TRANSLITERATION OF INDIAN LANGUAGES IN SOCIAL MEDIA CONTEXT

A Project Report Submitted

for the Course

MA498 Project II

by

Mitanshu Mittal - 160123020

Rahul Kumar Gupta - 160123030

to the

DEPARTMENT OF MATHEMATICS

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781039, INDIA

November 2016

CERTIFICATE

This is to certify that the work contained in this project report entitled "Transliteration of social media text for low resource languages" submitted by Mitanshu Mittal (Roll No.: 160123020) and Rahul Kumar Gupta (Roll No.: 160123030) to the Department of Mathematics, Indian Institute of Technology Guwahati towards partial requirement of Bachelor of Technology in Mathematics and Computing has been carried out by him/her under my supervision.

It is also certified that, along with literature survey, a few new results are established/computational implementations have been carried out/simulation studies have been carried out/empirical analysis has been done by the student under the project.

Turnitin Similarity: 16%

Guwahati - 781039	**Prof. Gautam K. Das**	**Prof. S. Ranbir Singh**.
November 2019	(Project Co-Supervisor)	(Project Co-Supervisor)
	(Mathematics Department)	(CSE Department)

ABSTRACT

Since the popularity of social media in India, the amount of unstructured text is enormous. Marketing agencies find this data very useful for advertising. Government agencies can also analyse social media data for different purposes including internal security, understanding the political sentiment etc. The informal social media environment promote colloquial elements in the text. For example, in the India context, many of Indians converse in native languages using Latin (transliterated) script without following proper grammatical or transliteration rules. One way to leverage this huge data is to transliterate them back to the native script and use traditional text processing models. We, therefore want to study state-of-the-art solutions of machine transliteration and investigate their effectiveness over different Indian languages. This will help us to better understand the strength and weaknesses of state-of-the-art transliteration methods and propose better solutions appropriate for Indian languages.

In this work, we start with introduction to the transliteration task, and then study major challenges in the field along with state of the art solutions. We will first look at bilingual solutions and then investigate advantages and architecture of various multilingual models.

We have evaluated different variations of character level RNN and Transformer based models for word transliteration and reported the results and observations obtained. Indian languages are low resource languages i.e. we don't have enough dataset to train our models. The dataset available is too noisy to give reliable results. Thus we have used the idea of multilingual translation in our multilingual transliteration models. In that we combined datasets for all pairs (En-Hindi, En-Bangla, En-Tamil) and train single transliteration model. In our research work we are trying to implement a one to many model using transformer architecture. The neural transliteration models using RNN(encoder-decoder) architecture take a lot of time to train, whereas transformer architecture inherently supports parallelism, thus it can reduce the training time if we have multiple cores available for training. We show that Multilingual model outperforms bilingual models when dataset is extremely scarce to train a decent bilingual model. We also observed that training along with a language with similar phonetic properties significantly boosts the accuracy of the model.

Contents

Chapter 1

Introduction

The process of phonetically transforming the words of a language from one script to another script is called transliteration.[20] for e.g (bacche –> बच्चे), (hamara –> हमरा).Converting a word written in the native script to a foreign script(for example “खुशबू ” to “Khushboo,” Devanagari is the native script here) is called forward-transliteration, and the conversion of a word back to its native script is called back-transliteration for example, “bacche” to “बच्चे ” . Traditionally transliteration has been used as a subtask of translation for converting words, which are nouns, names or out of dictionary words.
Multilingual Transliteration is a same concept except for the multiple source and target languages, where we transliterate between different language pairs. Multilingual transliteration is an example of multitask learning, in which individual tasks (language pairs) benefit from information sharing with similar tasks.

1.1 Motivation:

Since the advent of Web 2.0, the amount of data on the internet has been increasing at an exponential rate. There are over 3.48 billion social media users in 2019, and the number of users are growing at around 9% each year, with about a quarter of them in India.[1] Much of the textual information contained in these social media channels is in transliterated form. So in recent years, transliteration has become an important research topic in itself. Most of the Indians prefer to write regional language using Latin script. The reason being (1) Modern computers and mobiles provide seamless support for Latin script (2) Most of the Indian users are comfortable in using english keypads (3) Lack of suitable software for rendering native script (4) Users are inexperienced with writing in their native script.

Advantages of Multilingual models: (1) Since languages of Indian origin are low resource languages i.e. we don’ t have enough dataset to train our models. Here Multilingual transliteration can be a saviour where we can use all the available parallel corpus to train a single transliteration model.(2) Multilingual models are makes deployment easier and manageable, as we only have to maintain only one system for every language pairs.(3) Previous researches have shown that languages which are orthographically similar can train better models on language related tasks where parameter sharing across different languages is involved. [14]

1.2 Application:

The inference gained from the social media data can help us enhance social security, understand public sentiment, more personalized advertisement, better information retrieval, improve machine translation, and understand customer sentiment to improve marketing and customer services. However, existing text mining models are trained to analyze traditional formal text. However, since a large amount of this data is in the transliterated form, lack of standard solutions to process transliterated data hinders the extraction of useful information from this enormous amount of data. Therefore, an effective transliteration model for social media text can aid different text processing applications in extracting useful information.We are interested in exploring models for backward transliteration of social media text.
There are two approaches we can consider to analyze social media text (1) Build new text-mining models for each task trained over noisy text (2) Perform back-transliteration to convert this text to formal text. The former approach requires us to build and train new models for every task. The latter approach leverages the existing text mining models for text analysis and eliminates the need to build one for noisy text.

1.3 Challenges:

The social media data is noisy, which means that while writing a native language in Latin script, there may be different variations of a single word. For example "बच्चे " can be written as "bache" or "bachhe". The variations occur due to the following reasons:

- **Informal Transliteration:** While writing the native language in

Latin script, people do not follow a common standard, e.g., ITRANS, ISO, so decision for writing vowels and other sounds relies entirely on the users

- **Long Vowel transliteration:** Users write long vowel signs in different forms. For example, transliterating the word "पानी " to "pani" or "paani" one with a single a and another one with double a's.
- **Borrowed words' pronunciations:**Due to influence of foreign languages like the Persian language in Hindi while transliterating a foreign word, we rely on the pronunciation. For Example, we can observe both ("आजाद " ajaad) & ("आज़ाद ," azad) because some Persian sounds do not have any Hindi counterpart.
- **Accent and Dialect:**People also write based on their accent and dialect. For example, "शाम " can be transliterated as both "shaam" and "saam".
- **Non phonetic writing:** There may be variations because of multiple alphabets which may fit the corresponding Hindi character. For Example "मुकाम " could be transliterated as either "mukam" or "muqam", here क can be represented by both "k" and "q", since they both have same pronunciations.

• **Informal writing pattern:** These variations are due to informal and casual writing on social media platforms.

  - **Elongation:**Users elongate some words to express different expressions or moods. For example, coooool, gooood.
  - **Escaping vowels:** Users sometimes escape vowels to write faster. For Example, "महके " may be transliterated as "mhke."

Back-transliteration models provide an efficient way to clean the noisy social media data and transform it into a normalized form. The study is more critical for the development of different linguistic tools in the low-resource languages, specifically in the Indian context. However, the present state-of-the-art of transliteration for low resource Indian languages is in its initial stages.
Multilingual Transliteration in addition to the challenges faced by bilingual transliteration also has some additional challenges. Each language has its own collection of grapheme and phoneme sets. Every language has different set of rules to map grapheme to phoneme representation. Modelling them with the same shared architecture while preserving the specialized characteristic of every language needs deep understanding of the shared as well as specialized characteristics of languages. To implement this we require deeper understanding of the functions of each of the component of the architecture so that shared and specialized characteristic of languages are modelled by appropriate components.

1.4 Scope & Plan:

In this work, we are focusing on the problem of "back transliterating the text where the words belong to native Indian languages but are written in Latin script." In our study we studied Hindi, Gujrati, Kannada and Bangla.

- **Phase-I:** We focused on back transliterating Hindi text to Devanagari script from Latin script. We have done an extensive survey of different machine transliteration and translation models, starting from the basic architectures and extending to state of the art methods. We have performed experiments on the corpus, which simulates the social media data by taking motivation from the current researches. Finally, we have listed the future works for improving the current results.

- **Phase-II:** (1) Used transformer architecture to build a bilingual and Multilingual language transliteration model and compared RNN and Transformer (2) Investigated and implemented various parameter sharing approaches in multilingual transformer model valid for most Indian languages.

Chapter 2

Literature review

Earlier researches in machine transliteration were based on the phonetics of source and target languages, followed by statistical and language-specific methods.[11] The need for transliteration first arose as a subtask to translation. Transliteration and Translation are closely related, and to some extent, transliteration can be considered as a character-level translation problem[5]. Many of the works on language models are based upon machine translation, so understanding the major ideas behind machine translation and appropriately transferring them to solve the transliteration task seems to be a viable option. In this section, we will first discuss machine translation approaches, and then we will look at some of the research specific to machine transliteration.

2.1 Traditional approaches

Dictionaries and phonetic tools were used in early transliteration works to learn probabilistic mapping rules between languages [12] [23]. Later works implemented more reliable methods that do not require intermediate phonemic mapping and can learn direct orthographic mapping in a bilingual dictionary between two languages [8]. Traditionally, these works are modeled on the basis of the joint-sequence modeling technique common to Grapheme-to-Phoneme (G2P). An initial alignment between the corresponding graphema (word) and phoneme (pronunciation) sequences is learned, given word-to-pronunciation examples. On the matched tokens, a language model is then trained (see [3] for a detailed overview of the G2P joint-sequence models). The open source Phonetisaurus [19] has proven to achieve state-of-the-art scores on various G2P tasks [18]. Phonetisaurus implements the traditional EM-driven sequence alignment algorithm, with some restrictions, such as allowing only m-to-one and one-to-m alignments. It trains from the aligned pairs an n-gram language model, which is transformed to a Weighted Finite-State Transducer (WFST). Decoding can then be achieved by extracting the shortest path with the input word through the phoneme lattice produced by composition.

2.2 Phonetic-based Method

Many early transliteration research applied methods of speech recognition and researched transliteration within a system focused on phonetics. This family of methods is also referred to as pivot or phoneme-based in the literature.[11] The idea behind the phonetic-based method is that all languages have the same phonetic representation. Phonetic-based methods identify phonemes in the source word S and then map them to character representations in the target language to generate the target words. The words are converted to their phonetic representation using set transformation rules.[11] The main challenge of this approach is that many criteria decide the pronunciation of words in any language. For example, the location of words may decide how they are pronounced.

2.3 Sequence-to-Sequence Models

Seq2Seq models are encoder-decoder based machine translation models that maps an input of sequence to an output of sequence.
Though There are deep neural networks that have achieved a high performance on difficult tasks like voice recognition, computer vision, social network filtering, and machine translation. Despite this excellent performance, DNNs are only applicable to problems whose inputs and targets can be encoded with vectors of fixed dimensions. Thus it poses a significant limitation for sequential data since input & output is of variable length.

Sutskever et al.[24] has shown a straightforward application of Long Short-Term Memory (LSTM)[9] architecture which can solve general sequence to sequence problems. A useful feature of the LSTM is that it learns to transform a variable length input sentence into a representation of a fixed-dimensional vector. The idea is to use one LSTM to read the sequence of inputs to get a fixed-length vector and then use another LSTM to retrieve the sequence of output from that vector. These encoder-decoder models require the source sentence to be compressed into a fixed vector, which may give poor results for long sentences.

Bahdanau et al.[2] extended the encoder-decoder model to align and translate jointly. Each time the model generates a word in translation, it searches for a set of encoder outputs which are most relevant for generating the output on the corresponding position in decoder, this is called as attention mechanism. The model then uses the context vector based on these source positions and all previously predicted words to predict the target word. The distinguishing feature of this method from the basic encoder-decoder model is that instead of just using a fixed-length encoded vector, it chooses a subset of encoder outputs adaptively while decoding the translation. The improvement is more evident for longer sentences but can be observed for sentences of any length.

2.4 Fully Character-Level Language Models

The inability of word-level models to represent out-of-vocabulary words makes them unsuitable for low resource languages - also, the complexity of training and decoding increases as the size of the dictionary increases. Character-level models are more beneficial in following manner (1) No out-of-vocabulary issues (2) Rare morphological variants of a word are better represented (3) No need of explicit segmentation[4] (4) Easy to apply in multilingual translation settings (allows parameter sharing between languages with significant standard alphabets) (5) Not forcing model with explicit segmentation encourages it to better learn the internal structure of a sentence.

Lee et al. [15] proposed a fully character-level NMT model to generate a target sequence mapping from a given source sequence. The reduction in training time of a fully character-based encoder is achieved by substantially reducing the source sentence length using a stack of convolution, max-pooling, and highway layers. Max-pooling reduces the source vector representation allowing the model to learn at the speed of sub-word level models while preserving the local features. In the many-to-one translation task, by using universal encoder for all the language pairs, their model outperforms the sub-word level models without increasing the model size. They demonstrated the feasibility of a fully character-based language model in bi-lingual and multi-lingual settings.

2.5 Compositional Transliteration systems

Compositional transliteration systems refer to systems where multiple components of transliteration are composed to improve the quality of transliteration.[13] There are two forms of composition serial & parallel. By composing the transliteration engine between X&Y and Y&Z, transliteration functionality can be created in serial composition between two languages X&Z. Such compositions can be used when no parallel corpora exist between X&Z directly but have parallel data for X&Y and Y&Z. There are 22 recognized languages in India, for instance, but parallel information can be found between Hindi and any foreign language such as Spanish, but not for other Indian languages. In such cases, transliteration from Kannada to Spanish may be achieved by composing transliteration modules from Kannada to Hindi and Hindi to Spanish. This method leverages the existing resources and alleviate the need for developing parallel corpora for many language pairs. In parallel composition, the author explores different transliteration paths between X&Z to get better accuracy. In this approach, the availability of parallel corpus between 3 or more languages is leveraged to get more accurate transliterated results. They have used the Conditional Random Field-based transliteration model for training and further transliterating.
Serial Composition Model: For this model, three languages X, Y, Z, were selected on the basis of Weighted Average Entropy between X&Y and Y&Z, which is a measure of ease of transliteration. Top k outputs were taken from the X->Y transliteration engine and passed to the Y->Z engine, which in

turn takes the top 10 outputs based on the probability scores. This method is expected to give poor results since an erroneous result at an intermediate stage will result in incorrect result in the second stage, but on seeing the results and on doing the error analysis it was proved that even the erroneous result at intermediate stage might have enough information to get correct result in output. The dip in accuracy of composition models was very less as compared with the direct X->Z baseline model.
Parallel Composition Model: This model is possible if there is a parallel corpus available between the languages X, Y, Z. First, train an X→Z system, using the direct parallel names corpora between X & Z, which is called the direct system. Then an X→Y transliteration system, and second a fuzzy transliteration system, Y→Z that is trained using a training set that pairs the top-k outputs of the X→Y system. This is called a fuzzy system. The results from direct and fuzzy systems are combined to get final output for transliteration task of X->Z. This method improves the accuracy of the task.

This paper thus builds the composition models on already built state of the art transliteration models to improve the accuracy of the given transliterated task.

2.6 Normalization of Transliteration words in code-mixed data

The amount of code-mixed data (especially in the Indian context) in social media is increasing at a fast rate. A particular concern is needed to handle code-mixed data because of grammatical inconsistencies and phonetic variations. Although there are standard transliteration tools like ITRANS, ISO, it is unrealistic to use them. People transliterate different words differently based on their phonetic judgment.

Mandal et al.[17] focused on normalizing phonetic typing variation present in code-mixed data. Identical words are transliterated differently by different people based on their phonetic judgment. They proposed a two-step modular approach consisting of two-step normalization. The first module used the Seq2Seq Encoder-Decoder model to convert text into standard transliteration. The model is trained over a (Bn-En) Roman script dataset. Then the output from the first module was looked up in (BN_TRANS) dictionary. The word having the least Levenshtein distance is returned as a final normalized text. Model achieved a 90.27% accuracy on test data. It shows that a high accuracy can be achieved by using dictionary-based methods.

2.7 Exploring Multilingual models

NMT approach have proved to be very effective in transliteration tasks when compared to traditional statistical approaches. However, NMT method is limited in transliterating low resource language pairs because large amount of data is required in NMT models to get some useful mappings. If somehow we are able to leverage parallel data of different language pairs to to improve the accuracy of a single model, it can bring significant improvement in accuracy for low resource pairs. This approach of using multiple language pairs to jointly train our NMT models is called as Multilingual NMT. Since we are working with low resource Indian languages, so our main focus is now to explore multilingual transliteration methods

2.7.1 Multi-Task Learning for Multiple Language Translation

Dong et al. [6] investigated the problem of translating from single source to multiple target languages. Their solution uses sequence to sequence neural machine translation model [2] as the underlying model for their multilingual setup. This paper extends the neural machine translation to multi-task learning framework approach for one to many translation scenario, by sharing parameters across different languages on the encoder side. They have used a single encoder for all languages but have distinct attention mechanism and decoder specific to each target language. Through experiments authors have proved that this framework gives good results in both cases when parallel data is abundant or scarce. The basic assumption of this proposed solution is that languages may be different in grammatical and syntactical aspects but are similar orthographically. Since the encoder is shared, it helps in understanding the language in a better way and semantic representations that can not be discovered with just a limited amount of data can be learned. For example if we simultaneously translate English to Hindi and Bangla, Then the model learns about the common features shared by the two languages and gives a rich representation of the languages. The model training was optimised using mini-batch stochastic gradient descent but with a slight change i.e. they trained mini batches for a fixed language pair for some iterations before moving to next language pair.

2.7.2 Multi-Task sequence to sequence learning

Luong et al.[16] evaluated multi-task learning framework using attention free sequence to sequence model[24] for three scenarios. 1)one-to-many 2)many-to-one 3)many-to-many. One to many model is used when encoder can be shared due to similarity in source data, for example In machine translation and syntactical parsing where input is English words in both tasks but outputs are very different. Many to one model is used when decoder can be shared because of similarity in output, for example image captioning and machine translation, wherein both tasks the output is English words. Many to Many has multiple encoders and multiple decoders which is used for unsupervised goals and translation. This paper covers experiments of large tasks along with 1)small tasks(Penn Tree Bank parsing), For example, parsing tasks which maps English words to sequence of parsing tags shares the same encoder with English-German translation task. This is a one-to-many scenario. It resulted in +8.9 F1 score over baseline for parsing task. 2)medium size tasks(image caption generation), For example image captioning task outputs a sequence of English words thus it can share decoder with English-German translation task. It falls under many-to-one category and resulted in improvement in +0.7 BLEU score for translation task. 3) unsupervised task (auto-encoders and skip-thought vectors), Skip-thought vectors is the name given to fixed-size representation given to sentences using a neural based model with unsupervised training. Many-to-Many approach was used to couple English-German translation task with two other unsupervised tasks(learning autoencoders for German and English). This setup improved the BLEU score for translation by +0.5. Thus they concluded that Multi-Task framework can improve the accuracy of attention-free sequence to sequence model.

2.7.3 Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation

Johnson et al.[10]have proposed a very simple model for Many-to-Many scenario using bilingual sequence-to-sequence model proposed by Bahdanau et al.[2]. This solution requires no change except they have added a token in front of source word which is representative of target language for example(<2es>, target language Spanish), rest all the encoders, decoders, attention remains same. Since no change have been made to model thus extending to other language pairs is easy because we just have to add a token of target language and append that parallel data to the current multilingual corpora. The source language is not specified by token, thus ambiguity may arise in translation if same words from two different language have different meanings, but it can be handled by code switching. In most cases context provides enough evidence for translation than just relying on tokens.[10]. It was observed that when language pair with abundant data is trained along with data scarce language pair, then accuracy for low resource language pair is significantly improved. In addition to improving the accuracy of translation this model also provides bridging capabilities by translating between languages pairs which were never seen during training(zero shot translation). For example if Multilingual model was trained on English-French and French-German pairs then it gave good accuracy for English-German pairs also which was comparable to accuracy they got after training direct model for English-German pairs.

2.7.4 Multilingual Neural Transliteration

Most of the machine transliteration models are focused on bilingual training. There are many phonetically similar languages, so to leverage these similarities, multitask training approaches make sense. The aim is to (1) Jointly train multiple languages to favor superior transliteration results. Parameter sharing across phonetically similar languages can help learn richer patterns. (2) Leverage dataset of other languages for excellent training. Kunchukuttan et al. [14] proposed a multilingual transliteration models for training orthographically similar languages. Two languages are considered orthographically similar if they have: (i) immensely overlapping phoneme sets, (ii) similar grapheme to phoneme mappings, and (iii) mutually compatible orthographic systems.

They proposed an attention-based neural encoder-decoder model. An encoder is based on CNN instead of traditional bidirectional LSTM layer keeping in mind that most of the temporal dependencies are local in the transliteration task. The CNN based model is much faster to train with a slight decrease in accuracy. The decoder is a single layer LSTM network. Encoder and decoder are shared across the source and target languages, respectively. The output layers(fully connected feedforward layer), which are specific for every target language, are used to generate target words from the decoded output. The model offers maximum parameter sharing across language pairs to leverage orthographic similarity effectively while supplying sufficient space to learn language-specific features.

Quantitative observations. (1) Better transliteration accuracy than bilingual training. Arabic and Slavic are most benefitted. (2) Notable advancement over bilingual systems with phrase-based SMT. This result is consistent with previous works. (3) The better generalization of transliteration task as verified by zero-shot transliteration. Qualitative observations (1) Major decrements in vowel errors (average decreases 20%) (2) Less confusion between consonants having similar phonetic properties (3) Canonical spellings are preferred over alternative spellings with similar phonetics.

Further for languages having one to one character-phonemes correspondence, eg. Indic-English and Indic-Indic, the results on these languages are observed to be further enhanced when phonetic embeddings are given as input to the encoder instead of traditional embeddings generated by one-hot character vector. The phonetic embedding is generated by the product of the phonetic feature vector and weight matrix.

2.7.5 Parameter Sharing Methods for Multilingual Self-Attentional Translation Models

Sachan et al.[22] have explored various ways of sharing parameters but parameter sharing is not always good and often leads to a decrease in accuracy because it is not capable of representing the very different target languages through shared encoder or decoder. It helps in improving the BLEU score when target languages belong to a similar family. We have models like shared encoder and separate decoder[6] (high memory usage and high training time), or shared encoder and shared decoder[10](decoder's ability to model multiple languages can be significantly reduced). This paper presents a hybrid approach where shared encoder and multiple decoders but some of the parameters are shared across decoders which takes advantage of parameter sharing along with providing flexibility of decoding multiple languages. They have used the self attention Transformer Model[25] for translation and experimented while sharing parameters on different levels like attention weights, linear layer weights, or embedding weights. The objective of parameter sharing is two-fold: 1)eliminates the need of storing model for every language pair 2)effective for low resource languages since multiple data-sets are combined for a single model.

2.7.6 Multi-Way, Multilingual Neural Machine Translation with a Shared Attention Mechanism

This paper proposes[7] a approach for translation between multiple language pairs using multiple encoders and multiple decoders for each language in source and target side, but the amount of parameters only increases at a linear rate with respect to the number of languages. This is possible due to a shared attention mechanism[2] which is common for all the language pairs. In earlier work by Luong et al.[16] they have used a sequence to sequence without attention[24]. The most naive translation approach for multiple language pairs is using separate attention mechanism as used in work by Dong et al.[6]. If we were to use this approach here then parameters will grow on a quadratic rate with the number of languages. The authors proposed a very flexible approach where encoder and decoder specific to each language can have their own hyper parameters and used a single attention mechanism for all language pairs. This creates a need to transform the hidden states from encoder and decoder to undergo transformation to become compatible with the shared attention layer. It was made possible by using a shared dense layer for this transformation. These attention parameters we get after training over multiple language pairs can be used for transfer learning for low resource languages.

2.7.7 Transfer Learning for Low-Resource Neural Machine Translation

This paper[26] proposed a transfer learning based method which significantly improved the BLEU scores for low resource language. The idea is to train a parent model for language pairs whose data is abundant, then we can use those weights to train our model over low resource language pairs by fixing some of the parameters from the parent model. Transfer learning is justified by the fact that we need a good distribution over models which we do by training over large dataset. When model is retrained over low resource pair, some parameters that are likely to be used across different tasks are kept fixed. The authors have tried this model for different parent languages and got better results when parent and child languages were more similar.

2.8 Summary

In this work, we have reviewed existing works on machine transliteration. Though, phoneme based or statistical models were used earlier, most of the recent models use neural network based approaches. In the later part of our research we have mainly explored multilingual translation which have shown promising results for low resource languages. Recent research works have used various architecture for sequence-to-sequence models, While most of the research papers have used sequence-to-sequence models[2][24], some others have used advanced Transformer architecture also. We now aim to leverage the ideas of machine translation to solve our transliteration task. However, since only limited works were found for Multilingual transliteration, therefore we think there is a lot of scope for investigation and research in this area.

Chapter 3

Proposed Method

We are considering three language pairs (En-Hindi, En-Bangla, En-Gujrati) for training our Multilingual models. We have appended a token in both source and target words to identify the target language. We are proposing two methods for multilingual transliteration 1)one-to-many model[6] 2)fully shared model[10]. We combine alphabets of all the target languages into a single set and assign a unique integer to each of the character. We do the same character to integer mapping on the source side as well and map each word to an array of integer ("$mera@««" -> [1,15,7,20,3,2,0,0,0,0], padding character "<" is mapped to 0) using this character to integer mapping and then pass representation this array to embedding layer. We are using a Transformer as the base architecture for our transliteration task. We can see from Figure 3.1, a transformer as a transliteration black box which takes word as a input and outputs it's transliterated form. Transformer has an encoding and a decoding component. Encoding side takes some input (x1,x2,...xn) and generates a output (z1,z2,...zn) which is passed to the decoder which outputs (y1,y2,...yn).

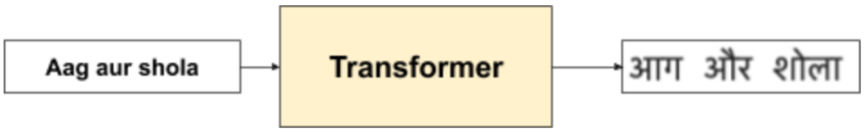

Figure 3.1: Input-output

.

3.1 Transformer Network

The architecture of the Transformer system follows the so called encoder-decoder paradigm, trained in an end-to-end fashion. Without using any recurrent layer, the model takes advantage of the positional embedding as a mechanism to encode order within a word. The transformer as shown in Figure 3.2 consists of an encoder, a decoder unit and connections among them. The encoder aims to represent the the source sentence into a representation which will contain all the positional information and dependency information as well which decoder will use to generate the character sequence in the target language.

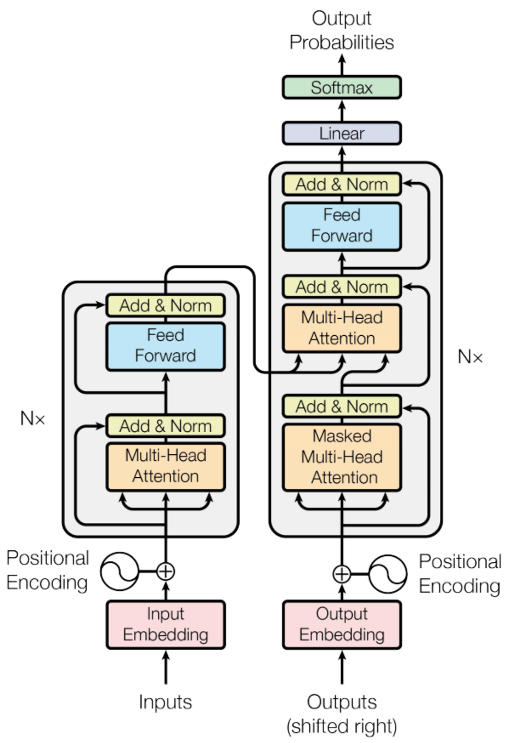

Figure 3.2: The Transformer architecture (illustration from Vaswani et al. (2017)[25]).

3.1.1 Embedding: d_model=128

We pass each word represented as an array of integers to a pre-trained tensorflow embedding layer(tf.keras.layers.Embedding) to generate embedding for each character in the word. Now we get a 2-D tensor (word_length, d_model) which is passed as input to the encoder.

3.1.2 Positional Encoding

Since the model has no recurrence, we add positional encoding as shown in Figure3.2 to the output of the embedding layer to help model learn about relative positions of characters in a word. In this work, we use sine and cosine functions of different frequencies. The formula for calculating the positional encoding is as follows:

$$PE(pos, 2i) = sin(pos/10000^2 i/d_{model})$$

$$PE(pos, 2i + 1) = cos(pos/10000^2 i/d_{model})$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid.

3.1.3 Encoder

The Encoding component consists a stack of N=4 identical encoder layers. Each encoder layer has 2 sub layers 1)Multi-head Self-Attention layer(num_heads(h) = 8). 2) Point wise feed forward layer. Self-attention is called "Scaled Dot-Product Attention" (left part of Figure 3.3)[25]. The input consists of queries Q and keys K of dimension d_k, and values V of dimension d_v, which are of dimension d_{model} for single head attention. We compute the dot products of the query with all keys, divide each by $\surd d_k$, and apply a softmax function to obtain the weights on the value.

$$Attention(Q, K, V) = softmax(QK^T/\surd d_k)V$$

. It is beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively[25]. The queries, keys and values are linearly projected h times(right part of Figure 3.3), to allow the model to jointly attend to information from different representation, concatenating the result.

$$MultiHead(Q,K,V) = Concat(head_1, ..., head_h)W^O$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

with parameter matrices

$$W_i^Q \in R^{d_{model} \times d_k}, W_i^K \in R^{d_{model} \times d_k}, W_i^V \in R^{d_{model} \times d_v} and W^O \in R^{h_{d_v} \times d_{model}}$$

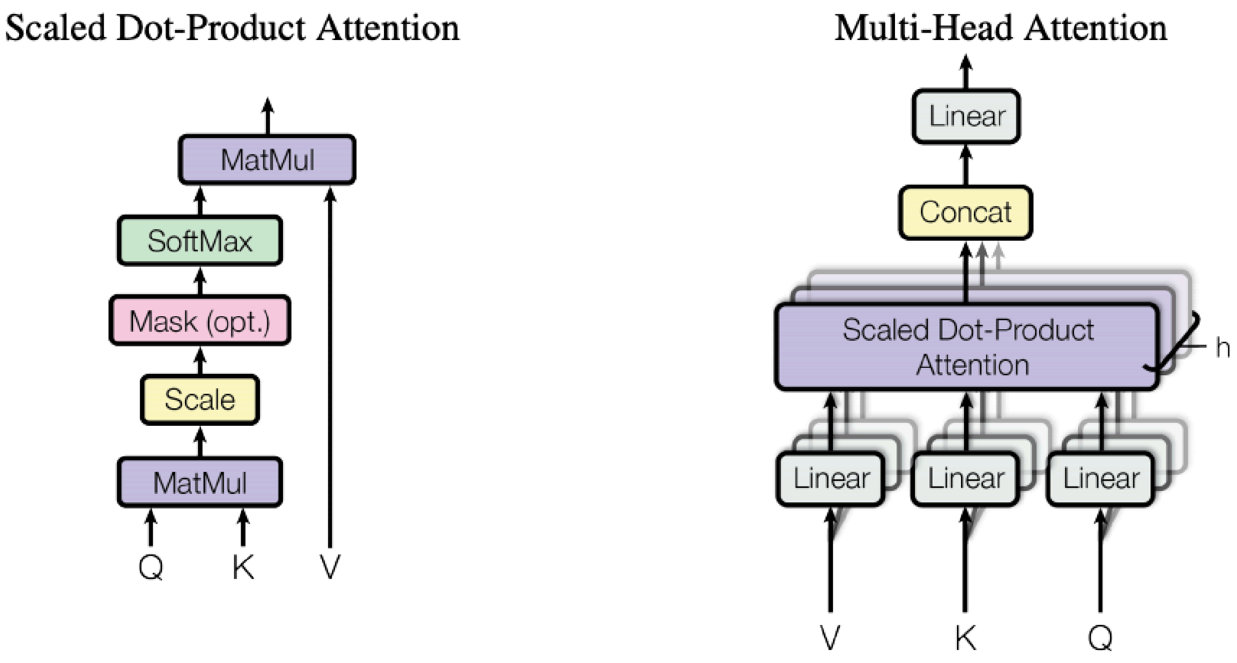

Figure 3.3: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.[25]

2)From Figure 3.2 we can see that Point wise feed forward layer takes concatenated input of all heads from self-attention layer as input. After each of the sub layer a residual network and a normalization is added. To make data compatible to residual connection, outputs from all layers including embedding layer are of dimension d_model. First encoder layer takes input from the embedding layer, rest all of encoder layers take output of previous encoder layers as their input.

3.1.4 Decoder

The Decoding component consists of stack of decoders. Here also we have 4 decoder layers. Each decoder layer has three sub-layers 1)encoder-decoder attention layer in which each position of the decoder attends to all positions in the last encoder layer 2) Multi-head Masked self-attention layer, in which each position attends to all previous positions including the current position. and 3)Point wise feed forward layer. Each of these sub-layers have a residual connection and a normalization layer of top of them. While training we use the concept of teacher forcing i.e. instead of using output from decoder to predict the next character we give the correct character as input to predict the next character.

3.1.5 Loss Function

We compute the corssentropy loss between the labels and predictions using tf.keras.losses.SparseCategoricalCrossentropy API. Since the target sequences are padded, it is important to apply a padding mask when calculating the loss.

3.2 Model Variation

To be able to use multilingual data we append a token which represents the target language in which we have to trasliterate the word. For example consider English-Hindi pair: hamara -> हमरा . It is converted to $hamara@ -> $हमरा @ We don't have to specify source language because it is English for all language pairs. After adding token we train the model with all the multilingual data.

3.2.1 One-to-many: shared encoder and multiple decoders

We have a shared encoding unit for all language pairs but distinct decoders for each target language(Figure 3.4). We pass the output from encoder to the corresponding decoder by checking the token of the input word and use that token as a initial input character for decoder to generate subsequent outputs.

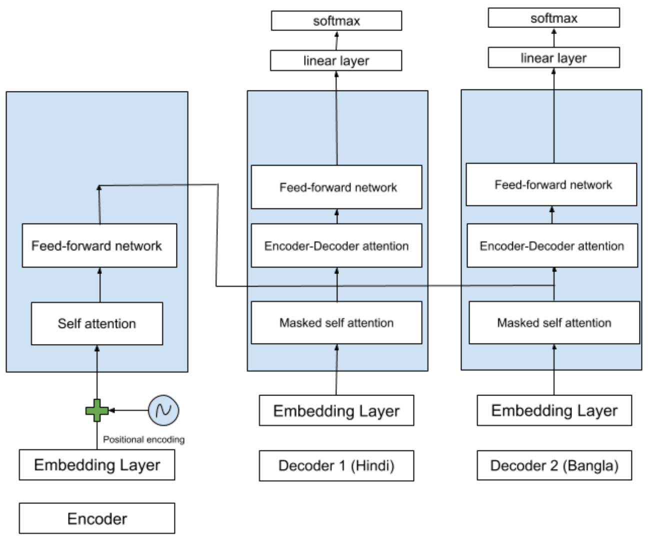

Figure 3.4: Block diagram illustrating one-to-many multilingual transliteration task

3.2.2 Fully shared: shared encoder and shared decoders

We have a single encoder and single decoder but to differentiate between two different language pairs we append a token in source word to specify the target language for example(for En-Hi pair we append “$” with the English word and append “% for En-Bangla pair” Figure 3.5). We pass the token as a initial input character for decoder to generate subsequent outputs.

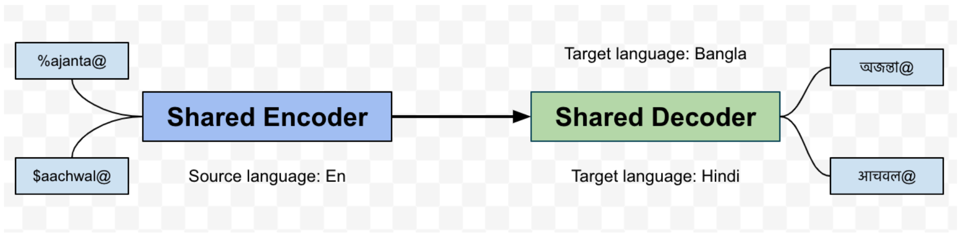

Figure 3.5: Shared encoder and decoder

Chapter 4

Experiment Setup

4.1 Training and Batching

4.1.1 Dataset

We have used "Datasets for FIRE 2013 Track on Transliterated Search," which has 30,000 English-Hindi Pairs for training and 500 English-Gujrati pairs. Hindi word transliteration pairs contain annotations collected from different users in multiple setups - chat, dictation, and other scenarios. We have also explored our model on NEWS dataset for two language pairs(En-Hindi, En-Bangla), which consisted of both word level and sentence level data. It consisted of seperate train and test data of size 13000 and 1000 pairs respectively for both language pairs.

4.1.2 Data preprocessing

We have appended a special token for all three different language pairs(i.e. $ for Hindi, % for Gujrati and # for bangla), for example, the word "Khushboo" having target language Hindi is converted to "$Khushboo". News dataset is very noisy with many out-of the vocabulary characters like "?,.,!,2" etc. We have also replaced all these punctuation and numbers from the sentences with empty string and then merged these datasets for different language pairs in to a single dataset. For fully shared model we shuffled this dataset and form batches of size=64 for batch training. In one-to-many model we create batches where all pairs in a batch belonged to a single language pair. We then shuffled these batches to achieve uniformity in training with respect to the language pairs. We have limited the data to only with input length less than or equal to 25.

4.1.3 Optimization

We have used a mini-batch Stochastic Gradient Descent (SGD) algorithm with Adam-optimizer with $\beta 1 = 0.9$, $\beta 2 = 0.98$ and $\epsilon = 10{-}9$ with a custom learning rate scheduler as in Vaswani et al. [25]

$$lrate = d_{model}^{-0.5} * min(step\_num^{-0.5}, step\_num * warmup\_steps^{-1.5})$$

. We use $warmup\_steps = 4000$. Each SGD update direction is computed using a mini-batch of 64 pairs. We trained each model for 50 EPOCHS.

4.1.4 Regularization

Residual Dropout: We apply dropout to each sub-layer's output before it is added and normalized to the input sub-layer[25]. Additionally, we apply dropout in both the encoder and decoder layers to the sums of the embedding and the positional encoding. For the base model, we use a rate of $Pdrop = 0.1$.

4.2 Model Variation

4.2.1 RNN - Encoder-Decoder with attention[2]

For this architecture we have experimented only for bilingual models with English-Hindi pairs. We have used LSTM cells in our model. In all models, we have used a Unidirectional decoder. The Encoder-Decoder with attentions mechanism models have the following variations :

1. Unidirectional single layer Encoder
2. Bidirectional single layer Encoder
3. Unidirectional Multi-layer Encoder, Decoder

4.2.2 Transformer Model

1. Bilingual (English-Gujrati(Fire), English-Hindi(Fire,News), English-Bangla(News))
2. Multilingual model with fully shared encoder and decoder(English-Hindi(News) and English-Bangla(News))
3. Multilingual model with shared encoder and multiple decoders(English-Hindi(News) and English-Bangla(News))

4.3 Result

We have used three metrics[21] to analyze our results.

1. Word accuracy: words correctly predicted/total words predicted
2. Character accuracy:characters correctly predicted/total characters in all words

3. Average character mismatch: average count of characters correctly predicted in a word

4.3.1 Fire Dataset

RNN: Encoder-Decoder with attention

Bilingual (English-Hindi)			
Model	word accuracy	character accuracy	avg. mismatch
Unidirectional encoder	51.3544201	92.9746489	**0.108467**
Bidirectional encoder	52.87915	94.1402	0.11
Unidirectional Multi-layer	**55.8637**	**94.7373**	0.17462

Table 4.1: Accuracy comparison between different variations of RNN model trained over English-Hindi pairs.

Results in Table 4.1 shows that increasing a layer results in much better better accuracy than just making it bi-directional. However, the average mismatch is much less in the unidirectional model. It suggest that if the bidirectional or multi layer produce wrong word, then the word had much more error as compared to the wrongly predicted word from the unidirectional model. But overall multi-layer and bidirectional model produced less wrong words.

Transformer: Self-attention

From Table 4.2 we observed that multilingual model was able to improve accuracy of English-Gujrati pair in every metric, which suggest that low-resource language was able to benefit from the resource-rich language Hindi. Except word accuracy all other metrics improved for the English-Hindi pair as well. These results motivates us to create better multilingual models

Bilingual			
Model	word accuracy	character accuracy	avg. mismatch
English-Hindi	**60.12**	94.63	0.13
English-Gujrati	27.27272	73.57	0.16
Multilingual: fully shared			
Model	word accuracy	character accuracy	avg. mismatch
English-Hindi	59.415	**95.165**	**0.099**
English-Gujrati	**44.0**	**88.88**	**0.112**

Table 4.2: Accuracy comparison between Bilingual and Multilingual(fully shared) Model trained over English-Hindi and English-Gujrati transliteration pairs.

which can help in transliteration of both resource scarce as well as resource rich language.

4.3.2 NEWS Dataset

Transformer: Self-attention

Bilingual			
Model	word accuracy	character accuracy	avg. mismatch
English-Hindi	**40.7558**	**91.92307692**	0.083
English-Bangla	41.2	91.4173	0.09
Multilingual: fully shared			
Model	word accuracy	character accuracy	avg. mismatch
English-Hindi	39.6322	91.1846	**0.0336**
English-Bangla	42.2	**92.68**	**0.029**
Multilingual: shared encoder and multiple decoder			
Model	word accuracy	character accuracy	avg. mismatch
English-Hindi	40.3688	90.979	0.0344
English-Bangla	**42.80**	92.125	0.031

Table 4.3: Accuracy comparison between bilingual and multilingual(fully shared and multiple decoders) transformer models trained over English-Hindi and English-Bangla transliteration pairs.

From Table 4.3 we observed that, as expected accuracy of English-Bangla pairs is better on the multilingual model as it is benefitting from the representation learnt from English-Hindi pairs. We can also observe that fully shared model and multiple decoders model have comparable accuracy metrics. Multiple decoder model has better word accuracy while the other two metrics are better for completely shared model. This give us hope on existence of a better hybrid architecture (where some parameters of decoder are shared) having better accuracy than both them. For English-Hindi, word and character accuracy are better for bilingual data. Although the accuracy are comparable with multilingual model, it does indicates a limitation of multilingual model which we also observed in Table 4.2. As the multilingual model try to capture more generalized feature among the languages, the models we experimented with give similar results as bilingual model for resource-rich languages and there is a need to employ special techniques and frameworks to make it suitable for resource-rich transliteration as well.

4.4 Observation

1. We can infer from Table 4.2 that Transformer outperforms the RNN based model with great margin on English-Hindi pairs in Fire Dataset.
2. Word accuracy and character accuracy were comparable in NEWS dataset for all three models(Table 4.3)(i.e. Bilingual, Multilingual: fully shared, Multilingual:multiple decoders)
3. Time involved for training in Transformer for 50 epochs was roughly 500 seconds whereas time in RNN model for 5 epochs was 30 mins.
4. Word accuracy is very less as compared to character level accuracy because even if a single character is wrong, then that word is considered

as incorrect.

5. In Fire dataset we only had 500 pairs for English-Gujrati data. From the above observations we observed a 62% increase in accuracy for English-Gujarati language pairs(Table 4.2). This shows that if we train a low resource language with a resource abundant language then it improves the accuracy significantly.

6. The average mismatch for all the models which is around 0.1. Most of the words have lengths in range of 10-20, so, on average, each word has 2 mismatch characters.

7. Most of the mismatch pairs were the ones which have similar pronunciations for e.g. '**(क ख), (ग घ), (द, ड)** .'

8. Other error are between the Hindi “maatras” are also known as vowel signs for example '**(आ अ), (◌ी ि◌), (◌ू ◌ु)** '.

Chapter 5

Conclusion & Future Work

In this work, we have summarized the performance and ideas behind different machine transliteration models along with the challenges in the field. We also presented observations and results recorded by us while implementing various RNN models and Transformer models.

We conclude that Transformer is clearly a better architecture for our task both in terms of accuracy and efficiency. Our results on Fire dataset were significantly improved when switched from RNN to Transformer architecture. In adition to that we can get comparable results from Transformer with much less epochs.

From the above experiments we have also inferred that Multilingual setup can help us overcome limitations of Neural Machine Transliteration models [24] [2] for low resource languages. Low resource pairs have given comparable results for multilingual models as bilingual models which proves that if we have scarcity of data for a particular language then it can be tackled by training this transliteration models with other language pairs which are orthographically similar.

We observed that high-frequency mismatch pairs occur for phonetically similar Hindi characters, e.g. (त, ट).In the future, we aim to reduce the number of mismatch pairs. One of the approaches is to pass the predicted word through a filter. The filter will first mark the characters of the output of our transliteration model, which have the high probability of getting mismatched and construct all possible variations of that word by replacing the marked character with their mismatch counterparts. This approach will give us a set of probable output words. We can apply a dictionary lookup for these words to get the most appropriate word.

Bibliography

[1] Digital trends 2019: Every single stat you need to know about the internet. `https://thenextweb.com/contributors/2019/01/30/digital-trends-2019-every-single-stat-you-need-to-know-about-the-internet/`. Accessed: 2019-10-30.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Maximilian Bisani and Hermann Ney. Joint-sequence models for grapheme-to-phoneme conversion. *Speech communication*, 50(5):434–451, 2008.

[4] Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio. A character-level decoder without explicit segmentation for neural machine translation. *arXiv preprint arXiv:1603.06147*, 2016.

[5] Thomas Deselaers, Saša Hasan, Oliver Bender, and Hermann Ney. A deep learning approach to machine transliteration. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 233–241. Association for Computational Linguistics, 2009.

[6] Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1723–1732, 2015.

[7] Orhan Firat, Kyunghyun Cho, Baskaran Sankaran, Fatos T Yarman Vural, and Yoshua Bengio. Multi-way, multilingual neural machine translation. *Computer Speech & Language*, 45:236–252, 2017.

[8] Li Haizhou, Zhang Min, and Su Jian. A joint source-channel model for machine transliteration. In *Proceedings of the 42nd Annual Meeting on association for Computational Linguistics*, page 159. Association for Computational Linguistics, 2004.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. Google' s multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351, 2017.

[11] Sarvnaz Karimi, Falk Scholer, and Andrew Turpin. Machine transliteration survey. *ACM Comput. Surv.*, 43:17, 04 2011.

[12] Kevin Knight and Jonathan Graehl. Machine transliteration. *Computational linguistics*, 24(4):599–612, 1998.

[13] A Kumaran, Mitesh M Khapra, and Pushpak Bhattacharyya. Compositional machine transliteration. *ACM Transactions on Asian Language Information Processing (TALIP)*, 9(4):13, 2010.

[14] Anoop Kunchukuttan, Mitesh Khapra, Gurneet Singh, and Pushpak Bhattacharyya. Leveraging orthographic similarity for multilingual neural transliteration. *Transactions of the Association for Computational Linguistics*, 6:303–316, 2018.

[15] Jason Lee, Kyunghyun Cho, and Thomas Hofmann. Fully character-level neural machine translation without explicit segmentation. *Transactions of the Association for Computational Linguistics*, 5:365–378, 2017.

[16] Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

[17] Soumil Mandal and Karthick Nanmaran. Normalization of transliterated words in code-mixed data using seq2seq model & levenshtein distance. *arXiv preprint arXiv:1805.08701*, 2018.

[18] Lili Mou, Zhao Meng, Rui Yan, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. How transferable are neural networks in nlp applications? *arXiv preprint arXiv:1603.06111*, 2016.

[19] Josef R Novak, Paul R Dixon, Nobuaki Minematsu, Keikichi Hirose, Chiori Hori, and Hideki Kashioka. Improving wfst-based g2p conversion with alignment constraints and rnnlm n-best rescoring. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[20] Dinesh Kumar Prabhakar and Sukomal Pal. Machine transliteration and transliterated text retrieval: a survey. *Sādhanā*, 43(6):93, 2018.

[21] Mihaela Rosca and Thomas Breuel. Sequence-to-sequence neural network models for transliteration. *arXiv preprint arXiv:1610.09565*, 2016.

[22] Devendra Singh Sachan and Graham Neubig. Parameter sharing methods for multilingual self-attentional translation models. *arXiv preprint arXiv:1809.00252*, 2018.

[23] Bonnie Glover Stalls and Kevin Knight. Translating names and technical terms in arabic text. In *Computational Approaches to Semitic Languages*, 1998.

[24] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[26] Barret Zoph, Deniz Yuret, Jonathan May, and Kevin Knight. Transfer learning for low-resource neural machine translation. *arXiv preprint arXiv:1604.02201*, 2016.