Data generation using sequence-to-sequence

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*—Sequence to Sequence models have shown a lot of promise for dealing with problems such as Neural Machine Translation (NMT), Text Summarization, Paraphrase Generation etc. Deep Neural Networks (DNNs) work well with large and labeled training sets but in sequence-to-sequence problems, mapping becomes a much harder task due to the differences in syntax, semantics and length. Moreover usage of DNNs is constrained by the fixed dimensionality of the input and output, which is not the case with most of the Natural Language Processing (NLP) problems. Our primary focus was to build transliteration systems for Indian languages. In the case of Indian languages, monolingual corpora are abundantly available but a parallel one which can be directly applied to transliteration problem is scarce. With the available parallel corpus, we could only build weak models. We propose a system to leverage the mono-lingual corpus to generate a clean and quality parallel corpus for transliteration, which is then iteratively used to tune the existing weak transliteration models. The results that we got prove our hypothesis that the process of generation of clean data can be validated objectively by evaluating the models alongside the efficiency of the system to generate data in each iteration.**

Keywords— Sequence2Sequence, NLP, transliteration, LSTM, encoder, decoder

I. INTRODUCTION

It has been a challenge for neural networks to learn sequence-to-sequence mappings, because of the difference in syntax, semantics and length in the input and output sequences. Transliteration is a sequence-to-sequence problem in which both the inputs and outputs are sequences written in different scripts. Using the encoder-decoder scheme for tackling such problems has shown great results [1] [4], because of the fact that the encoder has to go through the entire input first to build a context matrix which then is used by the decoder to help with its predictions. Sequence-to-sequence models take care of the difference in the syntax, length and also emphasizes on the semantics.

Our main focus was on an English/Indian Language to Indian Language transliteration system where parallel data was not available in large quantity. Using the available parallel data, we built the systems for transliteration using sequence-to-sequence architecture and as expected, due to weak models, the results were mediocre. We then leveraged the abundantly available monolingual corpus to feed the data generation system. The data generation system we proposed uses the weak models to create and filter the data from the available mono-lingual corpus, which in turn is used to tune the models. The generated data might not be completely clean, but the continual evaluation has empirically shown that the noise does not adversely affect the model's performance. The data generation process is both heavy on computation and time consumption, because of which we had to experiment with the constituting model's parameters and feed direction to make sure they are optimal.

II. PREVIOUS WORK AND MOTIVATION

Conventionally, attempts at solving transliteration have been usually achieved by using knowledge based systems or traditional machine learning approaches such as Support Vector Machines [7], Conditional Random Fields [9], Decision trees [8], LSTM [10] etc. The earliest approach of knowledge based systems was heavily dependent on domain

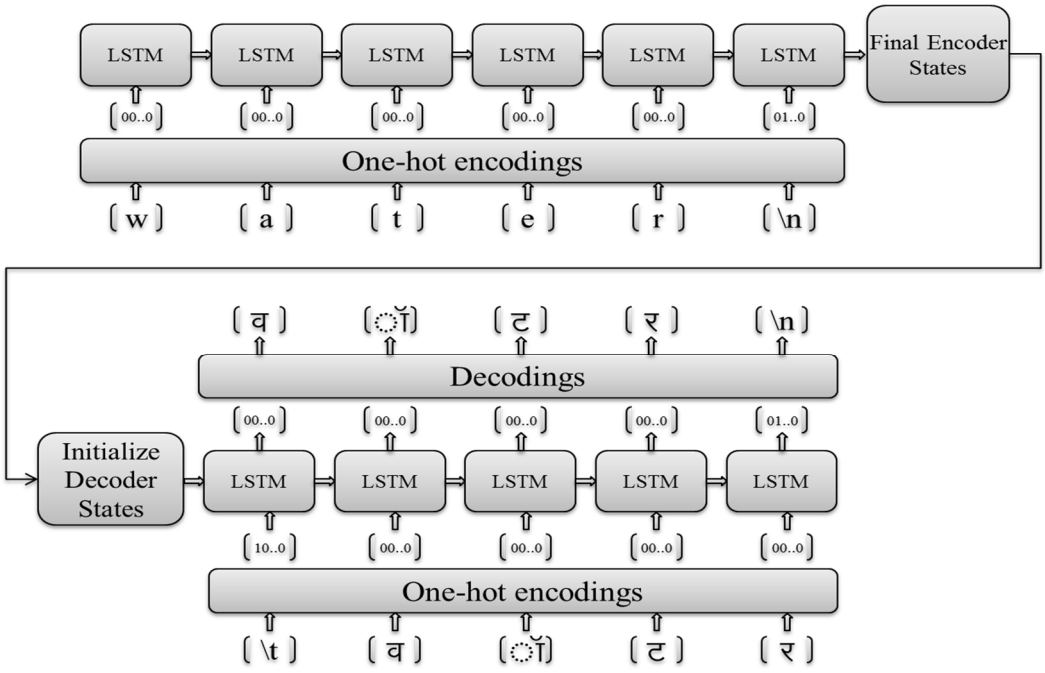

Fig. 1: Training setup for sequence-to-sequence with teacher enforcing.

knowledge and expert's adeptness. Scaling-up such methods for a mix of mainstream and obscure native languages was not easy for our use case due to the human factor involved. Grapheme based, Phoneme based and Hybrid transliteration systems [6] were some approaches which used the traditional machine learning techniques, but these systems were not scalable either because of the problem complexity.

The problem is complex as different basic alphabets and phonetic systems are used by different languages. This also results in different lengths of the input and output sequences. Moreover, there is no universal standard mapping possible for a transliteration problem, making the problem subjective to such an extent that evaluation of such systems is very unclear. It is clear from the facts stated above that this is not a straight forward one to one mapping problem. This results in complex relations between the input and output pairs. Hence, we used a vanilla sequence-to-sequence model as described in [1] [4] to tackle these problems.

In some languages, there is a direct, non-ambiguous and one-to-one correspondence between a particular grapheme and the intended sound of the same e.g. Hindi, whereas other languages may not necessarily have such properties, e.g. English. This creates complexities as; in the case of the latter, the same set of graphemes can correspond to multiple phonemes, e.g. in case of 'character' the 'ch' corresponds to 'k' sound "/ˈkarəktə/" whereas in case of 'chart', the 'ch' corresponds to 'cha' sound "/tʃɑ :t/". This phenomenon introduces another set of challenges for Transliteration problem. When a word represented in a language from a former set is to be transliterated to a language from the latter set, the multiple possibilities necessitate the need for specific contextual rules. This can be easily represented in rule-based transliteration systems, however this poses a major problem for machine learning based transliteration systems which are solely data driven. For a sequence-to-sequence transliteration system, to extract these associations, a considerable amount of data is required [2].

While training the weak models which would subsequently be employed in generating more parallel data, it was observed that the feed direction of the input data also affects the results of our model. This could be because reversing the order generates many short-term dependencies as stated in [1] [4], but the exact reason is not clear. We tried reversing the input order to validate the results in [3] and found that it did help with longer words where the forward feed model did not perform well. With the intent of finding a better feed direction and the reason behind why the feed direction mattered, we tried different methods of feeding the input to see the change in performance.

III. Datasets

We feed input to the encoder to generate hidden states to initialize the hidden states of the decoder. The hidden states are used for interpreting the context of the input. The way we feed the input sequence to encoder generates hidden states in different manner. How and why exactly these different orders matter is not very clear but as stated in [1], order does very much matter [11]. We experimented with three input feed directions, which are, forward, reverse and spaced out reverse and the results are shown in Table I.

A. Pre-processing

The feed direction is chosen and once the input word is accordingly manipulated, it is one-hot encoded. The maximum length of input is the longest word available in the corpus.

1) Forward feed

The usual input for sequence-to-sequence models is forward, which logically makes most sense but for some odd reason does not perform as good as the reverse feed for some inputs, specifically, longer ones. From what we have noticed, it is could be because of a large chunk of the dataset being short words. We then experimented with another custom-made dataset with longer and convoluted inputs, which did considerably improve the performance of the system.

2) Reverse feed

Much like the forward feed, we still pad the one-hot encoded string but it is now reversed. We've noticed that the model trained on this dataset works considerably better on longer words and there are rare occurrences of mistakes on shorter inputs.

3) Reverse spaced out

As we need to have fixed number of input and output characters, it sometimes becomes difficult for the model to understand the context of long words and usually messes up if the word is longer than a specific size. Hence here we experimented with spaced out characters during the input based on the max length of the word. We assumed that this might be helpful for longer words. The results show minimal increase in performance and on inspection, no patterns were found where this model excelled over the other two.

E2H references to the English-Hindi and H2E references to the Hindi-English transliteration model. These models were trained on a parallel corpus with 99995 exemplars. The average English and Hindi word lengths were 8.20 and 7.26 respectively. The English character set had 26 characters whereas Hindi had 70 characters.

TABLE I: RESULTS OF MODEL EVALUATION

Model Seq2Seq	Feed-direction	Character Error rate	Word Error rate
E2H	Forward	26.10	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

IV. Model Architecture

Sequence-to-sequence models [1] [4] are a combination of two neural networks, the encoder and the decoder, co-dependent on each other. The encoder is used to learn the representations of the input sequence. These learned representations, i.e., final encoder states are used to initialize the decoder hidden states. Then the decoder decodes the output sequence character-by-character in the target orthography. The model uses LSTM cells with single hidden layer for both encoder and decoder.

More advanced sequence-to-sequence models with attention are also used which help to focus on different parts of input

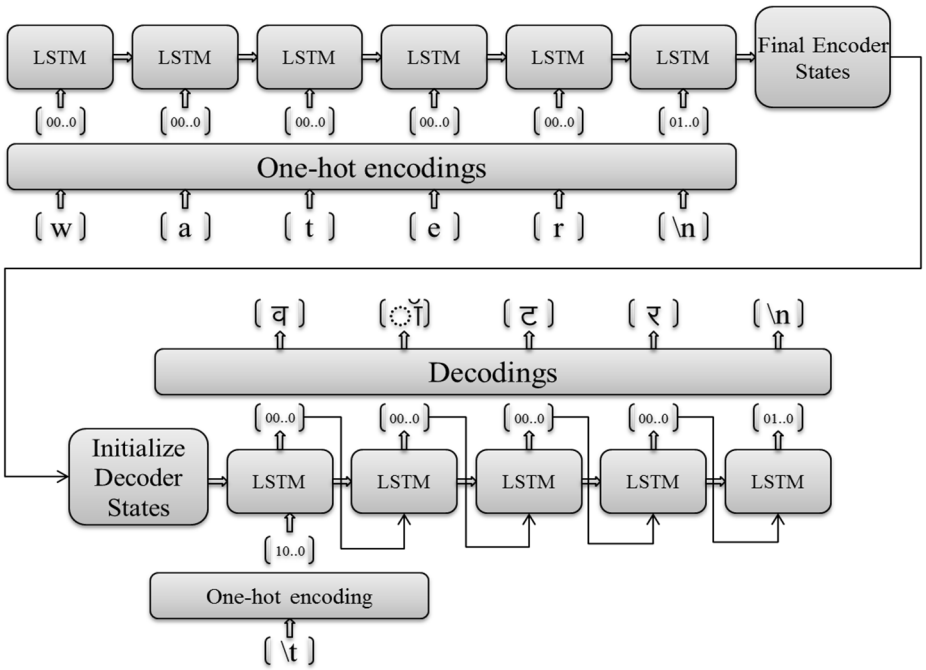

Fig. 2: Prediction setup for sequence-to-sequence.

for each time step in the output sequence [5]. These models are more flexible with the reordering of the input output, but this is not important for transliteration as the order of phonemes is preserved from source to the target. So we used vanilla sequence-to-sequence models for our experiments.

A. Training

For the training and prediction purposes '\t' and '\n' have been used as the start-of-word (SOW) and end-of-word (EOW) tokens respectively and are added to the character set. The characters are one-hot encoded and we have created an exhaustive character-set for the source and target languages. The characters outside this set are filtered out.

For all the experiments, we used 10% data for testing, 10% for validation and remaining 80% for training. The loss function used is categorical cross entropy. The model was optimized using mini-batch gradient descent with adaptive learning rate using RMSprop. The training setup is shown in Fig. 1.

Experiments were conducted for the batch sizes of 32, 64,128 with 64,128,256 hidden layer neurons, and trained for a total of 6, 9,12,15,18 epochs. The most optimal parameters were found to be batch size of 64 with 256 hidden layer neurons and training for 12 epochs.

During training, the word in source orthography is fed character-by-character to the encoder for learning the context or getting the final encoder states. The final encoder states are used to initialize the decoder hidden states. While training we use teacher enforcing, which means we feed the time shifted target sequence to the decoder instead of the previous character generated by decoder. The first character fed to the decoder is the SOW token which is '\t'.

B. Prediction

The operation of the encoder does not change during the prediction and the decoder states are initialized the same way. However, after the first character, the decoder is fed with the previous time-step prediction of the decoder as shown in Fig. 2. We use a greedy search algorithm for decoding the output with maximum probability.

As all the inputs and outputs are one-hot encoded, we need to convert it back to the target orthography using the same mappings which were used while training.

V. Generation of Data

Both models are trained on the limited parallel corpus available, because of which they build up as fairly weak models, i.e. a small fraction of outputs are completely unacceptable. We used these weak models, as shown in Fig. 3, to predict the output on a mono-lingual corpus, thereby creating a parallel corpus, albeit one riddled with inaccurate entries. This phase produces a parallel corpus in which a certain percentage of the data is accurate but there are inaccuracies in the data generated which have to be filtered out.

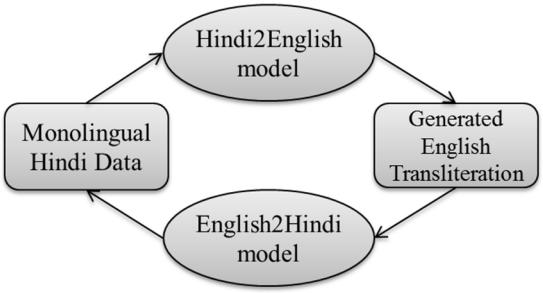

Fig. 3: Data generation System

A. Filtering generated data

Validating data with quantity of the order of 10^6 exemplars is a herculean task for humans. We designed an automated cyclic system to try and vet out as many inaccuracies as possible, leveraging the dual nature of the problem. The weak models constitute the proposed cyclic filtration system as described in Fig. 3. The idea behind this system is that post the generation of the parallel corpus we pass it through one weak model to convert it from source language to target language and then the intermediate output is fed to the accompanying model for back-transliteration. If the final output and the initial output are a perfect match, then and then only is the exemplar preserved.

VI. Experimental Results

Evaluating transliteration models in itself was an obstacle to begin with because of the subjective nature of the problem, particularly because of the existence of multiple, subjectively correct, mappings from a simpler language to a complex one. To objectively gauge the Transliteration models, we have used Character Error Rate (CER) and Word Error Rate (WER) as metrics. While training, the categorical cross entropy loss also helped us understand, in theory, how well the model was performing. We recorded these values for further comparison with the iteratively incremental models. We used the model pairs, i.e. target to source and vice-versa, in the filtration system to generate and clean the data. We hypothesized that, after tuning these models by re-training them on the generated clean data, the next cycle of generating the clean data would produce a larger dataset. We empirically found that this hypothesis does stand true. The performance of the weak models, iteratively trained on the generated clean data, did indeed improve over the cycle. The results for both the transliteration models are shown in Table II, alongside the summary outcome of the filtration iterations for the parallel data generation.

TABLE II: ITERATIVE EVALUATION OF MODELS

Iteration	Number of filtered exemplars	Character Error Rate		Word Error Rate	
		H2E	*E2H*	*H2E*	*E2H*
1	126612	14.82	22.90	46.89	62.90
2	134635	14.97	22.75	45.95	61.77
3	144300	14.73	22.24	45.39	59.32

A. Error analysis of the models

Table III gives a glimpse of some errors made by the Hindi-English and English-Hindi transliteration models. We found that the Hindi-English model had issues transliterating when phonetically similar alternatives were possible outputs, e.g., for *पंजाब* when *punjab* was the expected output, the model produced *panjab*, which also agrees with the pronunciation. We also noticed that the system had some trouble differentiating between the need of implicit vowels (typically referred in the linguistic jargon as the *'schwa'*) present in Indian languages and placing them, i.e., in case of *कांबले* which should have been transliterated to *kamble*, the model predicted *kamb**a**le*, adding an extra vowel 'a' which is also a possible interpretation but not relevant in this particular case. Even in the case of English-Hindi model, we observed a similar case where m*aharashtra* got transliterated to *महाराष्ट्रा* with an additional *ा* in the end which was not expected. Another type of error was also noted in case of proper nouns or loan words where the engine tried to convert it to the way it would be pronounced without taking into account any language specific nuances and thereby deviating from the established pronunciation of the word as given in the ground truth e.g., *indore* produces *इंदोरे* as opposed to *इंदौर*.

TABLE III: ERROR ANALYSIS

Input	Ground Truth	Model output
पंजाब	punjab	panjab
हरियाणा	haryana	hariyana
पवार	pawar	pavar
कांबले	kamble	kambale
बाबूराव	baburao	baburav
देवनागरी	devnagri	devanagari
indore	इंदौर	इंदोरे
kanmani	कन्मणि	कंमानी
khadakwasla	खडकवासला	खडकवस्ला
maharashtra	महाराष्ट्र	महाराष्ट्रा
migraine	माइग्रेन	मिगराइने

VII. Conclusion And Future Work

In this paper, we proposed a system to enhance the performance of transliteration models by generating a parallel corpus for tuning the models. The improvement visible in the evaluation metrics of the transliteration models corroborates our hypothesis and validates the data generated. Although this system has been validated only on the transliteration problem, we presume it can effectively be mapped to other sequence-to-sequence problems, like Q&A, Translation and Conversational Chatbots, as well. The caveat being, transliteration does not require variants of the sequence-to-sequence model like Attention based sequence-to-sequence to perform well. So, depending on the nature of the problem, the architecture of the constituting models needs to be changed for optimal performance.

References

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

[2] Kevin Knight, Jonathan Graehl. Machine Transliteration. ACM 1998

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

[4] Fethi Bougares, Holger Schwenk, Yoshua Bengio, Dzmitry Bahdanau, Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. arXiv:1406.1078v3 2014

[5] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. Neural machine translation by jointly learning to align and translate. arXiv:1409.0473v7 2015

[6] Anupama Sharma, Dr. Dhavleesh Rattan. MACHINE TRANSLITERATION FOR INDIAN LANGUAGES: A REVIEW. IJARCS. 2017

[7] 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.

[8] G.S. Lehal, A Gurmukhi To Shahmukhi Transliteration System, in: proceedings of ICON-2009:7th international conference on Natural Language Processing, 2009, pp.167-173.

[9] M.L. Dhore, S.K. Dixit, T.D. Sonwalkar, Hindi to English Machine Transliteration of Named Entities using Conditional Random Fields,

International Journal of Computer Applications (0975 – 8887), 48(23), 2012, pp. 31-37.

[10] Tigran Galstyan, Hrayr Harutyunyan, Hrant Khachatrian "https://yerevann.github.io/2016/09/09/automatic-transliteration-with-lstm/" . Accessed on: 29th June 2018

[11] Oriol Vinyals, Samy Bengio, Manjunath Kudlur. Order Matters: Sequence to Sequence for Sets. arXiv:1511.06391v4. 2016