SEQUENCE TRANSLITERATION USING ENCODER-DECODER MODELS

Kartik Soni[1]

School of Computer Science and Engineering, Student, Vellore Institute of Technology, Vellore, Tamil Nadu, India

***

Abstract - *Machine transliteration is an emerging research area which converts words from one language to another without losing its phonological characteristics. Transliteration is a supporting tool for machine translation and Cross language information retrieval. Transliteration is mainly used for handling named entities and out of vocabulary words in a machine translation system. It preserves the phonetic structure of the words. We propose a modified neural encoder-decoder model that maximizes parameter sharing across language pair in order to effectively leverage orthographic similarity. We also show that Bilingual transliteration models can generalize well to languages/language pairs not encountered during training and hence perform well on the zero shot transliteration task. We have applied this technique for transliterating English to Hindi and achieved exact Hindi transliterations for 75-80% of English names.*

Key Words: **Transliteration, Bilingual, Language.**

1. INTRODUCTION

Transliteration converts the text from one script to another. Transliteration can be seen as two level processes: first segmenting the source language word into transliteration units and then aligning and mapping these units to target language units. For e.g. the word “Mera” which can be segmented as (m,e,r,a) then these units are transliterated to target language units. Transliteration is mainly used to convert the foreign words in a language which are required to be phonetically but need not to be grammatically equivalent to the words in another language.

Transliteration simply converts a text from one script to another. It is not concerned about faithfully representing the sounds of the original; rather, it focusses on representing the characters with as much accuracy and unambiguity as possible. As a technique, transliteration can be seen in two ways. When one writes native terms using a non-native or foreign script, it is called forward transliteration. For example, GULAB (in Devanagari script) is a Hindi word meaning rose in English. It can be transliterated (written) in Roman script as gulab , gulaab , goolab or in some other form. On the other hand, when one represents conversion of a term back to its native script from a non-native script, it is called back-transliteration. For example, gulab written in Roman script is back-transliterated to in its native script. Forward transliteration allows for creativity of the transliterator, whereas back-transliteration is ideally strict and expects the same initial word to be generated (with some exceptions, especially for East Asian languages).

We propose a compact neural encoder-decoder model for multilingual transliteration, that is designed to ensure maximum sharing of parameters across languages while providing room for learning language-specific parameters. This allows greater sharing of knowledge across language pairs by leveraging orthographic similarity. We empirically show that models with maximal parameter sharing are beneficial, without increasing the model size. Machine translation or transliteration models are inspired by deep representation learning. Their memory requirements are very less as compared to statistical models. Neural Networks are trained after for certain domains or applications. After training, the network practices. With time it starts operating according to its own judgment, turning into an "expert".

2. LITERATURE SURVEY

Arbabi et al. developed an Arabic-English transliteration system [1] using knowledge-based systems and neural networks. The first step in this system was to enter the names into the database which was obtained from telephone dictionary. As in Arabic script, short vowels are generally not written, a knowledge-based system is used to vowelize these names to add missing short vowels. The words which cannot be properly vowelized by KBS are then eliminated using artificial neural network. The network is trained using cascade correlation method, a supervised, feed forward neural processing algorithm. Thus the reliability of the names in terms of Arabic syllabification is determined through neural networks. The output of the network is in binary terms. The artificial neural network is trained on 2800 Arabic words and tested on 1350 words. After this, the vowelized names are converted into phonetic roman representation using a parser and broken down into groups of syllables. Finally the syllabified phonetics is used to produce various spellings in English.

Kang et. al. presented an English-to-Korean automatic transliteration and back transliteration system [2] based on decision tree learning. The proposed methodology is fully bidirectional. They have developed very efficient character alignment algorithm that phonetically aligns the English words and Korean transliteration pairs. The alignment reduces the number of decision trees to be learned to 26 for English-to-Korean transliteration and to 46 for Korean-to-English back transliteration. After learning, the transliteration and back transliteration using decision tree is straightforward.

Wan and Verspoor have proposed an "Automatic English-Chinese name Transliteration" [3] system. The system transliterated on the basis of pronunciation. That is, the written English word was mapped to written Chinese character via spoken form associated with the word. The system worked by mapping an English word to a phonemic representation and then mapping each phoneme to a corresponding Chinese character. The transliteration process consisted of five stages: Semantic Abstraction, Syllabification, Sub-syllable divisions, Mapping to Pinyin and Mapping to Han characters. Semantic abstraction was a preprocessing step that performed dictionary lookups to determine which parts of the word should be translated or which should be transliterated. The phonetic representation of each sub syllable was transformed to Pinyin, which is the most common standard Mandarin Romanization system.

Harshit Surana and Anil Kumar Singh in 2008, proposed a transliteration system on two Indian languages Hindi and Telugu [4]. In their experiment, a word was first classified as Indian or foreign using character based n - grams. The probability about word's origin was computed based on symmetric cross entropy. Based on this probability measure, transliteration was performed using different techniques for different classes (Indian or foreign). For transliteration of foreign words, the system first used a lookup dictionary or directly map from English phoneme to IL letters. For transliteration of Indian word, the system first segmented the word based on possible vowels and consonant combinations and then mapped these segments to their nearest letter combinations using some rules. The above steps generate transliteration candidates which were then filtered and ranked using fuzzy string matching in which the transliteration candidates were matched with the words in the target language corpus to generate target word. The out of vocabulary words are not handled by this system.

Deep and Goyal have developed a Rule based Punjabi to English transliteration system for common names [5]. The proposed system works by employing a set of character sequence mapping rules between the languages involved. To improve accuracy, the rules are developed with specific constraints. This system was trained using 1013 preson's names and tested using different person names, city names, river names etc. The system has reported the overall accuracy of 93.22%.

P.J. et. all. proposed English to Kannada transliteration system [6] using Support Vector Machine. The proposed system uses sequence labeling approach for transliteration which is a two step approach. The first step performs segmentation of source string into transliteration units and the second step performs comparisons of source and target transliteration units. It also resolves different combination of alignments and unit mappings. The whole process is divided into three phases: preprocessing, training using SVM and transliteration. The preprocessing phase converts the training file into a format required by SVM. The authors are using database of 40,000 Indian place names for the training of SVM. In this phase, During training phase, aligned source language names are used as input and target language names are used as label sequence and given to SVM. The training phase generates a transliteration model which produces top N probable Kannada transliteration during transliteration phase. The system is tested on 1000 out of corpus place names. The system is also compared with Google Indic system and reported higher accuracy while transliterating Indian names and places. The overall accuracy of the system is 87.28%.

Dhore et al. proposed Hindi to English transliteration of Named entities using Conditional random Fields [7]. Indian places names are taken as input in Hindi language using Devanagari script by the system and transliterated into English. The input is provided in the form of syllabification in order to apply the n-gram techniques. This syllabification retains the phonemic features of the source language Hindi into transliterated form of English. The aim is to generate transliteration of a named entity given in Hindi into English using CRF as a statistical probability tool and n-gram as a feature set. The proposed system was tested using bilingual corpus of 7251 named entities created from web resources and books. The commonly used performance evaluation parameter was "word accuracy?. The system has received very good accuracy of 85.79% for the bi-grams of source language Hindi.

Sanjanashree and Anand Kumar presented a framework for bilingual machine transliteration for English and Tamil based on deep learning [8]. The system uses Deep belief Network (DBN) which is a generative graphical model. The transliteration process consists of three steps viz. Preprocessing, Training using DBN and testing. The preprocessing phase does the Romanization of Tamil words. The data in both languages is converted to sparse binary matrices. Character padding is done at the end of every word to maintain the length of the words constant while encoding as sparse binary matrices. Deep Belief Network is a generative graphical model made up of multiple layers of Restricted Boltzmann Machine, a kind of Random Markov Field and Boltzmann Machine.

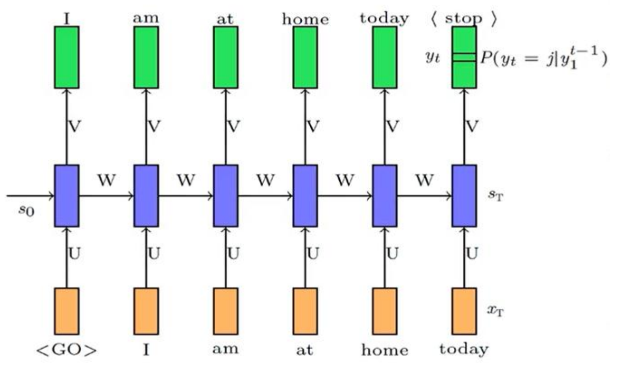
Mathur and Saxena have developed a system for English-Hindi named entity transliteration [9] using hybrid approach. The system first processes English words to extract phonemes using rules. After that statistical approach converts the English phoneme to equivalent Hindi phoneme. The authors have used Stanford's NER for name entity extraction and extracted 42,371 name entities. Rules were applied to these entities and phonemes were extracted. These English phonemes were transliterated to Hindi and a knowledgebase of English-Hindi phonemes was created.

Lehal and Saini have developed "Sangam: A Perso-Arabic to Indic Script Machine Transliteration Model" [10]. Sangam is a hybrid system which combines rules as well as word and character level language models to transliterate the words. The system has been successfully tested on Punjabi, Urdu and Sindhi languages and can be easily extended for other languages like Kashmiri and Konkani. The transliteration accuracy for the three scripts ranges from

91.68% to 97.75%, which is the best accuracy reported so far in literature for script pairs in Perso-Arabic and Indic script.

3. PROPOSED SYSTEM

Language Modeling is the task of predicting what word/letter comes next. Unlike the FNN and CNN, in sequence modeling, the current output is dependent on the previous input and the length of the input is not fixed. Given a 't-1' words, we are interested in predicting the ith word based on the previous words or information. This is how we solve the language modeling using Recurrent Neural Networks.

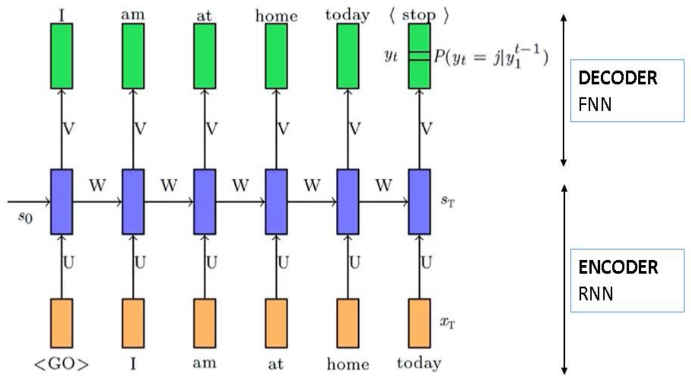


The input to the function is denoted in orange color and represented as an **x_t**. The weights associated with the input is denoted using a vector **U** and the hidden representation (**s_t**) of the word is computed as a function of the output of the previous time step and current input along with bias. The output of the hidden represented (**s_t**) is given by the following equation,

$$s_T = \sigma(Ux_T + Ws_{T-1} + b)$$
$$y_t = O(Vs_T + c)$$

we compute the hidden representation of the input, the final output (**y_t**) from the network is a softmax function (represented as O) of hidden representation and weights associated with it along with the bias.

ENCODER-DECODER MODEL:

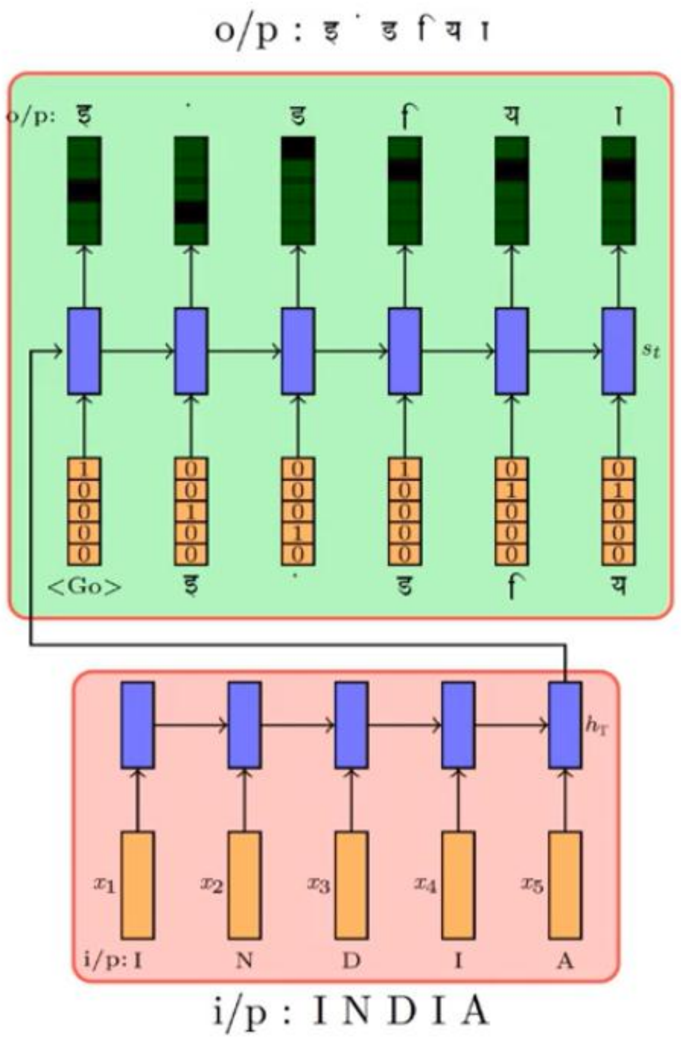


Encoder Model

The RNN the output of the first time step is fed as input along with the original input to the next time step. At each time step, the hidden representation (**s_{t-1}**) of the word is computed as a function of the output of the previous time step and current input along with bias. The final hidden state vector(s_t) contains all the encoded information from the previous hidden representations and previous inputs. Here, Recurrent Neural Network is acting as an **Encoder**.

Decoder Model

Once we pass the encoded vector to the output layer, which decodes into the probability distribution of the next possible word. The output layer is a softmax function and it takes hidden state representation and weights associated with it along with the bias as the inputs. Since the output layer contains the linear transformation and bias operation, it can be referred to as the simple feed-forward neural network. Feed-Forward Neural Network is acting as a **Decoder.**



Model

- **Encoder:**

$$h_t = RNN(h_{t-1}, x_{it})$$

- **Decoder:**

$$s_0 = h_T \quad (T\ is\ length\ of\ input)$$

$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$

$$P(y_t|y_1^{t-1}, x) = softmax(Vs_t + b)$$

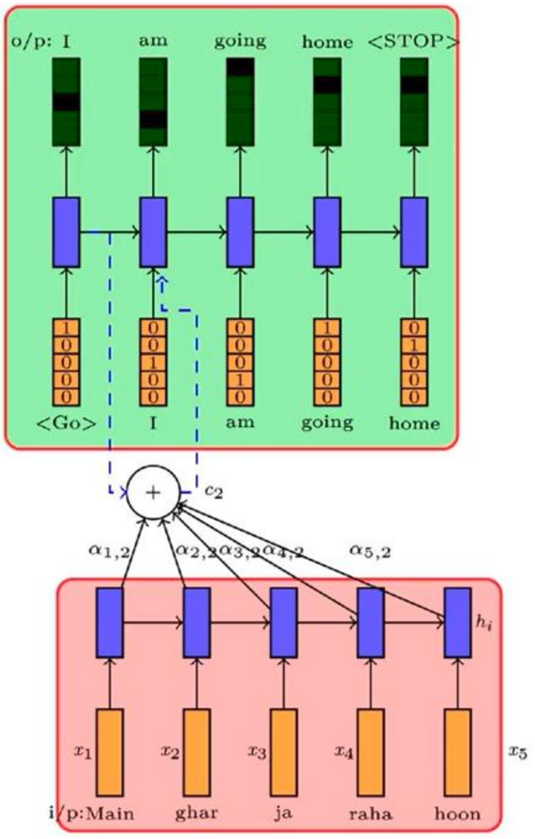
ATTENTION MODEL SOLUTION:

The problem in this approach is that encoder reads the entire sentence only once and it has to remember everything and converts that sentence to an encoded vector. For longer sentences, the encoder will not be able to remember the starting parts of the sequence resulting in the loss of information.

We, humans, try to translate each word in the output by focusing only on certain words in the input. At each time-step, we take only relevant information from the long sentences and then translate that particular word. ***Ideally, at each time-step, we should feed only the relevant information (encodings of the relevant information) to the decoder for the translation.***

What else we need?

So for each input word, we assign a weight **α** (ranges between 0–1) that represents the importance of that word for the output at the time-step '**t**'. For example, **α**12 represents the importance of the first input word on the output word at the second time-step. To generalize, the representation **α**jt represents the weight associated with the jth input word at the tth time-step. For example, at time-step 2, we could just take a weighted average of the corresponding word representations along with the weights **α**jt and feed it into the decoder. In this scenario, we are not feeding the complete encoded vector into the decoder, rather the weighted representation of the words.



Encoder

The encoder operation doesn't change much when we compare it to the vanilla version of encoder-decoder architecture without attention. At each time step, the representation of each word is computed as a function of the output of the previous time step and current input along with bias. The final hidden state vector(s□) contains all the encoded information from the previous hidden representations and previous inputs. RNN is used as an encoder.

Encoder:

$$h_t = RNN(h_{t-1}, x_t)$$

$$s_0 = h_T$$

Decoder

In the vanilla version of the encoder-decoder model, we would pass the entire encoded vector to the output layer, which decodes into the probability distribution of the next possible word. Instead of passing the entire encoded vector, we need to find the attention weights using the fancy equation that we discuss in the last section to find **e**jt. Then normalize the **e**jt weights using softmax function to get **α**jt. Once we have all the inputs to feed into the decoder and weights associated with them (Thanks to the fancy equation!), we will compute the weighted combination of all the inputs and weights to get the resultant vector Ct. We will feed the weighted combination vector Ct to the Decoder RNN, which decodes into the probability distribution of the next possible word. This operation of decoding goes for all the time-steps present in the input. The output layer is a softmax function and it takes hidden state representation and weights associated with it along with the bias as the inputs.

Decoder:

$$e_{jt} = V_{attn}^{T} tanh(U_{attn} h_j + W_{attn} s_t)$$

$$\alpha_{jt} = softmax(e_{jt})$$

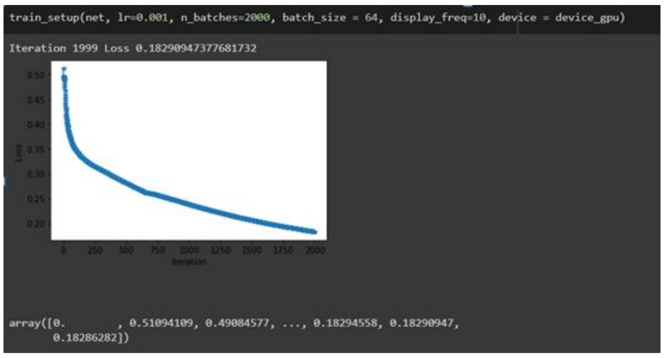
$$c_t = \sum_{j=1}^{T} \alpha_{jt} h_j$$

$$s_t = RNN(s_{t-1}, [e(\hat{y}_{t-1}), c_t])$$

$$\ell_t = softmax(V s_t + b)$$

4. IMPLEMENTATION AND RESULTS

Loss without Attention:

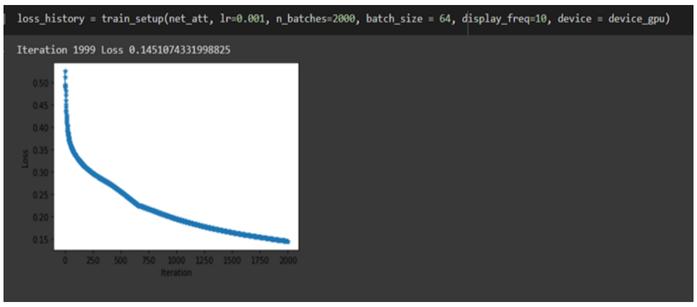


```
train_setup(net, lr=0.001, n_batches=2000, batch_size = 64, display_freq=10, device = device_gpu)

Iteration 1999 Loss 0.18290947377681732
```

```
array([0.        , 0.51094109, 0.49084577, ..., 0.18294558, 0.18290947,
       0.18286282])
```

Loss with Attention:



```
loss_history = train_setup(net_att, lr=0.001, n_batches=2000, batch_size = 64, display_freq=10, device = device_gpu)

Iteration 1999 Loss 0.1451074331998825
```

5. CONCLUSIONS

In this paper work, we have presented a survey on challenges, different approaches and evaluation metrics used for different machine transliteration systems. We have also listed some of the existing transliteration systems. From the survey we have found that almost all existing language machine transliteration systems are based on statistical and hybrid approach. We Implemented Bilingual transliteration model for languages (English and Hindi); Accuracy of the simple encoder-decoder model was close to 72% while Accuracy of the Attention encoder-decoder model was approximately 80%.

Given that transliteration is often used as part of machine translation systems, and that such systems themselves are increasingly character based end-to-end system, the question arises whether we need separate transliteration models at all. It appears likely that transliteration will remain a distinct submodule of such systems, since internal graphemic and phonetic representations inside transliteration modules are likely quite different from internal semantic representations required for translation. Experimental evidence from humans also supports the notion of separate and distinct processing of proper nouns and other nouns.

REFERENCES

[1] M. Arbabi, S. M. Fischthal, V. C. Cheng, and E. Bart, "Algorithms for Arabic name transliteration," IBM Journal of research and Development, vol. 38, pp. 183-194, 1994.

[2] B.-J. Kang and K.-S. Choi, "Automatic Transliteration and Back-transliteration by Decision Tree Learning," in LREC, 2000.

[3] 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, 1998, pp. 1352-1356

[4] H. Surana and A. K. Singh, "A More Discerning and Adaptable Multilingual Transliteration Mechanism for Indian Languages," in IJCNLP, 2008, pp. 64-71.

[5] K. Deep and V. Goyal, "Development of a Punjabi to English transliteration system," International Journal of Computer Science and Communication, vol. 2, pp. 521-526, 2011.

[6] P. Antony, V. Ajith, and K. Soman, "Kernel method for english to kannada transliteration," in Recent Trends in Information, Telecommunication and Computing (ITC), 2010 International Conference on, 2010, pp. 336-338.

[7] M. L. Dhore, S. K. Dixit, and T. D. Sonwalkar, "Hindi to english machine transliteration of named entities using conditional random fields," International Journal of Computer Applications, vol. 48, pp. 31-37, 2012.

[8] P. Sanjanaashree, "Joint layer based deep learning framework for bilingual machine transliteration," in Advances in Computing, Communications and

Informatics (ICACCI, 2014 International Conference on, 2014, pp. 1737-1743.
[9] S. Mathur and V. P. Saxena, "Hybrid appraoch to EnglishHindi name entity transliteration," in Electrical, Electronics and Computer Science (SCEECS), 2014 IEEE Students' Conference on, 2014, pp. 1-5.
[10] G. S. Lehal and T. S. Saini, "Development of a Complete Urdu-Hindi Transliteration System," in COLING (Posters), 2012, pp. 643-652.
[11] S. Karimi, F. Scholer, and A. Turpin, "Machine transliteration survey," ACM Computing Surveys (CSUR), vol. 43, p. 17, 2011.
[12] K. Kaur and P. Singh, "Review of Machine Transliteration Techniques," International Journal of Computer Applications, vol. 107, 2014.
[13] B.-J. Kang and K.-S. Choi, "Automatic Transliteration and Back-transliteration by Decision Tree Learning," in LREC, 2000.
[14] J.-H. Oh and K.-S. Choi, "An English-Korean transliteration model using pronunciation and contextual rules," in Proceedings of the 19th international conference on Computational linguistics-Volume 1, 2002, pp. 1-7.
[15] M. G. Malik, "Punjabi machine transliteration," in Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, 2006, pp. 1137-1144
[16] C.-J. Lee and J. S. Chang, "Acquisition of English-Chinese transliterated word pairs from parallel-aligned texts using a statistical machine transliteration model," in Proceedings of the HLT-NAACL 2003 Workshop on Building and using parallel texts: data driven machine translation and beyond-Volume 3, 2003, pp. 96-103.
[17] H. Surana and A. K. Singh, "A More Discerning and Adaptable Multilingual Transliteration Mechanism for Indian Languages," in IJCNLP, 2008, pp. 64-71.
[18] G. Hong, M.-J. Kim, D.-G. Lee, and H.-C. Rim, "A hybrid approach to english-korean name transliteration," in Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration, 2009, pp. 108-111.
[19] R. C. Balabantaray and D. Sahoo, "Odia transliteration engine using moses," in Business and Information Management (ICBIM), 2014 2nd International Conference on, 2014, pp. 27-29.
[20] M. A. Zahid, N. I. Rao, and A. M. Siddiqui, "English to Urdu transliteration: An application of Soundex algorithm," in Information and Emerging Technologies (ICIET), 2010 International Conference on, 2010, pp. 1-5
[21] C. Sunitha and A. Jaya, "A phoneme based model for english to malayalam transliteration," in International Confernce on Innovation Information in Computing Technologies, 2015, pp. 1-4.
[22] M. A. Malik, C. Boitet, L. Besacier, and P. Bhattcharyya, "Urdu Hindi machine transliteration using SMT," WSSANLP-2013, p. 43, 2013.
[23] P. Rathod, M. Dhore, and R. Dhore, "Hindi and Marathi to English machine transliteration using SVM," International Journal on Natural Language Computing, vol. 2, pp. 55- 71, 2013.
[24] A. A. Kak, N. Mehdi, and A. A. Lawaye, "Building a Cross Script Kashmiri Converter: Issues and Solutions," Proceedings of Oriental COCOSDA (The International Committee for the Co-ordination and Standardization of Speech Databases and Assessment Techniques), 2010.
[25] J. Kaur and G. S. Josan, "Statistical Approach to Transliteration from English to Punjabi," International Journal on Computer Science and Engineering, vol. 3, pp. 1518-1527, 2011.
[26] Waleed Ammar, Chris Dyer, and Noah A Smith. 2012. Transliteration by sequence labeling with lattice encodings and reranking. In Proceedings of the 4th Named Entity Workshop, pages 66– 70. Association for Computational Linguistics.
[27] Mayce Al Azawi, Muhammad Zeshan Afzal, and Thomas M Breuel. 2013. Normalizing historical orthography for ocr historical documents using lstm. In Proceedings of the 2nd International Workshop on Historical Document Imaging and Processing, pages 80–85. ACM.
[28] Kanishka Rao, Fuchun Peng, Hasim Sak, and Franc¸oise Beaufays. 2015. Grapheme-tophoneme conversion using long short-term memory recurrent neural networks. In Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on, pages 4225–4229. IEEE.
[29] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Advances in neural information processing systems, pages 3104–3112.
[30] Kaisheng Yao and Geoffrey Zweig. 2015. Sequence-to-sequence neural net models for grapheme-to-phoneme conversion. arXiv preprint arXiv:1506.0019.