Learning to Pronounce Chinese Without a Pronunciation Dictionary

Christopher Chu, Scot Fang, and Kevin Knight
DiDi Labs
4640 Admiralty Way
Marina del Rey, CA 90292
{chrischu,scotfang,kevinknight}@didiglobal.com

Abstract

We demonstrate a program that learns to pronounce Chinese text in Mandarin, without a pronunciation dictionary. From non-parallel streams of Chinese characters and Chinese pinyin syllables, it establishes a many-to-many mapping between characters and pronunciations. Using unsupervised methods, the program effectively deciphers writing into speech. Its token-level character-to-syllable accuracy is 89%, which significantly exceeds the 22% accuracy of prior work.

1 Unsupervised Text-to-Pronunciation

Many papers address the construction of automatic grapheme-to-phoneme systems using rules or supervised learning, e.g. (Berndt et al., 1987; Zhang et al., 2002; Xu et al., 2004; Bisani and Ney, 2008; Peters et al., 2017).

The task of *unsupervised* grapheme-to-phoneme conversion is introduced by Knight and Yamada (1999). Given two non-parallel streams:

- A corpus of written language (characters).
- A corpus of spoken language (sounds).

the goal is to build:

- A mapping table between the character domain and the sound domain.
- A proposed pronunciation of the written character sequences.

Motivated by archaeological decipherment, Knight and Yamada (1999) view character sequences as “enciphered” phoneme sequences. Their evaluation compares the proposed pronunciations with actual pronunciations. With a noisy-channel expectation-maximization method, they obtain 96% phoneme accuracy on Spanish, 99% on Japanese kana, but only 22% syllable accuracy on Mandarin Chinese.

In this paper, we re-visit the task of deciphering Chinese text into standard Mandarin pronunciations (Figure 1). We obtain an improved 89% syllable accuracy. We further explore exposing the internals of characters and syllables to the analyzer, as Chinese characters sharing written components often sound similar.

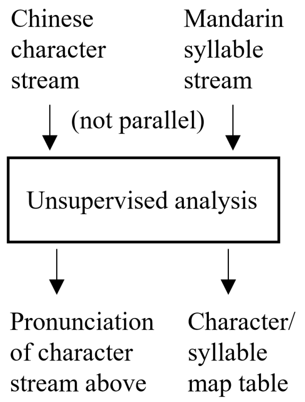

Figure 1: Learning to pronounce Chinese without a dictionary.

We find it compelling that pronunciation dictionaries are largely redundant with non-parallel text and speech corpora, even for writing systems as complex as Chinese. We also expect results may be of use in dealing with novel ways to write Chinese, such as Nüshu script (Zhang et al., 2016), with acoustic modeling of other Chinese languages and dialects, and with novel ways to phonetically encode and decode Chinese in online censorship applications (Zhang et al., 2014).

2 Chinese Writing

The most-popular modern Chinese writing system renders each spoken syllable token with a single character token (*hanzi*). There are over 400 syllable types in Mandarin[1] and several thousand character types. The mapping is many-to-many:

- Almost every syllable type can be written

[1]In this paper, we use standard *pinyin* syllable representation, and we refer strictly to Mandarin pronunciation.

with different characters (eg, *zhong* → {中, 重, 肿, ...}). The choice depends on context. For example, the word ***zhong**guo* ("China") is written 中国, but ***zhong**yao* ("important") is written 重要.

- Only a few character types are heteronyms, whose pronunciation depends on context and meaning (e.g., 了 → {*le*, *liao*}).

In addition, syllables have one of five tones, including the neutral tone (e.g., *dāng*, *dáng*, *dǎng*, *dàng*, *dang*). Tones increase pronunciation ambiguity. Most characters have single, unambiguous phonemic pronunciations (eg, 中 → *zhōng*), but a portion can be pronounced with different tones depending on context (eg, 当然 → ***dāng**rán*, but 适当 → *shì**dàng***).

While most Chinese words have two syllables, individual characters carry rough semantic meanings (eg, 中 = "middle", 重 = "weighty"). So it is no accident that the same character is used to write semantically-similar words:

- 中国 ("China = middle kingdom"), 中学 ("middle school"), 市中心 ("city center")
- 重要 ("important"), 重达 ("heavy"), 重点 ("focus")
- Similarly, the second syllable of "website" is spelled "site", not "sight".

Finally, many characters have loosely informative internal structure. For example, 鸦 can be analyzed into two *character components*: 牙 and 鸟.[2] Character components are sometimes a clue to pronunciation and/or meaning. For example:

- The character 鸦 ("crow", *yā*) is composed of 鸟 (meaning "bird") and 牙 (sound *yá*).
- The 中 (*zhōng*) component of 肿 ("swollen") is a clue to its pronunciation *zhǒng*, though the 月 ("moon") component is more loosely suggestive of its meaning.
- For a character like 法 ("law"), the components "water" and "go" do not provide much of a phonetic or semantic clue. This is the case with many characters.

The vast majority of characters have two top-level components, arranged either side-by-side (as in the examples above), top-bottom, or outside-inside. It should be noted that a top-level character component may often be recursively divided into further sub-components.

Generally speaking, it is impossible for a student to correctly guess the pronunciation or meaning of a new character, though their guess may be better than chance.

3 Data Preparation

From a Chinese Wikipedia dump,[3] we remove all non-Chinese characters, then convert to simplified characters. This forms our character corpus.

For our pronunciation corpus, we could record and transcribe Mandarin speech into *pinyin* syllables. Instead, we simulate this. We take a large subset of the Baidu Baike encyclopedia,[4] but then immediately convert it to tone-marked *pinyin* syllables, by using a comprehensive dictionary[5] of 116,524 words and phrases. 99.97% of Baike character tokens are covered by this dictionary.

We substitute Baike character sequences with *pinyin* sequences in left-to-right, longest-match fashion. This strategy works well most of the time. For example, it correctly pronounces 睡觉 as *shui jiao*, and 觉得 as *jue de*, despite the ambiguity of 觉. However, it incorrectly pronounces 想睡觉 because a dictionary entry 想睡 matches the phrase before 睡觉 = *shui jiao* can be applied; it also has trouble with single-character words like 还.

Using character sequences from Chinese Wikipedia and pinyin sequences from Baidu Baike is important. If we alternatively divided Chinese Wikipedia into two parts, unsupervised analysis could easily exploit high-frequency boilerplate expressions like 英重定向：这是由英名，指向中文名的重定向。它引出英至遵循命名常的合名，能助者作。("English Redirection: This is a redirect from the English name to the Chinese name. It guides the English title to a proper name that follows the naming convention and can assist the editor in writing.")

We also *pinyin*-ize the first 100 lines (6059 characters) of our character corpus, as a gold-standard reference set, for later judging how well we phonetically decipher the character corpus. Unless stated otherwise, all results are for token accuracy on this reference set.

Table 1 gives statistics on our corpora. We release our data at *https://github.com/c2huc2hu/unsupervised-chinese-pronunciation-data*.

We also record internal structure of syllables

[2]Non-Chinese speakers may want to visually confirm that 牙 and 鸟, suitably thinned and placed side-by-side, do indeed form the composite character 鸦.

[3]linguatools.org/tools/corpora/wikipedia-monolingual-corpora

[4]baike.baidu.com

[5]www.mdbg.net/chinese/dictionary?page=cc-cedict

	Tokens	Types	Singletons
Characters (Wikipedia)	510m	17,442	3,444
Syllables (Baike)	264m	w/o tones: 412 w/ tones: 1506	1 5
Test set (characters)	6059	783	236

Table 1: Token and type statistics for our non-parallel character and syllable corpora. Singletons are one-count types.

and characters. For syllables, we separate onset and rime (eg, *zhang* → *zh* + *ang*).

For characters, we employ the thorough graphical decompositions given in Wikimedia Commons,[6] which divide each character into (at most) two parts. This allows us to find, for example, 44 characters that include the second component 包 (咆, 孢, 狍, 炮, etc). We only use top-level decompositions in this work, forgoing any further recursive decompositions.

4 Supervised Comparison Points

Before turning to unsupervised methods, we briefly present two supervised comparison points.

First, if we had a large *parallel* stream of character tokens and their *pinyin* pronunciations, we could train a simple pronouncer that memorizes the most-frequent *pinyin* for each character type. Using the Baike data as processed above as a putative parallel resource, we obtain 99.1% pronunciation accuracy on the test set.[7] The only errors involve ambiguous characters, showing that a deterministic character-to-pinyin mapping table—whether obtained by memorization or by unsupervised methods—is sufficient to solve the bulk of the problem.

Second, to investigate whether written character components predict pronunciation, we use gold pronunciations of the most common 2000 characters to predict pronunciations of the next 1000. If a test character X has second (e.g., rightmost) written component Y, then we use the pronunciation of Y as a guess for the pronunciation of X. We find this works 25% of the time if we do not consider tones, and 17% of the time if we do. Table 2 confirms that character components are only a loose guide to pronunciation, even with supervision.

[6]commons.wikimedia.org/wiki/Commons:Chinese_characters_decomposition

[7]Experimental results from here on out refer to no-tone pronunciation.

Next we turn to unsupervised pronunciation, where we throw away pronunciation dictionaries and parallel data, working only from uncorrelated streams of characters and syllables.

5 Unsupervised Vector Method

Borrowing from unsupervised machine translation, which learns mappings between words in different languages (Lample et al., 2018a; Artetxe et al., 2018), we attempt to learn a mapping between embeddings for characters and embeddings for *pinyin* symbols. We train fastText (Bojanowski et al., 2017) vectors of dimension 300 and default settings on each of our corpora and use the MUSE system[8] to learn the relationship between the two vector spaces (Lample et al., 2018b).

There are two steps to this method: (1) map character vectors into *pinyin* space, (2) for each character type, find its nearest *pinyin* neighbor. This gives us a table that maps character types onto *pinyin* types. We apply this table to each character token of our 6059-character test set, obtaining token-level accuracy.

Unfortunately, this method does not work well. Only 0.5% of type mappings are correct, and token-level accuracy is similarly small. Reversing the mapping direction (*pinyin* embeddings into character embedding space) does not improve accuracy. It appears that the asymmetry of the mapping is difficult for the algorithm to capture. Each *pinyin* syllable should, in reality, be the nearest neighbor of many different characters. Moreover, the behavior of a *pinyin* syllable in running *pinyin* data may not be a good match for the behavior of any given character with that pronunciation.

Our next approach is to map words instead of characters. We break our long character sequence into a long word sequence, e.g., 竞争 很 激烈 by applying the Jieba tokenizer[9] to Wikipedia. We similarly break our long *pinyin* sequence into a long *pinyin*-word sequence, e.g., *wo xihuan chi jiaozi*, by applying the Stanford tokenizer[10] to pre-*pinyin*ized Baidu. We build embeddings for types on both sides, and we again map them into a shared space.

During the nearest-neighbor phase, we take each written-word and look for the closest *pinyin*-word, giving preference to *pinyin* words with the

[8]https://github.com/facebookresearch/MUSE

[9]github.com/fxsjy/jieba

[10]nlp.stanford.edu/software/segmenter.shtml

	Exact *pinyin* match, with tone	Exact *pinyin* without tone	Partial *pinyin*
Majority-class Baseline (*yù*)	0.01	0.02	0.19
Supervised Match 1	0.17	0.25	0.39
Supervised Match 2	0.19	0.28	0.48

Table 2: Even with a partial pronunciation dictionary, it is difficult to predict exact pronunciation of a new written character from its components. This table records accuracy of pronunciation guesses for characters 2001-3000 (by frequency), given pronunciations of characters 1-2000; for these types, *yù* is most frequent. Match 1 uses a character's second component, e.g., guessing (incorrectly) that 耗 (*hào*) is pronounced the same as 毛 (*mào*). Match 2 uses either the first or second component, whichever is better. Partial match credits either onset or rime, e.g., counting *hào* for *mào* as correct.

Source	Target	Tone?	Accuracy
Character	Pinyin	no	0.20%
Character	Pinyin	yes	0.05%
Pinyin	Character	no	0.15%
Pinyin	Character	yes	0.12%
Character word	Pinyin word	yes	81.41%

Table 3: Accuracy of vector-mapping approaches, measuring % of character tokens we assign the correct (no tone) *pinyin* pronunciation to. Testing is on the first 6059 characters of the character corpus.

same number of syllables as the written-word. If we cannot find a near neighbor with the correct number of syllables, we map to a sequence of *de*, the most common Chinese *pinyin* token.

We find that matched word pairs are much more accurate than the individual character-*pinyin* mappings we previously obtained.[11] To get token-level pronunciation accuracy, we segment our 6059-token character test set, apply our learned mapping table, and count how many characters are pronounced correctly. Table 3 shows that the accuracy of this method is 81.4%.

6 Unsupervised Noisy Channel EM Method

We next turn to a noisy-channel approach, following Knight and Yamada (1999). We consider our character sequence $C = c_1...c_n$ as derived from a hidden (no tone) pinyin sequence $P = p_1...p_n$:

$$\begin{aligned}&\text{argmax}_\theta \Pr(C) = \\ &\text{argmax}_\theta \sum_P \Pr(P) \cdot \Pr(C|P) = \\ &\text{argmax}_\theta \sum_P \Pr(P) \cdot \prod_{i=1}^{n} \Pr_\theta(c_i|p_i)\end{aligned}$$

$\Pr(P)$ is a fixed language model over *pinyin* sequences. $\Pr_\theta(c|p)$ values are modified to maximize the value of the whole expression. Examples of $\Pr_\theta(c|p)$ parameters are Pr(中 | *zhong*), which we hope to be relatively high, and Pr(很 | *zhong*), which we hope to be zero.

6.1 Previous Noisy-Channel

We first faithfully re-implement Knight and Yamada (1999). They drive decipherment using a bigram $\Pr(P)$, pruning *pinyin* pairs that occur few than 5 times.

Unfortunately, they do not provide their training data or code, giving only the number of character types as 2113, and the number of observed *pinyin* pair types as 1177 (after pruning pairs occurring fewer than 5 times). Using our own data, we estimate their character corpus at ~30,000 tokens.

We applied this re-implementation to our data. Their *pinyin*-pair pruning has little effect, due to the size of our *pinyin* corpus (155,219 unique pairs). We ran their expectation-maximization (EM) algorithm for 170 iterations on a character corpus of 300,000 tokens, then applied their decoding algorithm to our 6059-token test, obtaining a token pronunciation accuracy of 8.6%. Because this accuracy is lower than their reported 22%, we confirmed our results with two separate implementations, and we took the best of 10 random restarts. Increasing the character corpus size to 10m yielded a worse 5.1% accuracy. We conjecture that Knight and Yamada (1999) used more homogeneous data.

6.2 Our Noisy-Channel

In this work, we use a *pinyin*-trigram model (rather than bigram), and we apply efficiency tricks that allow us to scale to our large data.

First, we reduce our character data C to a list of unique triples C_{tri}, recording count($c_1c_2c_3$) for each triple. A sample character triple is "的 人 口" (count = 43485).

Likewise, we reduce our *pinyin* training data to triples, sorted by unsmoothed probability Pr(p) =

[11]Similar to Marchisio et al. (2020) and Kim et al. (2020), we note that unsupervised translation techniques require certain types of data to work well.

```
Given:
  Character triples $c_1c_2c_3 \in C_{tri}$, with counts
  Pinyin triples $p_1p_2p_3$, with probabilities

Produce:
  Values for Pr$_\theta(c|p)$

Do:
  initialize Pr$_\theta(c|p)$ table (uniform, random)
  for k = 1 to max_iterations
    initialize Pr$_k(C_{tri})$ = 1.0
    count($c$, $p$) = 0   (whole table)
    for each of top N character triples $c_1c_2c_3$
      sum = 0
      for each of top M pinyin triples $p_1p_2p_3$
        score($p_1p_2p_3$) =
          Pr($p_1p_2p_3$) Pr$_\theta(c_1|p_1)$ Pr$_\theta(c_2|p_2)$ Pr$_\theta(c_3|p_3)$
        sum += score($p_1p_2p_3$)
      Pr$_k(C_{tri})$ = Pr$_k(C_{tri})$ · sum$^{\text{count}(c_1c_2c_3)}$
      for each of top M pinyin triples $p_1p_2p_3$
        Pr($p_1p_2p_3|c_1c_2c_3$) = score($p_1p_2p_3$) / sum
        for i = 1 to 3
          count($c_i$, $p_i$) +=
            Pr($p_1p_2p_3|c_1c_2c_3$) · count($c_1c_2c_3$)
    normalize count($c$, $p$) into P$_\theta(c|p)$
  return final Pr$_\theta(c|p)$ table
```

Figure 2: EM algorithm for revising Pr$_\theta(c|p)$ parameters (*pinyin*-to-character substitution probabilities) to iteratively improve the probability of observed character triples C. EM guarantees Pr$_i(C_{tri}) \geq$ Pr$_{i-1}(C_{tri})$.

normalized count($p_1p_2p_3$). A sample *pinyin* triple is "*de ren kou*" (probability = $8{\cdot}10^{-6}$).

Our training objective now becomes:

$$\begin{aligned}
&\text{argmax}_\theta \ \Pr(C_{tri}) = \\
&\text{argmax}_\theta \ \textstyle\prod_{c_1c_2c_3 \in C_{tri}} \Pr(c_1c_2c_3) = \\
&\text{argmax}_\theta \ \textstyle\prod_{c_1c_2c_3 \in C_{tri}} \\
&\qquad \textstyle\sum_{p_1p_2p_3} \Pr(p_1p_2p_3) \cdot \Pr(c_1c_2c_3|p_1p_2p_3) = \\
&\text{argmax}_\theta \ \textstyle\prod_{c_1c_2c_3 \in C_{tri}} \\
&\qquad \textstyle\sum_{p_1p_2p_3} \Pr(p_1p_2p_3) \cdot \\
&\qquad\qquad \Pr_\theta(c_1|p_1)\Pr_\theta(c_2|p_2)\Pr_\theta(c_3|p_3)
\end{aligned}$$

Figure 2 gives an expectation-maximization (EM) algorithm for choosing Pr$_\theta(c|p)$ to find a local maximum in Pr(C_{tri}).

After we have obtained Pr$_\theta(c|p)$ values, we decode our 6059-character-token test sequence ($C = c_1...c_n$) using the standard Viterbi algorithm (Viterbi, 1967) Our decoding criterion is:

$$\begin{aligned}
&\text{argmax}_P \ \Pr(P|C) = \\
&\text{argmax}_P \ \Pr(P) \cdot \Pr(C|P) = \\
&\text{argmax}_P \ \Pr(P) \cdot \textstyle\prod_{i=1}^{n} \Pr_\theta(c_i|p_i) \approx \\
&\text{argmax}_P \ \Pr(P) \cdot \textstyle\prod_{i=1}^{n} \Pr_\theta(c_i|p_i)^3
\end{aligned}$$

```
Given:
  Test character string $c_1...c_n$
  Substitution model Pr$_\theta(c|p)$ from EM
  *Pinyin* bigram model Pr($p_2|p_1$)

Produce:
  Phonetic decoding $p_1...p_n$

Do:
  Standard Viterbi algorithm (Viterbi, 1967)
```

Figure 3: Pronouncing a character sequence $c_1...c_n$, using a *pinyin* bigram model Pr($p_2|p_1$) and EM-optimized Pr$_\theta(c|p)$ values.

N=M	EM iterations	Test accuracy
10k	20	29 - 44 %
10k	100	50 %
20k	20	37 - 46 %
20k	100	58 - 62 %
100k	100	71 %

Table 4: Accuracy of noisy-channel decoding after EM training. N is the number of unique character triples shown to EM, and M is the number of unique pinyin triples available to "explain" each character triple. Accuracy ranges denote multiple random restarts.

where Pr(P) is implemented with a smoothed bigram *pinyin* model Pr($p_2|p_1$). Figure 3 gives the outline. While EM only considers the top M *pinyin* triples, final decoding works on entire sentences and is free to create previously-unseen *pinyin* trigrams. Decoding is also free to pronounce the same character in different ways, depending on its context. We follow the prior work in cubing channel model values.

Because different random restarts yield different accuracy results, we report ranges. We are generally able to identify the best restart in an unsupervised way, due to the high correlation between EM's objective Pr(C_{tri}) and test-set accuracy.

Table 4 shows decoding accuracy results. We achieve 71%, substantially beating the 22% accuracy reported by Knight and Yamada (1999), as well as the 8.6% of a re-implementation applied to our data.

7 Exploiting Character Components

We next investigate whether character components can improve EM results. Instead of storing

N=M	EM iterations	Test accuracy w/o hints	Test accuracy w/ hints
20k	20	37 - 46 %	72 - 73 % (+27%)
20k	100	58 - 62 %	72 % (+10%)
100k	100	71 %	81 % (+10%)

Table 5: Improving EM results by assigning high initial weights to the 261 agreed-on mappings ("hints") from EM and vector-based methods. Accuracy ranges are due to multiple random restarts.

$\Pr_\theta(c|p)$ in a single lookup table, we compute it from five lookup tables ($\Pr_1..\Pr_5$):

$$\begin{aligned}&\Pr_\theta(c|p) = \\ &\quad \lambda_1 \cdot \Pr_1(c|p) + \\ &\quad \lambda_2 \cdot \Pr_2(part1(c)|p) \cdot \Pr_3(c|part1(c)) \\ &\quad \lambda_3 \cdot \Pr_4(part2(c)|p) \cdot \Pr_5(c|part2(c))\end{aligned}$$

As EM establishes a tentative high value for $\Pr_1$(排 | *pai*), we hope to also create a high value for $\Pr_4$(非 | *pai*), which will encourage *pai* to map to other characters with component 非 (such as 徘) in the following EM iterations.

Unfortunately, while we do see compelling $\Pr_4$ entries, we do not see an overall improvement in test-set accuracy from this method.

8 Combining EM and Vector Methods

The EM method gives 71% accuracy, while the vector method gives 81% accuracy. We find that the two methods agree 47% of time, and are 98.7% accurate in agreement cases, so in an unsupervised way, we distill out 261 high-confidence character/pinyin mappings.

Improved EM results. We use the 261 high-confidence (c, p) mappings as our initial start point, by replacing each one's random initial $P_\theta(c|p)$ value with a 1.0 weight. These weights bias the fractional counting in the first EM iteration. Table 5 shows that high-confidence mappings increase overall EM accuracy from 71% to 81%.

Improved vector-based results. We run the same initial procedure from Section 5, giving us a vector space inhabited by both written words and *pinyin* words. However, we modify the nearest-neighbor search that produces word/pronunciation mappings. Our modified nearest-neighbor search takes a written word's vector and returns the nearest pronunciation vector that is consistent with the 261 high-confidence (c, p) mappings. For example, given the word 重要, we prefer neighbors *dang yao* and *zhong yao* over *dang pin*, because (要, *yao*) is one of the high-confidence mappings originally proposed by both EM and vector-based approaches. We also use high-confidence mappings to improve *de* sequences (Section 5). This combination technique is also highly effective, raising accuracy from 81% to 89%.

9 Conclusion

We implement and evaluate techniques to pronounce Chinese text in Mandarin, without the use of a pronunciation dictionary or parallel resource. The EM method achieves a test-set accuracy of 71%, while the vector-based method achieves 81%. By combining the two methods, we obtain 89% accuracy, which significantly exceeds that of prior work.

We also demonstrate that current methods for unsupervised matching of vector spaces are sensitive to the structure of the spaces. In the presence of one-to-many mappings between *pinyin* and characters, the mapping accuracy is severely downgraded, leaving open an opportunity to design more robust unsupervised vector mapping systems.

References

M. Artetxe, G. Labaka, and E. Agirre. 2018. Unsupervised statistical machine translation. In *Proc. EMNLP*.

R. S. Berndt, J. A. Reggia, and C. C. Mitchum. 1987. Empirically derived probabilities for grapheme-to-phoneme correspondences in English. *Behavior Research Methods, Instruments, Computers*, 19.

M. Bisani and H. Ney. 2008. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50.

P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5.

Y. Kim, M. Graça, and H. Ney. 2020. When and why is unsupervised neural machine translation useless? In *Proc. EACL*.

K. Knight and K. Yamada. 1999. A computational approach to deciphering unknown scripts. In *Proc. ACL Workshop on Unsupervised Learning in Natural Language Processing*.

G. Lample, A. Conneau, L. Denoyer, and M. Ranzato. 2018a. Unsupervised machine translation using monolingual corpora only. In *Proc. ICLR*.

G. Lample, A. Conneau, M. Ranzato, L. Denoyer, and H. Jégou. 2018b. Word translation without parallel data. In *Proc. ICLR*.

K. Marchisio, K. Duh, and P. Koehn. 2020. When does unsupervised machine translation work? *CoRR*, arXiv:2004.05516.

B. Peters, J. Dehdari, and J. van Genabith. 2017. Massively multilingual neural grapheme-to-phoneme conversion. In *Proc. of the First Workshop on Building Linguistically Generalizable NLP Systems*.

A. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2).

J. Xu, G. Fu, and H. Li. 2004. Grapheme-to-phoneme conversion for Chinese text-to-speech. In *Proc. Interspeech*.

B. Zhang, H. Huang, X. Pan, H. Ji, K. Knight, Z. Wen, Y. Sun, J. Han, and B. Yener. 2014. Be appropriate and funny: Automatic entity morph encoding. In *Proc. ACL*.

T. Zhang, A. Chowdhury, N. Dhulekar, J. Xia, K. Knight, H. Ji, B. Yener, and L. Zhao. 2016. From image to translation: Processing the endangered Nyushu script. *ACM Trans. Asian Low-Resource Lang. Inf. Process.*, 15.

Z. Zhang, M. Chu, and E. Chang. 2002. An efficient way to learn rules for grapheme-to-phoneme conversion in Chinese. In *Proc. ISCSLP*.