Improving Back-Transliteration by Combining Information Sources

Slaven Bilac & Hozumi Tanaka
Department of Computer Science
Tokyo Institute of Technology, Tokyo
{sbilac,tanaka}@cl.cs.titech.ac.jp

Abstract

Transliterating words and names from one language to another is a frequent and highly productive phenomenon. Transliteration is information loosing since important distinctions are not preserved in the process. Hence, automatically converting transliterated words back into their original form is a real challenge. However, due to wide applicability in MT and CLIR, it is a computationally interesting problem. Previously proposed back-transliteration methods are based either on phoneme modeling or grapheme modeling across languages. In this paper, we propose a new method, combining the two models in order to enhance the back–transliterations of words transliterated in Japanese. Our experiments show that the resulting system outperforms single-model systems.

1 Introduction

With the advent of technology and increased flow of goods and services, it has become quite common to integrate new words from one language to another. Whenever a word is adopted into a new language, pronunciation is adjusted to suit its phonetic inventory. Furthermore, the orthographic form of the word is modified to allow representation in the target language script. This process of acquisition and assimilation of a new word into an existing writing system is referred to as transliteration (Knight and Graehl, 1998).

Since integration of new words is a very productive process, it often happens that the new pairs are not recorded in machine or human dictionaries. Therefore, it is impossible to rely on the dictionary lookup to find the transliteration pairs. Failure to find a target language equivalent represents a big problem in Machine Translation (MT) where failures in dictionary lookups can cause translation failures. Furthermore, transliteration represents a significant problem in the field of Cross-Language Information Retrieval (CLIR) where the goal is to retrieve all the related documents in two or more languages (Lin and Chen, 2002). In many cases the results would be greatly improved, if the system were able to correctly identify the English equivalent of a Japanese or Korean transliteration and then search for documents based on the original word.

When the source language and the target language use the same (or very similar) alphabet, there are hardly any problems, as the speakers of both languages can easily identify the string. On the other hand, if two languages use very different writing codes, the acquired word undergoes heavy transformation in order to become an acceptable vocabulary entry. For example, the English word *cache* is transliterated in Japanese as キャッシュ "kyasshu".[1]

[1] We use *italics* to transcribe the English words, while Japanese transliterations (e.g. キャッシュ) are given with romaji in "typewriter" font (e.g. "kyasshu"). The romaji used follows (Knight and Graehl, 1998), thus closely reflecting English–like pronunciation with long vowels transcribed as "aa" rather than "ā".

Although governments provide guidelines on how to transliterate, words commonly appear in several different forms. The wide variety of transliterations can be seen both in Korean (Jeong et al., 1999) and Japanese (Knight and Graehl, 1998; Brill et al., 2001). For example, the English word *interface* has five different transliterations in EDICT Japanese-English dictionary:[2] インターフェース "intaafeesu", インターフェイス "intaafeisu", インタフェース "intafeesu", インタフェイス "intafeisu" and インタフェス "intafesu".

While automatic transliteration in itself is difficult, back-transliteration or transliteration back into the original language is even harder. Increase in difficulty results from the fact that various distinctions, present in the source language, are not preserved when the word is transliterated into the target language. For example, Japanese has only five basic vowels and no /θ/ or /ð/[3] sounds, whereas nonexistent sounds are replaced with the closest equivalents. Consequently, the following three English words: *bass*,*bath* and *bus* are transliterated as バス "basu".[4] The system trying to obtain a back-transliteration for バス has therefore three valid choices which cannot be disambiguated in the absence of additional contextual transformation.

Transliterated words are normally written in katakana, one of three Japanese writing systems. While other vocabulary (i.e. animal names or onomatopoeic expressions) can also be written in katakana, the fact that something is written in katakana is a good hint that it might be a transliterated foreign word or a name. Thus, unlike Arabic or Korean, where a big part of the back-transliteration problem is identifying candidate transliterations (Stalls and Knight, 1998; Jeong et al., 1999), in Japanese back-transliteration can be directly applied to any katakana strings absent from the bilingual dictionary.

In this paper we propose a method to improve back-transliteration by combining the information based on pronunciation and spelling. Even though we concentrate on Japanese and English, our method is applicable to other language pairs.

[2] ftp://ftp.cc.monash.edu.au/pub/nihongo/

[3] All phonemes given in // are written in IPA symbols.

[4] Here /θ/ is replaced with /s/, and /æ/ is replaced with /a/.

The reminder of this paper is organized as follows: in Section 2 we review previous approaches to (back-)transliteration. In Section 3 we describe the proposed method and outline the implementation details. Finally, Section 4 gives a short evaluation and a discussion or results obtained.

2 Previous research

Previous approaches to (back-)transliteration can be roughly divided into two groups: grapheme- and phoneme-based. These approaches are also referred to as direct- and pivot-based methods, respectively.

2.1 Grapheme-based modelling

In this framework, the English string is not converted into a phonemic representation before its alignment with the transliterated string. Brill et al. (2001) propose a noisy channel model for Japanese. This model allows for non-atomic edits: several letters can be replaced by a different letter combination (Brill and Moore, 2000). The input string is broken down into arbitrary substrings, each of which is output independently (and possibly incorrectly). The model is trained to learn edit-probabilities and the best back-transliteration is chosen using a modified edit distance algorithm (Damerau, 1964; Levensthein, 1966). This method fails to generate the correct string in cases where English spelling is not reflected in the pronunciation (e.g マイム "maimu" being incorrectly back-transliterated into *maim* instead of *mime*).

For transliteration, Goto et al. (2003) propose a maximum entropy based model for Japanese, while Kang and Choi (2000) propose a decision tree-based model for Korean.

2.2 Phoneme-based modelling

The systems based on phoneme alignment are more numerous. Jeong et al. (1999) propose a method using first order HMM model to generate English strings from Korean input. The result is compared with dictionary entries using a variety of string similarity algorithms to find the best match.

For Japanese, Knight and Graehl (1998) employ a compositional model combining romaji-to-phoneme, phoneme-to-English and English word probability models into one. The combined structure is treated as a graph, and the top ranking strings

are found using the *k-best* path algorithm (Eppstein, 1994). A similar model has been applied for Arabic-English back-transliteration (Stalls and Knight, 1998). However, this model cannot handle cases where the transliteration reflects the original spelling. For example, *tonya* and *tanya* have different transliterations of "toonya" and "taanya" but the system taking only pronunciation into account is unable to distinguish between the two.

Finally, Oh and Choi (2002) propose a system trying to incorporate two different English-to-phoneme models into a single Korean transliteration system: standard English pronunciation and pronunciation closely following the spelling. However, this system uses only the origin of the word (estimated by a match against a finite set of affixes) to decide which model to apply when producing the transliteration.

3 The Combined model

The systems introduced in the previous section model transliteration based on either phoneme or grapheme level. Nonetheless, even though most of the transliterations are based on the original pronunciation, there is a significant number of words where transliteration corresponds more closely to the spelling of the original. For example the first *e* in *eternal* is transliterated as "e" instead of "i" in エターナル "etaanaru". *phantom* is transliterated as ファントム "fantomu" rather than "fentamu"*. We believe, that we can better account for such behavior by combining the two information sources to maximize the use of the data available.

3.1 Probabilistic model specification

Given the Japanese word in romaji (i.e. alphabet),[5] the goal is to produce an English word (phrase) that maximizes the probability $P(E_a|J_a)$. Applying the Bayes' rule and dropping the constant denominator we get $P(J_a|E_a) \times P(E_a)$ where $P(E_a)$ is the source model and $P(J_a|E_a)$ is the noisy channel. We train the channel model as described below, and then reverse it to handle the romaji input.

3.1.1 Grapheme-based model (GM)

In this model the English word is directly rewritten as a Japanese romaji string with probability $P_g(J_a|E_a)$. Here, we follow (Brill et al., 2001) to arbitrarily break up the E_a string into n parts and output each part independently. Thus, the resulting probability of outputting J_a can be rewritten as in the equation (1).

$$P_g(J_a|E_a) \cong \prod_{i=1}^{n} P_g(J_{a_i}|E_{a_i}) \quad (1)$$

We implement $P_g(J_a|E_a)$ as a weighted Finite State Transducer (WFST) with E_{a_i} as inputs, J_{a_i} as outputs (Knight and Graehl, 1998; Pereira and Riley, 1997) and transition costs as negative logs of probabilities. This WFST is then reversed and the best transliteration is computed as its composition with the source model $P(E_a)$.[6] The resulting WFST is searched for k-best transliterations using the k-best path algorithm. A probability $P_g(E_a|J_a)$ is associated with each path obtained.

3.1.2 Phoneme-based model (PM)

In this model the channel is broken into two stages: a) conversion of the English alphabet into English phonemes with some probability $P(E_p|E_a)$ and b) conversion of the English phonemes into romaji with some probability $P(J_a|E_p)$. Consequently, $P_p(J_a|E_a)$ can be rewritten as equation (2). Rather than manipulating these two distributions separately, we compute their composition to obtain a unique probability distribution $P_p(J_{a_i}|E_{a_i})$.

$$P_p(J_a|E_a) \cong \prod_{i=1}^{n} P(J_{a_i}|E_{p_i}) \times \prod_{i=1}^{n} P(E_{p_i}|E_{a_i}) \quad (2)$$

Consequently all English alphabet strings can be rewritten directly into romaji without requiring their conversion into intermediate phoneme representation. This removes the requirement of having a pronunciation dictionary for the back-transliteration.[7]

[5] As stated above, transliterated words are normally written in katakana, potentially inducing an another stage in the model: rewriting romaji characters into katakana $P(J_k|J_a)$. However, katakana characters generally have a unique alphabetic equivalent, thus reducing this distribution to 1. We implement the katakana to romaji conversion as a preprocessing module.

[6] We use the AT&T FSM library (http://www.research.att.com/~mohri/fsm/) for WFST composition.

[7] However, the pronunciation dictionary is still necessary for the training.

Furthermore, since both models are dealing with the same unit types, it is possible to directly combine them, allowing for certain parts of the input string to be converted by one and the rest by the other model. We leave this method of combination for future research.

3.1.3 Combining the models

After obtaining the back–transliterations $E_{a_{phon}}$ and $E_{a_{graph}}$ with the respective probabilities of $P_p(E_a|J_a)$ and $P_g(E_a|J_a)$, we can assign the final score of a transliteration $S_c(E_a|J_a)$ as in equation (3) where γ and δ are set to maximize the accuracy on the training set.[8] Transliteration with the highest score is selected as the best.

$$\begin{aligned} S_c(E_a|J_a) &= \gamma P_p(E_a|J_a) + \delta P_g(E_a|J_a) \\ &\text{s.t.} \quad \gamma + \delta = 1 \end{aligned} \tag{3}$$

3.2 Training the models

For the GM, we follow (Brill et al., 2001) closely to extract the character-string mappings. Since romaji and English alphabet are equivalent character sets, they can be aligned using the non-weighted Levensthein distance. Then, letter-edits are expanded to include up to N edits to the right and to the left. For example for the pair (roo,*row*) we get: $r \rightarrow r \quad o \rightarrow o \quad o \rightarrow w$. For $N = 1$, edits $ro \rightarrow ro$, $roo \rightarrow row$, $oo \rightarrow ow$ are also added to the set. We collect a complete set of edits $\alpha_g \rightarrow \beta_g$ in the training set and assign the probability to each according to equation (4). Throughout, we distinguish edits that appear at the beginning or the end of the word or neither.

$$P(\alpha \rightarrow \beta) = \frac{count(\alpha \rightarrow \beta)}{count(\alpha)} \tag{4}$$

Given the collection of edits $\alpha_g \rightarrow \beta_g$ for each input word J_a we can generate a WFST which contains all possible ways to rewrite the input string.

3.2.1 Training the phoneme model

For the PM, the English phoneme and English alphabet sets are not equivalent, hence the edit-distance algorithm cannot be applied directly to obtain the optimal alignment. Instead we proceed to obtain the best alignment using the EM algorithm (Dempster et al., 1977). Given the input strings, we generate all possible alignments constrained so that: a) each unit in one string aligns to one or more units in the other string and b) there are no crossing alignment arcs. Here the base unit represents either a letter or a phoneme.[9]

After the EM algorithm selects the optimal alignment, we proceed to expand the set of individual alignments with N adjacent units as above to obtain a set of possible rewrites $\alpha_{e_a} \rightarrow \beta_{e_p}$. This process is repeated to obtain the set of all possible rewrites of English phonemes into romaji $\alpha_{e_p} \rightarrow \beta_{j_a}$.

Each input α_{e_a} with all its mappings β_{e_p} is converted into a WFST and composed with a WFST encoding the complete set of mappings $\alpha_{e_p} \rightarrow \beta_{j_a}$ to obtain the set of all possible rewrites of English alphabet strings α_p into romaji strings β_p based on the PM.

For the case ($N = 0$), the model for mapping romaji to English phonemes is similar to the one described by Knight and Graehl (1998). However, we learn the alignments both for English alphabet to English phoneme strings and English phoneme to romaji strings, add context information and compose the resulting models to get direct mappings from English alphabet to romaji. We will see the benefits of these improvements in the following section.

4 Evaluation

We extracted a collection of about 7000 words in katakana together with the corresponding English translation from the EDICT dictionary. About 10% (714 tokens) of these entries were left out for evaluation. The remaining set was expanded, so that for each katakana word containing a long vowel or a geminate consonant, we add one with these removed. The pronunciations for training the PM were obtained from the CMU pronouncing dictionary. When no pronunciations were available the words were excluded from the training.

Table 1 gives the result of our experiments with 714 EDICT transliterations for the Phoneme Model without context (PM0), the Grapheme Model (GM), the Phoneme Model (PM) and the combined model

[8] Parameters are trained using Golden Section Search (Press et al., 1992).

[9] The CMU pronouncing dictionary (http://www.speech.cs.cmu.edu/cgi-bin/cmudict) phoneme set is used for a total of 39 phonemes without the tone marks.

		EDICTa		EDICTb	
	Inputs	Top-1 (%)	Top-10 (%)	Top-1 (%)	Top-10 (%)
PM0	714	281 (39.36)	368 (51.54)	232 (32.49)	365 (51.12)
GM	714	473 (66.25)	595 (83.33)	455 (63.73)	591 (82.77)
PM	714	571 (79.97)	664 (93.00)	484 (67.79)	623 (87.25)
COMB	714	604 (**84.59**)	698 (**97.76**)	504 (**70.59**)	649 (**90.90**)

Table 1: Transliteration results for the EDICT test set

		CMUa		CMUb	
	Inputs	Top-1 (%)	Top-10 (%)	Top-1 (%)	Top-10 (%)
PM0	150	27 (18.00)	47 (31.33)	20 (13.33)	36 (24.00)
GM	150	49 (32.67)	86 (57.33)	69 (46.00)	96 (64.00)
PM	150	58 (**38.67**)	82 (54.67)	67 (44.67)	91 (60.67)
COMB	150	57 (38.00)	106 (**70.67**)	70 (**46.67**)	107 (**71.33**)

Table 2: Transliteration results for the EDR test set

(COMB). Here, PM0 was trained only on the directly aligning edits (N=0), and the remaining models used a context of two units to the left and to the right ($N = 2$). The test dictionary contains all words appearing in the English translations in the EDICT dictionary (over 30,000 words). The top-1 and top-10 accuracies are given for two language models (LM): EDICTa where all words have equal probability and EDICTb where probabilities reflect the corpus frequencies from the EDR English corpus (EDR, 1995). The transliterations were considered correct, if they matched the English translation, letter-for-letter, in a non-case-sensitive manner.

We can see that the PM yields better results than the GM with the same context window. This justifies the consideration of pronunciation for transliteration, and it shows that our method of mapping English to romaji using pronunciation is effective. Furthermore, we can see that the proposed method (COMB) gives the best performance in all cases.

It might seem surprising that using EDICTb results in reduced accuracy. However, the corpus frequencies bias the model so erroneous transliterations consisting of shorter more frequent words are preferred over longer, correct, but infrequent words. This shows the importance of a good LM from which to select transliterations.

For the second set of experiments we extracted 150 katakana words from the EDR Japanese corpus not in the EDICT dictionary and we used the complete CMU dictionary word set (around 120,000 words) compiled into models CMUa and CMUb, as described above.

Table 2 gives the transliteration results for this test set. We can see a significant overall drop in accuracy. It is partially due to a larger set of words to choose from, hence a more difficult task. Since various spellings with similar pronunciations are contained in the dictionary, corpus frequencies help improve the top-1 accuracy, thus the higher accuracy rates for the CMUb language model. For example, with CMUb *service* is selected rather than *servis* as the top transliteration of "`saabisu`" in センター サービス "`centaasaabisu`" *center service*.

However, a bigger problem is the inability of our system to handle non-English terms (e.g. サハリン "`saharin`" *Sakhalin*) and abbreviations (e.g. リハビリテーションセンター "`rihabiriteeshon-sentaa`" *rehabilitation center* is abbreviated as リハビリセンター "`rihabirisentaa`") which make a sizable portion of EDR out-of-vocabulary items. Rather than trying to obtain an English equivalent of these terms, the system would ideally be able to determine the possible origin of the word from the context available (e.g. user query in CLIR) and then apply an adequate language model.

Brill et al. (2001) provide no direct evaluation of their transliteration system. Instead, they evaluate the ability of their system to extract English-katakana pairs from non-aligned web query logs. On

the other hand, Knight and Graehl (1998) give only the accuracy for transliteration of personal names (64% correct, 12% phonetically equivalent) but not for general out-of-vocabulary terms. This makes comparison with our system difficult. Nonetheless, the above experiments show that the combination of the phoneme- and grapheme-based models helps the overall accuracy and coverage. In the future, we would like to explore different ways of combining these models to further increase the positive effect of the combination.

5 Conclusion

Back transliteration is the process of converting transliterated words back into their original form. Previous models used either only phoneme- or only grapheme-based information contained in the transliteration. Instead, we propose a method for improving back–transliteration by combining these two models. We go on to describe how we implemented the models to allow combination and finally, we evaluate the effectiveness of the combined model and point out some deficiencies we hope to address in the future.

Acknowledgements

We would like to thank Taiichi Hashimoto, Michael Zock and three anonymous reviewers on valuable comments and help in writing this paper.

References

E. Brill and R. C. Moore. 2000. An improved error model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting of the Association for Computarional Linguistics (ACL 2000)*, pages 286–293.

E. Brill, G. Kacmarcik, and C. Brockett. 2001. Automatically harvesting katakana-English term pairs from search engine query logs. In *Proc. of the Sixth Natural Language Processing Pacific Rim Symposium*, pages 393–399.

F. Damerau. 1964. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7:659–664.

A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete via the em algorithm. *Journal of the Royal Statistical Society*, 39:1–38.

EDR. 1995. *EDR Electronic Dictionary Technical Guide*. Japan Electronic Dictionary Research Institute, Ltd. (In Japanese).

D. Eppstein. 1994. Finding the k shortest paths. In *In Proc. of the 35th Symposium on the Foundations of Computer Science*, pages 154–165.

I. Goto, N. Kato, N. Uratani, and T. Ehara. 2003. Transliteration considering context information based on the maximum entropy method. In *Proc. of IXth MT Summit*.

K. S. Jeong, S. H. Myaeng, J. S. Lee, and K. S. Choi. 1999. Automatic identification and back-transliteration of foreign words for information retrieval. *Information Processing and Management*, 35:523–540.

B. J. Kang and K. S. Choi. 2000. Automatic transliteration and back-transliteration by decision tree learning. In *Proc. of the 2nd International Conference on Language Resources and Evaluation (LREC 2000)*.

K. Knight and J. Graehl. 1998. Machine transliteration. *Computational Linguistics*, 24:599–612.

V. Levensthein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics–Doklady*, 10:707–710.

W. H. Lin and H. H. Chen. 2002. Backward machine transliteration by learning phonetic similarity. In *Proc. of the Sixth Conference on Natural Language Learning*, pages 139–145.

J. H. Oh and K. S. Choi. 2002. An English-Korean transliteration model using pronunciation and contextual rules. In *Proc. of the 19th International Conference on Computational Linguistics (COLING 2002)*, pages 393–399.

F. C. N. Pereira and M. Riley. 1997. Speech recognition by composition of weighted finite automata. In E. Roche and Y. Shabes, editors, *Finite-State Language Processing*, pages 431–453. MIT Press.

W. H. Press, B. P. Flannery, A. Teukolsky, and T. Vetterling. 1992. *Numeric Recipies in C*. Cambridge University Press, 2nd edition.

B. G. Stalls and K. Knight. 1998. Translating names and technical terms in Arabic text. In *Proc. of the COLING/ACL Workshop on Computational Approaches to Semitic Languages*.