English-Chinese Name Machine Transliteration Using Search and Neural Network Models

Julia Gong Benjamin Newman
{jxgong, blnewman}@stanford.edu

***Abstract*—Recently, much progress has been made on developing accurate machine translation systems. However, because these systems focus on translating word meanings, they don't translate names between languages as effectively, as name transliteration depends on translating sound. To address this gap, in this project, we focus on the transliteration of English names into Chinese. We propose many search-based approaches, as well as sequence-to-sequence deep learning models. We find that search-based methods outperform deep learning ones, likely due to the relatively small number of English names with standard Chinese translations in the accessible dataset. Despite determining that incorporating syllable length heuristics and phonetic information into the search improves performance significantly, we are unable to generate native-speaker level translations. We hope this work provides a good starting point for more future work in the area of machine transliteration.**

I. Introduction

While automated translation systems have made huge leaps in accuracy in recent years, they still lag behind in translating names that originate from one language into the writing system of another. Intuitively, this may be because translating names is fundamentally different from translating other parts of speech. Modern machine translation systems translate based on word or utterance meanings, but names do not have any intrinsic meaning, so they must be translated on the pronunciation level. For languages with consistent pronunciation rules, such as Spanish, this may not be too much of a problem, but for languages like English, it is a more difficult task. In this work, we will investigate the possibility of translating names written in English into the Chinese pronunciation representation: pinyin.

Translating English names into pinyin presents an additional challenge; many English names have widely-accepted standard Chinese translations whose pronunciations do not necessarily symbolically match the English translations on paper, though they may phonetically still sound similar. Some examples that exhibit this behavior include:

Erasmo → ***āi lā*** **sī** **mò**

Julia → ***zhū*** **lì** **yà**

Roy → ***luō yī***

Archana → **ā** ***qí*** **nà**

Matthew → **mǎ** ***xiū***

In many ways, this task is thus more similar to machine transliteration than machine translation. Transliteration tasks are most often solved with grapheme to phoneme (g2p) systems. These systems take in written text (i.e. graphemes) and output some representation of their pronunciations, either in the international phonetic alphabet (IPA) or the ASCII-friendly ARPAbet. This process is often associated with text-to-speech software, such as accessibility readers or personal assistants, that have to read web content aloud. The difference between this application and our task is that we must also take the additional step of converting the phonemic representation into pinyin rather than acoustic signals. There have been a number of different approaches toward solving these tasks in the past. These include linear classifiers, such as support vector machines (SVMs); automata-based methods, such as conditional random fields and weighted finite-state transducers; deep learning methods, such as recurrent neural networks with LSTM modules; and Bayesian approaches, such as joint-sequence models and Hidden Markov models [1].

Our task is slightly different from g2p systems because, instead of abstracting phonemes into a universal phonetic script (e.g. the International Phonetic Alphabet), we must limit ourselves to valid phonemes in Chinese. Others have attempted similar versions of this problem, such as Wan and Verspoor, who use a rules-based method with modest success [2]. Shao et al. use a modified version of the M2M aligner to segment the English words [3]. Upadhyay et al. have had good success using attention-based RNN methods with low resource languages, a simpler version of which we try to adopt in this work [4].

II. Methodology

We take two general approaches to solving this problem; we create a search model and a deep learning model. For search, we propose a number of approaches based on empirical English-pinyin transliteration frequencies. For deep learning, we use a recurrent neural network to find sequential features.

A. Data

We obtained 1510 common English names (both traditionally male and female) and their Chinese character equivalents from [5]. We then used the online translation API GLOSBE to obtain the space-separated pinyin for each of the characters [6]. Pinyin is a system of the pronunciation glosses used in Mandarin Chinese (普通话) that represents its pronunciation. Each pinyin consists of three components—an initial, a final, and one of five tones. For example, the pinyin **zhuāng** has an initial **zh**, a final **uang**, and first tone (the flat line above the **a**). Each entry of the dataset we compiled includes the English name, the Chinese characters of the ground truth transliteration, the pinyin of these characters, and the traditional gender of the name. A sample data entry in the dataset thus might look as follows:

Alice, 爱丽丝, **ài lì sī**, f

B. Metric

Because we have ground-truth data, to score a potential transliteration, we can compare it to this ground-truth. We do so using a modified edit distance metric that we call pinyin edit distance. The edit distance between two strings is the minimum number of insertions, deletions, or substitutions that it takes to transform the first string into the second one. We represent this edit distance with the function $\texttt{edit\_distance}(\cdot,\cdot)$. However, for our purposes, we modify this metric slightly by penalizing pinyin that have the same vowels with differing tones half as much as we would if they were completely different characters. We use the function $\texttt{edit\_distance}_{pin}(\cdot,\cdot)$ to denote this modified edit distance between its pinyin arguments.

C. Baseline and Oracle

First, we define a baseline and an oracle for this transliteration task. These give an lower and upper bound, respectively, on how well we expect to perform.

1) Baseline: Our baseline algorithm provides a lower bound on our expected performance. It uses the following rules-based method to transliterate a name. Given an English string, the string can be segmented into syllables that are comprised of either a consonant (C), a vowel (V), a consonant followed by vowel (CV), or a consonant followed by a vowel and another consonant (CVC). CVC is always preferred over CV-C when the character after the second consonant is another consonant (i.e. CVCV will be parsed as CV-CV, while CVCC will be parsed as CVC-C). When the CV and CVC rules fail to apply, the default value of the lone character (either C or V) is used.

Next, a standardized transcription look-up table is used to transform the segmented English names into Chinese Characters. The table columns contain the pinyin initials written in a the ARPABET (e.g. K, R, ZH, SH) and the rows have the pinyin finals written in the same form (e.g. AA, UW, AEN, IHNG). The intersection of a row and column contains a common Chinese character corresponding to the combination of these sounds (including cases where only a vowel or a consonant is provided).

For each syllable in the input English string, the baseline algorithm converts the CV, CVC, C, or V segment from the original English string into the corresponding IPA column (initial) and row (final) that has the lowest `edit_distance`. For example, the 'LI' in 'ALICE' has the lowest `edit_distance` to the initial-final combination 'L' and 'IY', yielding 'LIY', which is the chosen lookup in the table that ultimately yields '利'. The syllable 'LI' is then assigned the character '利' in the final transliteration.

When we presented our baseline with our entire dataset, it achieved an average edit distance of approximately 3.56.

2) Oracle: For our oracle, we asked two native Chinese speakers to independently transliterate a list of 30 anglicized (e.g. 'Golrokh') or English (e.g. 'Carly') names to the best of their knowledge. Of the 30 names, 12 names differed between the two oracles, and the average distance metric between the oracles' answers was only 0.8333. The oracles were very consistent with one another for the majority of the names, and the distance metric reflects how minute the differences were when they arose (often being merely a tone difference). Names such as 'Alice' or 'Jake', which have known standard transcriptions, were all correctly identified by both oracles. This provides an upper bound for our translation.

To compare the baseline and the oracle, we presented the baseline with the same list of 30 names—some of them common names and others English transliterations of names in other languages (See

VII for a list of names). Of the 30 names assessed by the oracles, none of the baseline transcriptions perfectly matched the oracles' answers. The average $\texttt{edit\_distance}_{pin}$ between oracles was 0.8333 indicating a lot of agreement, but the average $\texttt{edit\_distance}_{pin}$ between each of the oracles and the baseline was 3.4417. The disparity between these two distances indicates that there is potential for an improved algorithm and the development of a better transliteration system from an English name to a segmented pinyin string.

We take two approaches to solving this problem. First, we formalize it as a search problem by calculating a cost for each possible transliteration and trying to minimize this cost. Next, we treat this problem as a classification problem, and use a recurrent neural network with an encoder-decoder structure.

III. Search Model

At a high level, formalizing transliteration as a search problem involves defining a cost for every possible pinyin representation of an English name and choosing the transliteration with the lowest cost. There are two basic approaches to this search problem—one based on the edit distance metric, and one based on a pinyin-English co-occurrence table.

For both problems, we first determine how to assign a cost to a transcription. We want to include frequent pinyin in names, so we create a pinyin cost function, `cost`, similar to the one presented in the `reconstruct` assignment. We define the cost of a pinyin syllable to be its surprisal ($-\log(p)$, where p is the maximum likelihood estimate of the probability of a pinyin syllable in our data set, i.e. the number of times the pinyin syllable appears divided by the total number of pinyin syllables in the corpus). Because the names are short, a unigram cost function is sufficient and a more fine-grained bi-syllable cost function is unnecessary.

A. Edit Distance Search

Next, we formulate the search problem. Given a single English name, our initial thought was to first break it up into syllables and then find the pinyin that most closely matches each syllable. Unfortunately, this turns out to not be ideal in many cases. In our dataset, there are numerous examples of single syllables being mapped to multiple pinyin (the 'br' in 'Bruce' maps to **bù lǔ**), as well as, though much rarer, multiple syllables being mapped to a single pinyin ('riel' in 'Ariel' maps to **liè**). To take into account this variation, we start by considering all possible segmentations of the given name (2^n total possibilities for an English name of length n). Even though this approach grows exponentially with the length of the name, names tend to be short enough that this does not pose a significant problem. For example, when transliterating the name BEN, we would consider the segmentations:

$$\{\text{B, EN}\}, \{\text{BE, N}\}, \{\text{B, E, N}\}, \text{ and } \{\text{BEN}\}$$

We denote the set of all segmentations as $\mathcal{S}$.

Now, we have to calculate the cost of each segmentation, $S \in \mathcal{S}$. We can do this by adding together the costs of the individual subwords $sw \in S$. This is a bit challenging because the cost function we described earlier provides the cost of a pinyin syllable, not of an English sub-word. Calling the cost function directly on the English sub-word is an option, but there will be many English sub-words that are not possible pinyin, and the cost function will have no way of assigning them a cost. To this end, we ultimately decided to assign an English sub-word a pinyin by iterating through all the pinyin in our dataset and finding the pinyin that minimizes the product:

$$\texttt{edit\_distance}(\text{subword, pinyin}) \cdot \texttt{cost}(\text{pinyin})$$

This means the cost of a segmentation S is the sum of all the scores of the sub-words, sw. This is the cost we try to minimize over all possible segmentations. The key equation for deriving the optimal segmentation is thus:

$$\underset{S \in \mathcal{S}}{\operatorname{argmin}} \sum_{sw \in S} \min_{p \in \text{pinyin}} \texttt{edit\_distance}(sw, p) \cdot \texttt{cost}(p)$$

To extract the pinyin from the optimal segmentation, we just use the ones that were found when we were performing the search. Note that the edit distance here is *not* $\texttt{edit\_distance}_{pin}$. This is because we are finding the number of insertion, deletions, etc. between characters in two different writing systems, not between two pinyin strings.

Here, edit distance is defined similarly to how we defined it above. Because we are more likely to need to make some additions or deletions rather than substitutions, we assign lower costs to strings that start out more similar to one another by subtracting the total number of characters the two strings have in common from their final edit distance.

That said, it is not ideal to use the edit distance between these two languages. A simple reason why not might be as simple as looking at how the 'Ju' in 'Julia' is transliterated as **zhū**. The edit distance here is two characters, one for the substitution of the 'z' for the 'j' and the other for the insertion of the 'h'. This is despite the fact that **zh** and 'j' sound almost identical in Chinese and English (respectively). In the

next section, we describe an approach we use to try to mitigate these phonemic discrepancies.

B. Improved Edit Distance with Phoneme Considerations

As previously mentioned, the written edit distance does not perfectly capture the phonetic differences and similarities between the two languages. To address this issue, we implement a pre-processing function to adjust the English name strings before passing them to the dynamic programming search algorithm. This function uses preset rules to perform phoneme conversions on the English strings to convert common convention-equivalent phonemes to the appropriate Chinese pinyin characters. For example, one rule encodes converting all instances of 'ia' in the English string to 'iya' in the phoneme-adjusted English string. Other rules include replacing double-letters with single instances (e.g. 'tt' to 't', 'll' to 'l'), replacing 'r' with 'l', replacing 'ph' with 'f', replacing 'th' with 'x', and replacing 'v' with 'w'. This helped significantly decrease the average $\texttt{edit\_distance}_{pin}$, as seen in Table II. The predicted pinyin transliterations had an average edit distance of 3.4538, and an average edit distance of 3.4718 for just the names that differed from the targets. 4.50% of the predicted pinyin matched the target exactly.

C. Improved Edit Distance with Syllable Number Heuristic

One category of error we noticed in our transliterations was the selection of lowest-cost names that had too few or too many syllables, even though there were names that were much closer to the correct length and had only a slightly higher total cost. We saw this as a flaw in the calculation of the total cost of a full pinyin transliteration. Thus, to improve the cost metric, we introduced another heuristic to factor into the search algorithm: an estimate of the number of syllables. This metric produces an estimated correct number of syllables for the target pinyin transliteration as a function of the phoneme-adjusted English string input. We then modified the equation for deriving the optimal segmentation to include the difference between the heuristic `syll_heuristic` and the actual number of syllables in the proposed pinyin transliteration `num_syll` (which is the same as the number of subwords in the segmentation):

$$\underset{S\in\mathcal{S}}{\operatorname{argmin}}\Big(\sum_{sw\in S}\min_{p\in\text{pinyin}}\texttt{edit\_distance}(sw,p)\cdot\texttt{cost}(p) + 10(\texttt{syll\_heuristic}(S)-\texttt{num\_syll}(S))\Big)$$

We approached the implementation of the syllable length heuristic in two ways: a model-based technique and a rules-based technique.

1) Model-Based Syllable Heuristic: For the model-based heuristic, we built a linear regression model with two independent variables: the length of the phoneme-adjusted English string, and the number of consonants in the phoneme-adjusted English string. The model outputs the predicted number of syllables in the target pinyin for that English name. The model achieved a mean squared error of 0.6987 for predicting the target number of syllables. When using the cost function as defined above in the search model with the model-based syllable heuristic, the predicted pinyin transliterations had an average edit distance of 3.3212, and an average edit distance of 3.4754 for just the names that differed from the targets. 4.44% of the predicted pinyin matched the target exactly.

2) Rules-Based Syllable Heuristic: For the rules-based heuristic, we used a similar syllable segmentation algorithm based on consonant (C) and vowel (V) patterns to the one used in the baseline algorithm. The possible syllable patterns are C, V, CV, VC, CVV, and CVC. See pseudocode in Algorithm 1.

Because the minimum number of syllables is 2 for Chinese names, we also impose this restriction on the heuristic. The mean squared error of the predictions provided by the rules-based heuristic was 1.0523. When using the same cost function in the search model, the predicted pinyin transliterations had an average edit distance of 3.2609, and an average edit distance of 3.4147 for just the names that differed from the targets. 4.50% of the predicted pinyin matched the target exactly.

Despite the higher mean squared error, the rules-based syllable heuristic was more successful than the model-based heuristic overall, likely because the model-based heuristic overfitted on the small provided dataset. It also did much better on transliterating names with no standard translation, such as translating 'Golrokh' into **guǒ luò kè**. We proceeded to use the rules-based syllable heuristic for the edit distance-based search problem.

D. Improved Edit Distance Metric with Error Type Considerations

The final improvement we added for the edit distance-based search problem involved weighting different kinds of character differences between the predicted and target pinyin. The simple edit distance metric we used did not take into account the types of characters being substituted, deleted, or added; for example, having an 'a' in place of a 't' was weighted the same as having an 'a' in place of an 'e'. We

```
for each letter in phoneme-adjusted English do
    if C then
        if next letter is C and CC is not valid two-consonant syllable ('th', 'sh', 'ch', 'zh', 'kh') then
            use C ;
            num_syllables += 1 ;
        else if next letter is C then
            if following letter is V and next following letter is V then
                Use CVV ;
                num_syllables += 1 ;
            if following letter is V andnext following letter is C and not 'n' then
                use CV ;
                num_syllables += 1 ;
            else
                use VC ;
                num_syllables += 1 ;
    end
    if V then
        if last character then
            use V ;
            num_syllables += 1 ;
        else
            use VC ;
            num_syllables += 1 ;
    end
    increment to next unprocessed letter ;
end
return max(2, num_syllables) ;
```

Algorithm 1: Rules-Based Heuristic

noticed that names generally were less accurate when the target string and predicted string differed in the character type—in other words, when a vowel was in the place of a consonant, or when a consonant was in the place of a vowel. Thus, we modified our edit distance metric to incorporate this unequal weight, assigning a higher cost (2) to errors of this type than the cost for same-type errors (1). With this additional weighted edit distance metric, the accuracy of segmentations improved substantially. The predicted pinyin transliterations had an average edit distance of 3.2026, and an average edit distance of 3.3747 for just the names that differed from the targets. 5.1% of the predicted pinyin matched the target exactly.

E. Co-occurrence Table

In the above search formulations, we have used edit distance to decide which pinyin are more likely to be matched with which English sub-words. This works well, especially after we adjust for some of the common sound substitutions that occur during transliteration. An alternative approach is to use the data we have to do more than just define a global cost function. We can define a more fine-grained cost function based on what pinyin tend to be in translations of names with certain combinations of letters. To do this, we construct a table whose columns are the pinyin in the dataset and whose rows are n-grams from the names. For instance, given the names 'Julia' and 'Lia', and looking at just unigrams and bigrams, we arrive a the example in Table I above.

	zhū	**lì**	**yà**	...
J	1	1	1	...
U	1	1	1	...
L	1	2	2	...
I	1	2	2	...
A	1	2	2	...
JU	1	1	1	...
UL	1	1	1	...
LI	1	2	2	...
IA	1	2	2	...
...	...	...	...	...

TABLE I
EXAMPLE SECTION OF A CO-OCCURRENCE TABLE FOCUSING ON UNIGRAMS AND BIGRAMS FOR THE NAMES 'JULIA' AND 'LIA'

We construct this co-occurrence table for all of the names including all of the names in our dataset. As in the other search formulations, when given a name, we iterate over all possible segmentations of the name. For each sub-word in a segmentation, we find all of its n-grams (where $n \in \{1, 2, 3\}$, and for each n-gram, we take the top k pinyin that it co-occurs with. We use unigrams through trigrams because because most English syllables are at most three letters and because using larger n might lead to overfitting. We found empirically that $k = 2$ tended to lead to the best results, capturing the most common pinyin without including too much background noise.

Next, with our list of tuples consisting of the n-grams and associated top k pinyin with their counts, we computed a score for each pinyin. First, we normalized the counts by the number of times the pinyin appeared in the dataset (akin to the `cost` function). This was so that common pinyin did not dominate in transliterations of English names where an uncommon pinyin just happened to co-occur with a more common one. Additionally, because there are only 26 English unigrams compared to 676 possible bigrams and 17,576 possible trigrams, we expected unigram co-occurrence counts to consist principally of the most common pinyin. This turned out to be the case in general—unigrams, particularly vowels and uncommon consonants, tended to just reflect

background noise. Bigrams and trigrams give a little more fine grained control, so we up-weighted the counts of them by a factor of 26 (because bigrams are approximately 26 more times less likely to occur than a particular unigram). For consistency, we should have up-weighted trigrams by another factor of 26, but because of the size of our dataset, doing so gave trigrams too much weight in deciding the resulting pinyin. Finally, to further reduce the influence of background noise, we made one more adjustment to the pinyin counts to consider the actual sounds it contains. We counted up how many times each individual initial or final appeared in the pinyin and increased the counts of the common ones. For example, if the candidate pinyin were **běn**, **bā**, and **sī**, the **běn**, **bā** would both have their counts increased because they share the common initial **b** while **sī** would not. This count is then the final score for the pinyin, and the pinyin with the highest score is the predicted pinyin for that sub-word.

To summarize, obtaining the pinyin and score for a sub-word consists of:

1) Finding all n-grams of the sub-word
2) Getting candidate pinyin by looking at the most common co-occurring pinyin from co-occurrence table
3) For each pinyin:
    a) Normalize by frequency of pinyin
    b) Normalize by length of the n-grams
    c) adjust score based on common characters with other candidate pinyin

We then sum up the score of each sub-word to arrive at a score for the segmentation. The segmentation with the highest score is the result, and the pinyin associated with the scores of the sub-words in that segmentation are the pinyin that are returned.

We take a few additional steps to ensure that correct segmentation is found.

1) English names that start or end with vowels often have different pinyin to account for those vowels than names that have the same vowels in the middle. For example, the transliteration of the name 'Aaron' starts with **ā**, but the name 'Baron' starts with **bā**. To account for this, we introduce a start and end token to names that start or end with a vowel, respectively. Bigrams with these tokens co-occur with particular pinyin in the dataset and these pinyin are potentially more representative candidates.
2) To prevent long segmentations from accumulating score that are too large, we use a heuristic to limit the maximum length of the segmentation. The heuristic involves calculating where hyphens could appear in the names, the number of syllables, if the name is in the CMU phonetic dictionary, and the number of consonant clusters [7], [8].
3) For certain transliterations, some pinyin depend on sounds that are outside their segmentation, or in other words, some English sounds are represented in multiple pinyin. For example, in 'Julia', the 'i' influences both the pinyin **lì** and **yà**, but no segmentation would capture this—they would either be {LI, A } or {L, IA }. To address this issue, when double vowels or consonants arise in names, we consider segmentations with the the letters completely separated, and with both on each side, and one on the other. For example, with 'Julia', we consider { LI, A }, { LI, IA }, and {LIA, A }.

F. Results

For each of these trials, we found the predicted translations of all 1510 names in our dataset. In table II we summarize the results from these trials. We present the average $\texttt{edit\_distance}_{pin}$, as well as the percentage of names that were predicted exactly correctly. There are two ways to calculate the average $\texttt{edit\_distance}_{pin}$. The first is to compute it over all the names. However, this deflates the metric a little bit, because it also takes into account names that we predicted correctly (which have an edit distance of 0). Instead, we compute the average $\texttt{edit\_distance}_{pin}$ over all names we classify incorrectly, as well as the percentage of names we do classify correctly. That gives us a sense of how accurate our predictions are and how wrong they are when they are incorrect.

Unfortunately, our original search formulation, which takes into account the edit distance between the English subwords and the pinyin, does much worse than our baseline. We take a closer look at some example names to try to understand why:

Benjamin $\longrightarrow$ **běn jiā mǐn**

This is in comparison to the ground truth transliteration:

Benjamin $\longrightarrow$ **běn jié míng**

Here, we can see that the name 'Benjamin' gives the pinyin **běn**, **jiā**, and **mǐn**, which is actually relatively close to the actual transliteration.

However, running this algorithm with the name 'Julia' gives the following result:

Julia $\longrightarrow$ **bù lián**

Approach	Avg $\texttt{edit\_distance}_{pin}$	% correct
Baseline	3.65	4.70
ED	4.36	1.40
ED + Phoneme	3.45	4.50
ED + SC	3.47	4.44
ED + SH	3.41	4.50
Co-occurrence	3.55	8.30

TABLE II
THE RESULTS FROM THE SEARCH-BASED APPROACHES WITH AVG EDIT DISTANCE FOR NAMES THAT DIFFERED FROM GROUND TRUTH PINYIN. ED IS EDIT DISTANCE. SC INCORPORATES THE MODEL-BASED COST FOR STRAYING FAR FROM THE SYLLABLE LENGTH HEURISTIC RESULTS. SH USES THE RULES-BASED SYLLABLE LENGTH HEURISTIC.

Approach	Benjamin	Julia
Baseline	**bān jià míng**	**zhū lì ā**
ED	**běn jiā mǐn**	**bù lián**
ED + Phoneme	**běn jiā mǐn**	**zhū lì yà**
ED + SC	**běn jiā mǐn**	**zhū lì yà**
ED + SH	**běn jiā mǐn**	**zhū lì yà**
Co-ocurrence	**bān jié míng**	**zhū lì yà**

TABLE III
EXAMPLE TRANSLATIONS FOR BENJAMIN AND JULIA USING VARIOUS SEARCH METHODS

As compared to the ground truth transliteration:

Julia $\longrightarrow$ **zhū lì yà**

The pinyin **bù**, **lián** is incorrect both in the number of syllables (there should be three) and in the sounds that they make (**bu** and **ju** are not very related). Part of the problem likely relates to the fact that there is no pinyin sequence **ju** of any kind in our dataset.

For the translations of these names from all the search techniques we used, see table III.

IV. Deep Learning

A. Seq2Seq

Next, we take a deep learning approach to solve the problem of transliterating English names into Chinese. As deep learning itself is not really the focus of the class, we treat this approach similarly to a classification problem. Because we are training a model and then evaluating it, we split the dataset of 1510 names into a training set of 1410 names and a test set of 100 names. We use a modified version of the machine translation tutorial code available on the Pytorch website. Here, we use an encoder-decoder model. First, we calculate an embedding of each character in the name and then use a gated recurrent unit (GRU) layer to learn sequential relationships between characters and produce a final encoding. The GRU output consists of a character encoding and a hidden state representing the "context" of the name. Then, to train the decoder, we calculate an embedding for each character and use the hidden state we calculate in the encoder stage. We run the embedding through another GRU layer and then put the output through a linear layer to act as a classifier for what the next character of the pinyin translation should be (See figure 1). Our input layer is the size of the number of English characters we have; the output layer is of the number of pinyin characters we have; and the hidden layers are of size 20 for the character decoding and 28 for the syllable decoding. The reason we use such a small embedding size and have single layer networks is because we do not have a lot of data.

Because this is a learning task, we need a loss function to calculate our error and backpropagate it through the network to update the network parameters. We decided to use the cross-entropy loss function for this task. This is because it allows us to have a probability calculated for each next possible pinyin character, and we want to change the weights in the network such that the probability for a given pinyin character is higher for the correct one. We propagate the loss back through the network using stochastic gradient descent with a learning rate of 0.01.

A potential problem with this method is that, because it is operating on characters rather than entire pinyin syllables, it could output character sequences that do not correspond to valid pinyin. To deal with this problem, we also create a model where, instead of finding embeddings for characters, we embed entire syllables. Practically, this is not a huge change. The inputs for our network are still the same. The output dimension is now the number of pinyin syllables we have, and the inputs to the decoder during training are full pinyin rather than individual characters.

B. Search + Deep Learning

The final approach we try is augmenting our search results with deep learning. The idea here is that search gets us pretty far along, so we should learn the corrections that would turn our search predictions into the true pinyin. To do this, we perform the same character encoding and syllable encoding tasks as described above. However, we change the structure of the prediction task so that the input to the encoder is the output pinyin of the search model for each name, and the target output is still the true pinyin. We try this approach with both character-wise and syllable-wise prediction, but it did not show better performance, as we will discuss.

C. Results

The results of the deep learning trials were somewhat disappointing. Because this is basically a classification problem, we could not run our final trained model on our entire dataset, so we used the 100 names in our evaluation set. It took approximately 8 minutes to train both networks. We present the same statistics as in the search section: average $\texttt{edit\_distance}_{pin}$ of the incorrect translations as well as the percentage of correctly translated names in table IV.

Approach	Avg $\texttt{edit\_distance}_{pin}$	% correct
Baseline	3.65	4.7
DL Char	4.06	3
DL Syll	4.08	7
Search + DL Char	4.59	0
Search + DL Syll	5.37	0

TABLE IV
SUMMARY OF RESULTS FOR DEEP LEARNING APPROACHES. CHAR AND SYLL STAND FOR CHARACTER-WISE AND SYLLABLE-WISE PREDICTION, RESPECTIVELY.

We were curious at why these deep learning methods did so poorly compared to the search ones—they were not even able to outperform the baseline. We investigated some of the translations for the names we focused on in the search section:

Benjamin ⟶ **pěn méi ěr**
Julia ⟶ **jié ěr u**

We can see that the results here are worse than the results we found in the search. For 'Benjamin', while /p/ and /b/ do share some similarities, the /j/ sound is completely absent, and there is a seemingly random **ěr** sound as well. For 'Julia', while we do have the correct number of syllables, in this case, arguably, the sounds we have are much worse as well.

The kinds of mistakes the network made in this case were different from the kinds of mistakes that the search problem made. Because we had embeddings for individual characters in the deep learning case, it was possible that some of the pinyin that were generated were not actually valid pinyin at all. We observed this result for many names, including the transliteration of 'Julia' above. The pinyin **u** does not exist and would have be preceded by a **w** for the pinyin to be valid. Below are a few other names we found this problem with:

Henry ⟶ **hăn nn**
James ⟶ **jiél s nn**

Another interesting mistake that this network makes is the use of spaces. In the character encoding network, the input pinyin that are being fed in are space separated, so the network also has the necessary data to learn the correct syllable separations. In general, this works well, as even when the pinyin themselves are incorrect, they are not separated by unnecessary spaces. Additionally, the generated spaces usually do not separate portions of pinyin that would be pinyin if there was not a space between them. Of course, there are some counterexamples to these trends as well, such as the output for 'Claudia', which produced two spaces between the last two pinyin syllables.

Claudia ⟶ **kè lì kè**

Many of these problems were solved in the deep learning with syllable embeddings approach. Because we were not generating characters one at a time, all outputs had to be valid pinyin. Because of this, we did not have any problems with spaces either. Here are some of the results:

Benjamin ⟶ **běn jié**
Julia ⟶ **zhū lì yà**
Claudia ⟶ **kè kè lǐ sī**

While some of these names look like they were translated well, particularly 'Julia', it is possible that the network was overfitting and that they appeared in the training set. Additionally, the name 'Claudia' is not well-transliterated, and these doubled syllables like **kè kè** show up somewhat frequently in mistranslations.

V. DISCUSSION

The results we obtained show an interesting pattern that is contrary to much of the NLP literature of the day.

Translating English names (or name written in Latin characters roughly following English phonetic rules) into Chinese is a difficult task. Though we were able to outperform our baseline using search techniques, we did not come close to the performance of our oracles. On average, our best performing algorithm still required more than 3 insertions, deletions and/or substitutions to arrive at the correct name. We now investigate why this is the case.

A. Task Difficulties

There are many difficulties involved with performing this transliteration. First of all, there are some English letters that can be transliterated by multiple pinyin. For instance, the 'j' in 'Julia' corresponds

to **zh**, but in 'Benjamin' it corresponds to **j**. Analogously, a single pinyin can corespond to multiple English letters. **x** is the transliteration for both the 'c' and 'th' in 'Cynthia'. Not only that, but while some pinyin characters closely correspond to English ones (such as 'p' and **p**), many others do not (like 'j' and **zh**). This is likely the most significant reason the original edit distance search performed so poorly. It encoded the flawed assumption that names in English and pinyin that look the same sound the same as well. The changes that were made to the edit distance search to replace common English characters associated with lexicographically different but phonetically similar pinyin helped to improve the performance of search, as did considering consonants that were in the same sound families as more likely candidates. The co-occurrence table search took a different approach—instead of hard-coding common English-to-pinyin substitutions, it tried to derive them from the data. This would probably have been more successful if there had been more data (see below: V-B).

Finding the best way to segment English names to assign pinyin to each segment was also a formidable challenge. Ideally, each pinyin is associated with an English syllable, but as described earlier, this is not the case. There are many instances of consonants in consonant clusters being assigned their own pinyin (such as the 'br' in 'Bruce', transliterated as **bù lǔ**), as well as single letters like 'x' sometimes being associated with the two pinyin **ke** and **si**, or the 'i' in 'Julia' factoring into the transliterations **lì** and the **yà**. If we can determine the number of pinyin a transliteration ought to have, we can more easily assign pinyin to match segmentations of those lengths while pruning out all the ones that are too short or long. This is the motivation for including syllable-counting heuristics into the search algorithms, and they do help these algorithms avoid segmentations with drastically different numbers of pinyin than are correct. For the co-occurrence table search, where we are trying to maximize the score, there is a pressure to increase the size of the segmentation. Each additional sub-word corresponds to a pinyin that contributes to the total score of the segmentation. Adding in a constraint or penalty to avoid exceeding the maximum expected number of syllables helps keep the translations from being too long.

The deep learning, encoder-decoder models took an even more relaxed approach and did not start with any initial assumptions about the segmentation of the names. They just attempted to find structure, wherever it might be. Unfortunately, the lack of a large dataset made this difficult to accomplish.

B. Data Difficulties

A major difficulty with this task is the relative lack of data. Even though English and Chinese are both high-resource languages, the number of common names with standard translations is necessarily small. There are approximately 1200 possible pinyin combinations, including tones, so our dataset of 1510 names could not possibly reliably determine what the English equivalents are in every case (even when taking into account that each name consists of multiple pinyin). This means that the syllable cost function we used in the search problems might have placed too hard a penalty on somewhat common pinyin that just did not have a strong presence in our dataset. Additionally, this means that normalizing by the frequency of the pinyin in the dataset, like we did in the co-occurrence table search, may have actually increased the probability of rare pinyin more than necessary (See figure 3).

The limited size of our dataset also might provide some insight on the differences between the various search results, particularly the discrepancy between the edit distance with phoneme-adjustment and the co-occurrence table methods. The edit distance based search with phoneme-adjustment achieved a better average $\texttt{edit\_distance}_{pin}$ for incorrect translations than the co-occurrence table, but the co-occurrence table actually translated more of the names correctly. This may be an artifact of some kind of overfitting. As mentioned in the methodology section, the number of possible English bigrams and trigrams is more than the number of pinyin in our dataset, so it is possible that the few instances of uncommon bigrams are making the translation of words with these bigrams too easy while not generalizing well enough to the rest of the names.

The biggest place where this lack of data poses a problem, however, is in the deep learning approaches. We do not really have enough data to generalize translation patterns, so the models are either going to overfit or produce meaningless results. It is a little difficult to investigate overfitting in this case. Normally, we would just compare our training error to our evaluation error, but in this case we are training and evaluating on two different (hopefully related) metrics. We are training with the differentiable softmax function and evaluating with our $\texttt{edit\_distance}_{pin}$. That said, we can calculate what our loss would be. For the seq2seq model with syllables, our final loss on our training set is approximately 1.3, but the average loss on our evaluation set is 10.5. Similarly, for the character based model, the final training loss is about 1.5, but the average loss on the evaluation set is 19.2 (See figure 2. This means that we are definitely

overfitting, which is corroborated by the almost perfect reproduction of the name 'Julia' while other names like 'Claudia' seem to be translated poorly. This overfitting is likely why the search methods were superior to the deep learning ones - they took advantage of the structure that we explicitly encoded in the problem—structure that would be found by the recurrent neural net if there was more data.

Even though there are not many more names we could use to train, there are other English loanwords in Chinese, which are words from non-Chinese languages that are transliterated and adopted into Chinese vocabulary. Examples include yoga (**yú jiā**), bikini (**bǐ jī ní**) and copy (**kǎo bèi**). These could provide some more training data, but they tend to use different sets of characters that have different meanings than characters used for names. This might skew the transliterations a little, but may be a partially viable solution.

C. Metric Difficulties

It is also possible that the metric we were scoring with, the $\texttt{edit\_distance}_{pin}$, was not fine-grained enough to capture our progress. In other words, there may have been positive things that our approaches were succeeding at that we missed. One potential issue is that this edit distance formulation does not take into account how similar two characters sound. For instance, it would penalize a **b** substituted for a **p** the same as a **f** substituted for a **p** even though **b** and **p** sound more similar (and would therefore be "more correct") than **f** and **p**. Another factor that the metric does not control for is name length. This might not be a significant problem, as names are generally around the same number of characters, but it would still be interesting to see what the per-character edit distance would be.

VI. Future Work

With regard to the search problem, we hope to take into account more features in the edit distance and unigram cost functions. First, we hope to construct family clusters of phonemes that sound similar and correspond between English and Chinese (either by hand, as in Htun et al. [9] and Fung et al. [10], or via another learning or clustering technique, as in Li et al. [11]). For example, $\{\mathbf{zh}, \mathbf{z}, \mathbf{j}\}$ may be a phoneme cluster in Chinese, while the similar-sounding phonemes $\{j, z, g\}$ might be a cluster in English. Between the two languages, we can link these phonemes together as phonemes that sound similar. Using these phoneme clusters, we can then derive a metric for the phonetic distance between two given phonemes, where one is in Chinese and one is in English. We can incorporate this metric into our edit distance calculation so that the edit distance between **z** and 'c' may be considered smaller than that between **z** and 'p', for instance. This would likely improve the cost function for conversion of English syllables to Chinese.

Second, we also hope to implement a phoneme alignment algorithm, in the spirit of that in Fung et al. [10], whereby the English and proposed Chinese names are broken up into respective phonemes that align, or correspond to one another with respect to the transliteration. For example, the transliteration Julia → **zhū lì yà** would have the alignment:

J	u	l	i	a
Zh	**ū**	**l**	**ì**	**yà**

Once this alignment is implemented, we can then use it in tandem with the phoneme family clusters to calculate a more accurate edit distance between English and Chinese representations. We can also use the segmentations of the alignments to better determine segmentations for which to calculate co-occurrences in the n-grams-based approach.

Third, we hope to take some inspiration from the bootstrapping for low-resource languages technique [4] to be able to perform transliteration. Despite neither English nor Chinese being a low-resource language, common names as a sub-category are relatively limited in number, and we could benefit from applying the techniques they discuss in their work.

VII. Conclusion

In this project, we investigated the challenge of finding Chinese transliterations of English names. While we were able to perform better than our baseline, which involved a dictionary look-up with some segmentation rules, we were not able to achieve human-level performance. We approached this task with both search and deep-learning based methods and found that, likely due to our limited dataset, search performed better. Transliteration of names is a difficult problem, and one that is not frequently studied as much as other NLP tasks. We hope this work provides a good starting point for future work in this area.

References

[1] K. Kaur and P. Singh, "Review of machine transliteration techniques," *International Journal of Computer Applications*, vol. 107, no. 20, 2014.

[2] S. Wan and C. M. Verspoor, "Automatic english-chinese name transliteration for development of multilingual resources," in *Proceedings of the 17th international conference on Computational linguistics-Volume 2*, Association for Computational Linguistics, 1998, pp. 1352–1356.

[3] Y. Shao, J. Tiedemann, and J. Nivre, "Boosting english-chinese machine transliteration via high quality alignment and multilingual resources," in *Proceedings of the Fifth Named Entity Workshop*, 2015, pp. 56–60.

[4] S. Upadhyay, J. Kodner, and D. Roth, "Bootstrapping transliteration with constrained discovery for low-resource languages," *arXiv preprint arXiv:1809.07807*, 2018.

[5] L. Mack, *Ever wonder what your name translates to in chinese?* https://www.thoughtco.com/chinese-and-english-names-688196, Accessed: 2018-12-09.

[6] GLOSBE Partners, *Transliteration and romanization utilities*, https://glosbe.com/, 2017.

[7] K. Lenzo, *Cmu pronounciation dictionary*, http://www.speech.cs.cmu.edu/cgi-bin/cmudict, 2015.

[8] Kozea Community, *Pyphen*, http://pyphen.org, 2018.

[9] O. Htun, S. Kodama, and Y. Mikami, "Cross-language phonetic similarity measure on terms appeared in asian languages," *International Journal of Intelligent Information Processing*, vol. Volume 2, p. 9 21, Jun. 2011. DOI: 10.4156/ijiip.vol2.issue2.2.

[10] P. Fung, W. Byrne, F. Zheng, T. Kamm, Y. Liu, Z. Song, V. Venkataramani, and U. Ruhi, "Pronunciation modeling of mandarin casual speech," Johns Hopkins University, 2000, pp. 1–45. [Online]. Available: https://pdfs.semanticscholar.org/4a52/fc064e45b5b3b4df535c9c13ee88b843e3d1.pdf.

[11] M. Li, M. Danilevsky, S. Noeman, and Y. Li, "Dimsim: An accurate chinese phonetic similarity algorithm based on learned high dimensional encoding," in *Proceedings of the 22nd Conference on Computational Natural Language Learning*, Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 444–453. [Online]. Available: http://aclweb.org/anthology/K18-1043.

Appendix

List of names presented to baseline and oracle: Golrokh, Anastasia, Benjamin, Krystal, Alice, Greses, Krishna, Jake, Siena, Mateo, Yatharth, Aiden, Jocelyn, Mehran, Jenny, Olen, Thariq, Timothy, Julia, Abhishek, Mary, Sanja, Irvin, Carly, Valerio, Kenneth, Marty, Judith, Murali, Robert

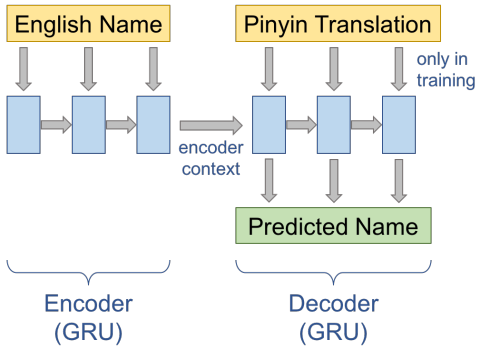

Fig. 1. Illustration of our encoder-decoder framework

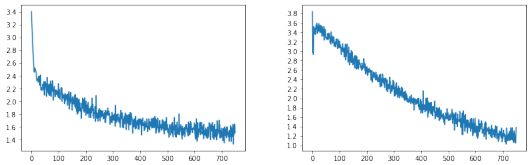

Fig. 2. On the left we have the loss curves from the character embedding seq2seq model and on the right we have the syllable embedding one. Graphs of Loss versus Training Iteration

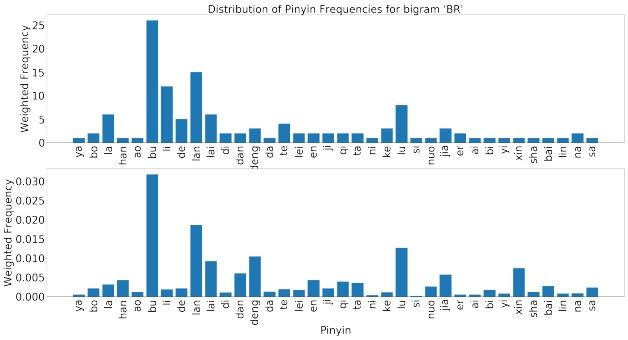

Fig. 3. Distribution over (toneless) pinyin frequencies associated with bigram 'br' unnormalized on top and normalized by pinyin frequency on bottom. Notice how the relatively uncommon pinyin **xin**'s mass increases dramatically with normalization.