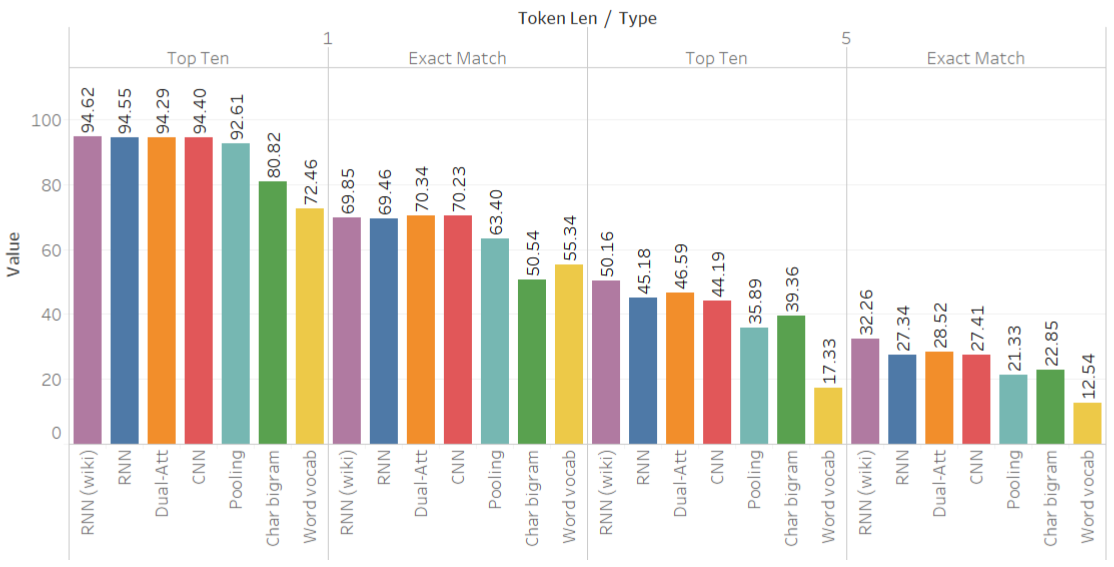


Figure 5: Evaluation Results

symbols, especially given the huge size of words in the vocabulary. The misaligned output can potentially make the IME tool unusable.

5.3 Analysis

5.3.1 Segmentation Bottleneck for Bigram

Since the bigram model heavily depends on the segmentation results, the segmentation error poses a bottleneck for the prediction performance, especially for long input sequences.

We set up an experiment to evaluate the segmentation accuracy on Pinyin strings. The accuracy is 88.96 on the LCMC data. Most errors happen when one Pinyin string has different ways to be segmented. For example, 'Xian' can be both segmented to 'xi' 'an' and 'xian'. "langan" can be segmented into 'lan' and 'gan' or 'lang' and 'an'. These errors are less of an issue for the Seq2Seq model, since it learns the alignment jointly, as can be seen from the attention heatmap.

5.3.2 Data Noise Level Comparison

The chart above shows the prediction accuracies for the basic RNN seq2seq model, when trained and tested on datasets with different level of noise added respectively. As we expected, the more noise added, the lower the accuracies are. However, since a little less than 50% of the abbreviation dataset actually contains abbreviations, yet

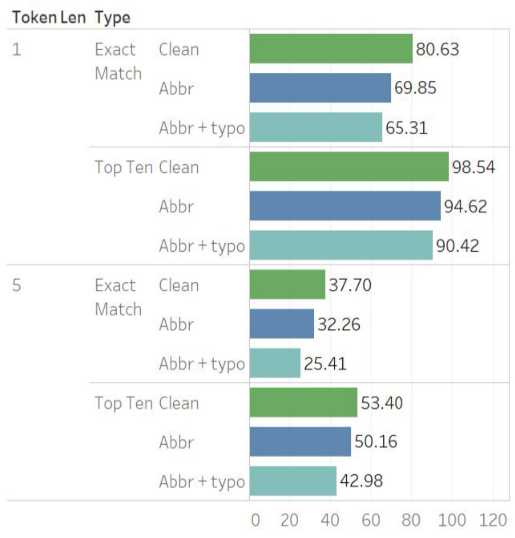


Figure 6: Data Noise Level Comparison

the accuracy only decrease a little, we can see that our model actually learns the mapping from abbreviated Pinyins. This is reflected when we tested out our IME tool. Abbreviation works well especially when the last Pinyin token is abbreviated.

For typo, we are a little conservative when choosing noise parameters, since adding too many typos to the training set might disturb the normal case. We hoped that it would achieve better performance if the parameters could be better tuned.

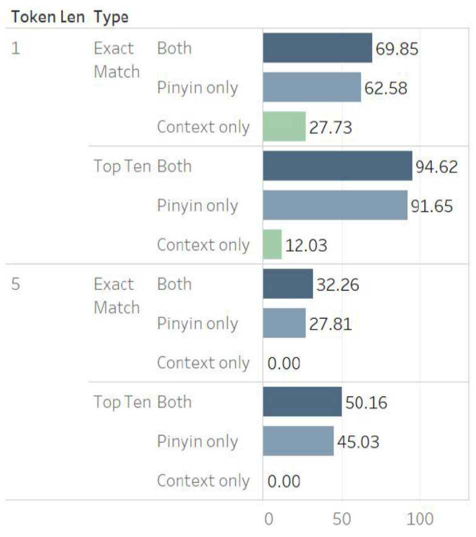


Figure 7: Effectiveness of Context

5.3.3 Effectiveness of Context

To further understand the effectiveness of previous context, we ran experiment with pinyin only and context only, and compared it with the original model:

To our surprise, context only is a really poor predictor for the following characters, while pinyin only already gives pretty good predictions. However, context does help non-trivially in increasing the accuracies, when combined with the pinyin input. It could be the case that the context provides hints to disambiguate subtle homophones in the prediction. Pinyin worked well also because it has a very small vocabulary compared to all the possible characters, which are much easier to train given our time and data resources.

6 Software Implementation

In this section we will describe more details on the IME software implementations.

6.1 Chrome Extension and ML server

In the frontend, we built a Chrome extension to allow the users to input in text boxes (inputs and textareas) in web pages. The tool works by detecting keystrokes, buffering the current pinyin input, and querying the machine learning backend as the user types.

The UI follows the design of popular existing Chinese IME softwares, as a widget floating along as the user types. The tool allows various user actions. For instance, the user can use number keys to select an item, space key to select the top prediction, and "enter" to input the buffered symbols as-is, and other keys to navigate through the paginations. Non-pinyin symbols can be input directly. It also support partial selection and adjustment, where the user can select the first few characters of a long input, and then the next few, and make adjustment as needed. Such pre-selection mechanism is implemented using a history stack, which is examined and popped when the user hits backspace.

In the backend, our server exposed APIs for prediction and segmentation (for bigram). For each prediction query, our inference handler will start an inference task and returns the suggestion list as Json.

6.2 The tf-seq2seq Library

We used Google's newly released Seq2Seq library (Britz et al., 2017) for training and inference of our models. The library is built on top of Tensorflow, using its r1.1 API. The main advantage of the library is that it is highly modularized, providing an interface much easier to use than the original Tensorflow API. It contains a rich set of built-in components, such as the CNN encoder. It can even be used entirely as a black box, with configurations specified in yml files. On the other hand, it's wrapped in so many layers of abstractions that it could be hard to understand and customize. Also since it's just released in March, there's still lots of features not supported and issues to be addressed. For instance, by default it only supports batch inference. We had to really dig deep into the code base to add support for single string inference and our customized beam search algorithm. We posted our solution on-line and the community were finding it helpful.

6.3 Design trade-offs

When using the Chrome Extension, we saw that the network latency takes up a large portion of the time for the suggestions to load, which drastically slowed down the typing speed. Thus the client-server architecture may not be ideal, not to mention that dealing with the potential heavy work load on the servers when lots of users are typing at the same time would be very costly.

However, in order to employ a client-only architecture, the trained model (at least several hundred MB in size) would need to reside on the client machine, which could make the IME software too heavyweight compared to the popular IME softwares, which only needs to store the sufficient statistics for the vocabulary.

7 Conclusion

7.1 Summary

In this work the explored the possibility of using the Seq2Seq model for Chinese Pinyin IME and incorporating the previous context. We built our own dataset from parallel corpora, and added noise to emulate user input. We used encoder-decoder model with beam search to generate the ranked list of predictions per input length, and merged the results of different lengths.

The various models we tried achieved similar performance, and significant outperformed the baseline character-level bigram model. Our further analysis shows that although the previous context alone is not a great predictor of the exact following characters, it does help in making the pinyin based predictions more accurate. For instance, when playing around with our IME tool, we found that given a context ”My dad/mom is at home”, and ”ta” as the pinyin input (The pronouns ”he”, ”she”, ”it” share the same pinyin ”ta”), our model successfully gives ”he”/”she” as the top prediction respectively.

During training, we are surprised to find that our models converges even before we hit one epoch. This could be the case that the models learns the pinyin-to-character mapping pretty quickly. However there is definitely more space for parameter fine-tuning.

One important idea we employed in our model is that although some problems are hard, the reverse are easier. For instance, one pinyin may have many corresponding candidate characters, but one character normally has a much smaller number of pronunciations. Also, recovering from abbreviations and typos are harder than mutating well-formed pinyin tokens. Therefore sometimes we can just take a step back, encode these into the model, and let the neural network learn all the hard rules.

7.2 Future work

We are positive that our models can achieve better performance if given more time for parameter tuning and training on large corpus. Other than that, our IME implementation could potentially be extended in several directions as follows.

Firstly, it would be nice if the user input history could be collected and incorporated into the model. It can not only enable personalized predictions, but also close the loop as new training data of much better quality in terms of naturalness and user input behaviors. User privacy might be an issue, and the federated learning scheme proposed by Google might help (res, 2017).

Secondly, in this work we used a small windowed context. However, the document and paragraph-level topic information could also be used.

Finally, we believe that our framing of the IME is general enough for input symbols other than Pinyin and languages other than Chinese. It could be interesting to see how it could be extended to support numeric key as input (on mobile keyboard), and languages such as Korean and Japanese, where IMEs are also needed.

References

2017. Federated learning: Collaborative machine learning without centralized training data. https://research.googleblog.com/2017/04/federated-learning-collaborative.html.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. *The semantic web* pages 722–735.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* .

Denny Britz, Anna Goldie, Thang Luong, and Quoc Le. 2017. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906* .

Tao Chen and Min-Yen Kan. 2013. Creating a live, public short message service corpus: the nus sms corpus. *Language Resources and Evaluation* 47(2):299–335.

Zheng Chen and Kai-Fu Lee. 2000. A new statistical approach to chinese pinyin input. In *Proceedings of the 38th annual meeting on association for computational linguistics*. Association for Computational Linguistics, pages 241–247.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* .

Jonas Gehring, Michael Auli, David Grangier, and Yann N Dauphin. 2016. A convolutional encoder model for neural machine translation. *arXiv preprint arXiv:1611.02344* .

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122* .

Hankcs. 2017. hankcs/hanlp. https://github.com/hankcs/HanLP.

Tony McEnery and Richard Xiao. 2004. The lancaster corpus of mandarin chinese. http://www.lancaster.ac.uk/fass/projects/corpus/LCMC/.

Jerry Norman. 1988. *Chinese*. Cambridge University Press.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. pages 3104–3112.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* .

Yabin Zheng, Chen Li, and Maosong Sun. 2011. Chime: An efficient error-tolerant chinese pinyin input method. In *IJCAI*. volume 11, pages 2551–2556.

Barret Zoph and Kevin Knight. 2016. Multi-source neural translation. *arXiv preprint arXiv:1601.00710* .