Indian Language Transliteration using Deep Learning

Akshat Joshi[*], Kinal Mehta[*], Neha Gupta[*], Varun Kannadi Valloli[*]

[*] All authors contributed equally

Centre for Development of Advanced Computing (C-DAC), GIST Group, Pune, India

{akshatj, kinalm, nehag, varunkv}@cdac.in

***Abstract*—Transliteration of Indian languages has been a challenging problem, given their complex nature, which has conventionally been handled by rule based systems developed by trained linguists. Scalability is a prominent issue with these systems and given the number of language pairs possible for Indian languages, a scalable pipeline is necessary for swift development of these systems. Deep learning systems are pragmatic for building a scalable pipeline, as they are completely data driven. We experimented with LSTMs and Sequence to Sequence models to find an optimal model for the scalable pipeline by comparing the results. The results show Sequence to Sequence models are a better fit for this solution. We also discuss techniques for pre-processing the data and post processing the output for optimal performance.**

Keywords— LSTM, Seq2Seq, Transliteration, NLP, Decoder, Encoder, Akshar

I. Introduction

Transliteration is the process of transforming words written in one script to another script without losing their phonological characteristics. It enables readers familiar with the one script (target script) to read the text written in the other script (source script). This technique is widely employed in Machine Translation systems for converting Named Entities to their desired form in the target language in a different script. Transliteration now-a-days is also heavily used in Transliterated inputting, i.e., facilitating users to use the keyboard of one script to input languages represented by a different script. If a particular text belongs to a dual script language like Sindhi which is written in both Devanagari and Perso-Arabic script, the effect of transliteration is as good as that of a translation.

Our main focus was on English↔Indian Language and Indian Language↔Indian Language transliteration system using deep learning. Traditionally this problem has always been solved with the help of rule-based systems [1][2] which were mostly hand-crafted by linguists. The shortcomings of these systems are, the linguists are expected to have good command over both the source and target languages and scripts, and such skilled resources, cannot be expected to be available for all possible language pairs. The rule-based systems come with many drawbacks like unscalability, inflexibility and being limited in terms of handling all possibilities in the solution space. In contrast to the rule-based systems, deep learning-based systems only need data and the compute capability which are abundantly available these days [3]. Hence, we decided to design a transliteration system that would leverage this and provide a scalable solution towards this problem.

We experimented with Sequence to Sequence [4] and LSTM [5] models for building the transliteration systems. We evaluated the performance of these models not only on raw parallel corpus, but also after preprocessing the parallel corpus with techniques like Epsilon insertion [6] and reordering of the input sequences.

The problem is complicated as different basic alphabets and phonetic systems are used by different languages [7]. Moreover, lengths of the input and output sequences also vary. The parallel corpus can have entries with ambiguous mappings, making the problem more subjective to such an extent that evaluation of such systems becomes very challenging. This paper discusses the experiments and results in our attempts at transliteration using Deep Learning.

The rest of our paper is organized as follows. We address our motivation in section II. Section III discusses our datasets and their descriptions. The language specific challenges are discussed in section IV followed by the experiments and results in section V. Section VI describes the patterns observed in the outputs which could be used for post processing and finally we conclude in section VII.

II. Motivation

India is a multilingual country with vastly different cultures. Most of the cultures have their own languages which are in some way unique and fundamentally different from one another [14]. The administrative regions (States) in India, by and large, were created on the basis of linguistic division. This linguistic division paved a path for very robust linguistic preservation of Indian languages with the State support. Hence, many of these regions have a very prominent presence of local languages in communication they have with the populace. With vast amounts of cross-cultural exchanges taking place, transliteration takes a centre-stage in information exchange.

Traditional Transliteration systems were mostly rule-based. Even though they are quite accurate, their performance heavily depends on the linguist's domain knowledge and adeptness. Scaling-up of such systems for a large number of mainstream and obscure native languages was not easy due to the human factor involved. Even if all the known rules are put in place, it still leaves out some cases which require handling on case to case basis. Conventional machine learning techniques like Support Vector Machines [8], Hidden Markov Models [9], Decision trees [10] etc. are also unscalable as they require feature engineering. This is where deep learning comes into play. Building a data driven model for transliteration helps not only in scalability but also in handling certain corner cases which could be dealt with from the data itself. The transliteration data in most of the cases is not completely consistent as transliteration is a highly subjective problem. The ambiguities in the data can degrade the model's performance, but deep learning is robust to noise, the caveat being that it is data hungry.

III. Datasets

We evaluate our models on Hindi (Devanagari)↔English (Latin), Sindhi (Perso-Arabic)↔Sindhi (Devanagari) and Kannada (Kannada) to English (Latin) transliteration. The datasets we used are described in Table 1. The datasets are in a parallel corpus format, i.e., source-word"\t"target-word. For Hindi↔English transliteration, we used a list of 200k proper nouns augmented by another 40k loan words. The Sindhi (Perso-Arabic)↔Sindhi (Devanagari) and Kannada (Kannada) to English (Latin) had a similar composition. For the seq2seq models we trained, the dataset was not pre-processed, however for LSTM models using Epsilon Insertion(EI)[6], we inserted '-' for alignment of characters, e.g., 'gha' was mapped to '--घ'. The problems we encountered in EI, which are language specific, are discussed in the challenges section.

TABLE 1: DATASET DESCRIPTION

Language (script1↔ script2)	**Dataset size**	**Avg. word length script1**	**Avg. word length script2**	**Vocab script1**	**Vocab script2**
Sindhi (Devanagri ↔ Arabic)	363231	6.26	6.25	69	76
Kannada (Kannada-Latin)	421128	7.38	8.23	89	26
Hindi (Devanagri ↔ Latin)	246024	8	9	70	26

IV. Challenges

From prior discussions, we know that transliteration is not a trivial problem. Building a generic model and expecting it to portray the same behaviour with different language pairs is naive. Language specific nuances introduce a set of challenges which causes the models to behave differently for different language pairs. This section will discuss a few prominent issues we faced with Hindi, Sindhi and Kannada.

A. *Hindi*

Hindi is a language primarily written in Devanagari script which is an alphasyllabary derived from the ancient Brahmi script. Apart from having regular character categories like Consonant and Vowel, it also has some special characters which are of dependent nature, i.e., they get attached to their preceding character, e.g., Vowel signs, Anusvara etc. This makes Hindi a complex language. Apart from that, all the Indian languages have a special character to form the conjuncts, called as *Halanta*. This character kills the inherent vowel in the preceding consonant thereby forming the conjunct. In addition, the Halanta behaves in a peculiar manner which is not observed for any character in any of the non-brahmi derived scripts. It not only maps to *nothing* in the target script, but also introduces variations in the way the other characters are to be depicted in the target script.

B. *Sindhi*

Sindhi is one those Indian languages which is prominently written in two totally different scripts. One is Devanagari and the other is Perso-Arabic. These scripts not only greatly differ in their character set but also belong to different writing systems.

Sindhi when written in Devanagari is written in a very definitive manner and there are hardly any ambiguities in terms of the pronunciation of the words. However, the same when written using Perso-Arabic script, there is a lot of contextual information involved and many of the intended sounds are not directly available in the form of actual characters. Those sounds are to be deciphered by the reader, keeping in mind the context of the word. The same word can be read differently in different contexts, e.g., the word
شهزادي - can be read as शहज़ादी /shehzadi/ as well as शहज़ादे /shehzade/ depending on the context of the word.

C. *Kannada*

Kannada is one of the Indian languages that belongs to the Dravidian family. One of the major differences between Indo-Aryan and the Dravidian languages is that of pronunciation of the implicit vowel at the end of the word. In Hindi, it is \a\ whereas in Kannada, it is \aa\. However, depending on the word origin, this implicit vowel varies thereby changing the expected transliteration of the words.

Another interesting aspect of the Kannada is the transliteration of the characters ತ \ta\ and ಥ \tha\. While character "ಥ" is consistently mapped to its intended sound, i.e., \tha\, the character ತ also expected to map to \tha\ in some cases and this behaviour is observed to be inconsistent and yet transliteration systems are supposed to handle such deviations.

V. Experiments and Approaches

We tried out two approaches, first using the LSTM and the other one using Sequence to Sequence. For each approach, the input feed method is chosen first and the input sequence is accordingly reordered and one-hot encoded. The maximum length of the input is fixed to 36 which was the length of the longest word in the datasets. Both the approaches have been discussed below in detail.

A. *LSTM:*

LSTM networks are capable of modelling sequential aspects of data and hence have been used widely for Natural language processing problems. Our model is fed a single source character and generates a single target character at a time. For our experiments, we use bidirectional-LSTM [11] which has two layers; one accesses information in forward direction and other in the reverse direction. These networks have access to the past as well as the future information and hence the output is generated from both the past and future contexts. This network tries to learn one-to-one mapping given the neighbouring characters. Since it tries to learn one-to-one mappings, the source and target words need to be well aligned for optimal performance. We have experimented with EI and our extension of it, which aligns the words on the basis of the constituent *akshars* [12]. An *akshar* is a syllable inherent to the Brahmi based languages. An *akshar* comprises a nodal consonant or vowel to which a character or a sequence of characters such as Halant, Vowel Sign (Matra) or Diacritic may be adjunct based on specific rules.

Thus, *akshar* forms a basic building unit for all Brahmi based languages.

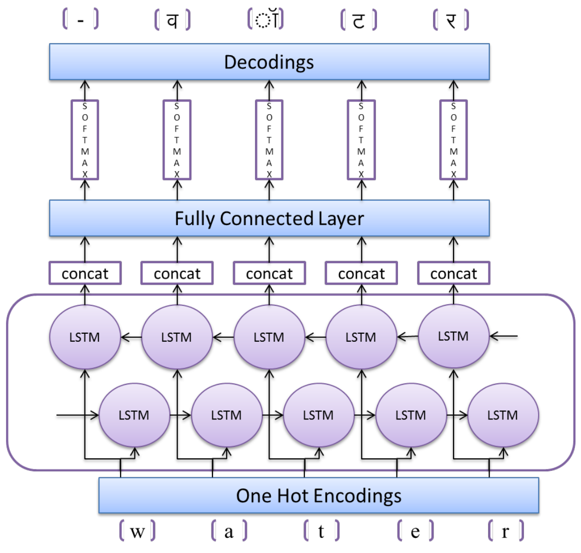

Fig. 1. Bidirectional LSTM networks.

1) Epsilon insertion based on word length match

We used Epsilon insertion for aligning the source and target words in the dataset. Epsilon, which we represent with '-', is inserted as a placeholder in the target script for alignment, e.g., 'ghar' was mapped to '--घर'. This is only possible when the input is longer than the output, i.e., there is either a one-to-one mapping or a many-to-one mapping between the source and target language characters. The problem with one-to-many relation between the source and target language characters is that, it requires EI in the source words, which would not be possible during the prediction because our implementation of EI requires both input and output beforehand, if one-to-many relation exists, for successfully aligning the words.

2) Epsilon insertion based on akshar length match:

In this case we split the input and output by akshars, in case of Brahmi derived languages, and by equivalent set of characters (referred here as *syllables*) in case of non-brahmi derived languages, and then make the respective syllables/akshars of same length, by inserting the epsilon character '-' wherever required. This ends up making the source and target words of the same length.

E.g., as seen in the following table, there are two akshars in source word "गुप्ता" and respectively two syllables in the target word "gupta". The EI can be seen in the second syllable where its equivalent akshar has four characters while the syllable has only three.

TABLE 2: SYLLABLE EPSILON INSERTION

Source akshars	ग ु	प ् त ा
Target syllables	g u	- p t a

This method should theoretically perform better as each akshar/syllable in the source language has a corresponding unique akshar/syllable in the target language. The model gets to learn these mappings between the syllables, as opposed to characters, thereby minimizing the ambiguity. This enhances the previous raw input to a refined state, i.e., after processing the input data with this method, it does hold information.

This is only possible when the individual syllables in the target script are shorter in length than the syllables in the source script, as we won't be able to determine where to insert the epsilons in the source script during the prediction time, without the knowledge of the target script syllables. This problem is described in Table 3.

TABLE 3: SHORTER SOURCE AKSHAR

प ् र े	म	-व	त ी
-pre	m	va	ti

In this case, both the source and target words need epsilon insertion. This is possible during training time as we have both the input and output words which are necessary for our alignment algorithm. However, during prediction, it is not possible to align the akshars/syllables and the system falters.

3) Syllable split:

The character by character feeding approach suffers heavily due to alignment mismatch in terms of phonetic syllables in source and target languages. This approach tries to overcome that shortcoming by introducing the length matching at the phonetic syllable level thereby trying to bring in a better context. Eventually we one-hot encoded the syllables.

As compared to basic alphabets, the number of unique syllables is far greater in number. This increases the problem complexity which not only demands relatively much larger corpus but also the compute capability. Moreover, larger one-hot vectors also increase the memory footprint. Due to these reasons the method is not scalable, and as expected, the results were poor.

B. Sequence-to-sequence:

Sequence-to-sequence models consist of two neural networks. The encoder network takes the input sequence one character at a time and generates the context vector. The decoder network is initialized with this context vector (final encoder states). Then the decoder network is used to generate the target sequence one character at a time. This system is adopted here as it handles the difference in the syntax and length mismatch. This makes the model very flexible and scalable. The system works as illustrated in the fig 2.

The direction in which we feed input sequence to the encoder also affects the output [4][13]. We experimented with a different input feed directions and compared the performance of the systems, however the output was always in the forward direction. The feed directions we experimented with are discussed below, followed by the results.

1) *Forward feed:*

Here we directly feed the sequences as we read. This is logically the most obvious input format, however research shows that there are better feed directions[13]. The sample input and output for the pair "साहीन ↔ sahin" with different feed directions has been shown in Table 4.

2) *Reverse feed:*

Here we reverse the input sequence to the encoder. This somehow gives better results than the forward feed, especially for longer words.

3) *Reverse spaced-out feed:*

It was observed that, regardless of the feed direction, the model's performance for the longer and obscure words still had some issues. We hypothesized that, as large chunk of the training corpus comprised of shorter words, the initial few bits of the word encoding contribute more to the context building than the succeeding ones explaining the issues with longer words. We spaced-out the characters throughout the maximum length (36 in our case) of the input word in an attempt at building a more uniform distribution at context building of the words.

TABLE 4: FEED DIRECTION SAMPLES

	Input	**Output**
Forward	स ा ह ी न \n - - - - - - - - -	\t s a h i n \n - - - - - - - -
Reverse	न ी ह ा स \n - - - - - - - - -	\t s a h i n \n - - - - - - - -
Reverse Spaced out	न - ी - ह - ा - स - \n - - -	\t s a h i n \n - - - - - - - -

The models are evaluated using Word Error Rate (WER) and Character Error Rate (CER) as evaluation metrics. The results of the discussed feed directions are shown in Table 5. We have tested the models with different feeds for only one language pair, since the feed direction is not language specific. The results clearly show that reverse feed reduces the CER and WER. This could be because reversing the order generates many short-term dependencies as stated in [4] [6].

TABLE 5: RESULTS OF SEQ2SEQ OVER FEED DIRECTIONS

Model Seq2Seq	**Feed-direction**	**Character Error rate**	**Word Error rate**
E2H	Forward	26.1	56.87
E2H	Reverse	24.91	55.37
E2H	Reverse spaced out	26.04	56.83
H2E	Forward	16.66	42.18
H2E	Reverse	16.17	42.18
H2E	Reverse spaced out	16.45	42.18

We have used only one hidden layer with 256 neurons for both encoder and decoder. During the training, time we used teacher enforcing, i.e., the decoder is fed with the actual previous character instead of the character predicted by the decoder in previous time-step. During the prediction, the decoder is fed with the previous predicted character. We use greedy search algorithm for the word prediction. Table 6 shows the results of the Sequence to Sequence models which are fed reverse input on different language pairs.

TABLE 6: RESULTS OF SEQ2SEQ OVER LANGUAGE PAIRS

Language pair	**WER**	**CER**
Sindhi(Devanagari) to Sindhi(Arabic)	65.41	16.82
Sindhi(Arabic) to Sindhi(Devanagari)	23.85	5.29
Kannada toEnglish	53.13	12.15
English to Kannada	55.37	24.91
Hindi to English	42.18	16.17

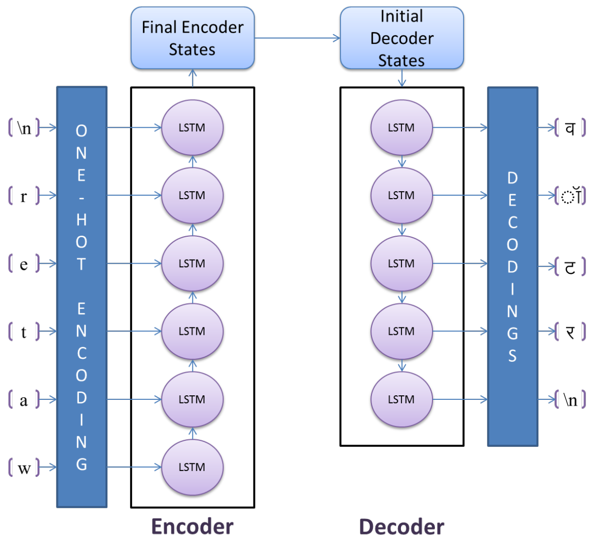

Fig. 2. Sequence-to-sequence model with encoder and decoder network

VI. POST PROCESSING

The sequence to sequence model outputs a probability distribution over 'k' output classes, i.e., the models have a softmax activation on the output layer. We found patterns in the probability distribution that validates the hypothesis that sequence to sequence models can handle certain language specific challenges appropriately. These patterns are discussed below.

Tables 7, 8 and 9 clearly show the patterns in the probability distribution. The probability difference between the two most probable characters show how plausible the output class is, i.e., a higher difference (~0.9 which is the norm) indicates how unrivalled the output is and a lower difference (< 0.45) points to a case where there could be another plausible contender for the output.

A. Alternative matra combinations

In Indian languages, there are different vowel signs used to represent varying lengths of the vowel sound, e.g., in Devanagari ' ि ' represents shorter ' i ' vowel whereas ' ी ' represents the longer one. However, in practice the usage of the same is not consistent and is highly subjective to various parameters. This gives rise to ambiguous mappings in the dataset and the following results portray how sequence to sequence models handle these ambiguities.

TABLE 7: PREDICTIONS FOR DIP WHICH CAN MAP TO दिप AND दीप

Time-step	Most probable char	2nd most probable char	Probability difference
1	द	ड	0.8791721
2	ि	ी	**0.3205122**
3	प	फ	0.9884761

The aberration in the probability difference clearly shows how the sequence to sequence model has adapted to the ambiguous mappings in the data and has assigned significant enough probability to both the characters in the second time-step for them to be plausible outputs.

B. Similar mappings for Retroflex and Dental Varga consonants

Among all the character classifications [12] in Indian languages, the transliterated interpretation of two of the vargas, i.e., Retroflex and Dental, quite often conflict with each other. E.g., Table 8 describes such a relationship between ' द ' and ' ड ', which can be extrapolated to the rest of the pairs, i.e., ट-त, ठ-थ, ड-द, ढ-ध and ण-न.

TABLE 8: PREDICTIONS FOR RETROFLEX AND DENTAL CONFLICTS

Time-step	Most probable char	2nd most probable char	Probability difference
1	म	क	0.99647903
2	ो	ॉ	0.9649083
3	द	ड	**0.31311885**
4	ी	ि	0.79156876

In the third time-step, it is clearly visible that the data has dictated ambiguous mappings between द and ड, which has manifested in the form of marginal probability difference. The cases described in VII.A and VII.B can be handled using simple substitution.

C. Matra insertion/deletion

We observed that in some cases where the probability difference is low and one of the two most probable characters is a matra, a mere substitution results in the removal of a valid non-matra character. This case is shown in Table 9. This case can be handled by the insertion of the ' ा ' matra instead of substitution.

TABLE 9: PREDICTIONS FOR MATRA INSERTION/DELETION

Time-step	Most probable char	2nd most probable char	Probability difference
1	क	ख	0.997335
2	म	ा	**0.106853**

VII. CONCLUSION

In this paper, we compared the performance of scalable system for transliteration of Indian languages using deep learning. The results show that the reverse feed sequence 2 sequence models are the most successful of the lot. Sequence 2 sequence models perform better than the other models, not only in terms of the performance metrics but also in terms of handling the language specific nuances discussed. However, they do have a critical drawback, i.e., they need a large dataset for training good models. The LSTM models do not need such large datasets; however, they are incapable of handling issues which seq2seq handles.

REFERENCES

[1] C-DAC GIST, "https://www.cdac.in/index.aspx?id=mlc_gist_ntrans". Accessed on: 29th July 2018

[2] Kanwaljit Kaur. Machine transliteration: A Review of Literature.International Journal of Engineering Trends and Technology (IJETT) – Volume 37 Number 6 - July 2016

[3] "https://datascience.berkeley.edu/machine-learning-neural-systems/". Accessed on: 29th July 2018

[4] Ilya Sutskever, Oriol Vinyals, Quoc V. Le. Sequence to Sequence Learning with Neural Networks. arXiv:1409.3215 2014

[5] Sepp Hochreiter and Jurgen Schmidhuber. Long Short ¨Term Memory. Technical Report FKI-207-95, Technische Universitat M ¨ unchen, M ¨ unchen, August 1995.

[6] Mihaela Rosca, Thomas Breuel. Sequence-to-sequence neural network models for transliteration. arXiv:1610.09565v1 2016

[7] "https://www.omniglot.com/writing/index.htm". Accessed on: 29th July 2018

[8] P. H. Rathod, M L Dhore, R. M. Dhore, HINDI AND MARATHI TO ENGLISH MACHINE TRANSLITERATION USING SVM, International Journal on Natural Language Computing (IJNLC), 2(4), 2013, pp. 57-71.

[9] XIA Li-sha, FANG Hua-jing, ZHENG Luo. Survey on Hidden Markov Model Based Fault Diagnosis and Prognosis.

[10] Kotsiantis, S.B. Artif Intell Rev (2013) 39: 261. https://doi.org/10.1007/s10462-011-9272-4

[11] Wöllmer, M., Eyben, F., Graves, A. et al. Cogn Comput (2010) 2: 180. https://doi.org/10.1007/s12559-010-9041-8

[12] Govind et. al. "https://archive.icann.org/en/topics/new-gtlds/devanagari-vip-issues-report-03oct11-en.pdf"

[13] Oriol Vinyals, Samy Bengio, Manjunath Kudlur. ORDER MATTERS: SEQUENCE TO SEQUENCE FOR SETS. arXiv:1511.06391

[14] "http://mhrd.gov.in/language-education". Accessed on: 14th August 2018