GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

DATA TRANSLITERATION FROM HINDI TO ENGLISH

A Report for the Evaluation 3 of Project 2

Submitted by

AZEEM AHMAD
(1613112014/16SCSE101893)

In partial fulfilment for the award of the degree of

Bachelor of Technology

IN

Computer Science and Engineering

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

UNDER THE SUPERVISION OF

DR. D. GANESH GOPAL (Professor)

APRIL/MAY-2020

GALGOTIAS UNIVERSITY
(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this project report "**TRANSLITERATION FROM HINDI TO ENGLISH**" is the Bonafide work of "**AZEEM AHMAD(1613112014)**" who carried out the project work under my supervision.

Signature of Head

School of Computer Science and Engg.

Signature of Supervisor

Dr. D. Ganesh Gopal

Professor
School of Computer Science and Engg.

<u>ABSTRACT</u>

Machine learning models can achieve high performance and accuracy these days due to the large amount of research and data available to data scientists. Transliteration is a key component of machine translation systems and software internationalization. With increasing globalization, information access across language barriers has become important. Given a source term, machine transliteration refers to generating its phonetic equivalent in the target language. This is important in many cross-language applications. This project explores Hindi to English transliteration. We will be working on algorithms like Sequence to Sequence modelling, RNN. Here we will be working to transliterate Hindi words to corresponding English words. First, the vocabulary of the dataset is created and then it is embedded and provided to the encoder and then RNN algorithm is applied on the data to transliterate the input.

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

Transliteration allows people to read and understand articles without the need to become proficient in a new script or language. Transliteration is particularly used by libraries for the processing of textual data eg in messaging.There are words that do not need to be translated as they remain same in all the languages like names of person, place, medicines, terms used in sports etc. These entities are known as “Named Entities” and remain the same whatever be the language and conserve their phonetics. Our projects will be designed to perform data transliteration from Hindi to English on word level . A dataset will be given on which our model will be trained to perform following input and output.

Input Text	**Transliterated Output**
तेरा	Tera
नीला	Neela
लड़की	Ladki
दुनिया	Dunia

Transliteration is utilized when a word or phrase must be conveyed in a language with a different writing system. For example, when you go to a Chinese restaurant, the menu might feature Chinese characters that you don’t understand. When those characters are transliterated, they approximate the Chinese word’s pronunciation using Latin letters. If you can’t read or speak Chinese, you still won’t understand the transliterated language. Only when that Chinese word on the menu is translated into English will you be able to comprehend it.

For instance, let’s take the Chinese word 面条. If you just wanted 面条 transliterated it would be mein (as in the Chinese menu item lo mein). Mein does not tell you what the original word means in English, but it does help you pronounce it the way a Chinese speaker would. If you wanted to translate the word it would be noodles.

1.1 Challenges

Due to lack of standardization in transliteration, a single Hindi word can have multiple surface forms (e.g. Humara, Hamara, Hamaaraa etc.). Some Hindi words can take the same surface form as an English word. The words ’hi’ (an auxiliary verb), ’is’ (this), and ’us’ (that) are some examples.

Our extracted datasets contain a diverse set of names from different origins, including many names with different linguistic conventions. Names can be pronounced differently depending on origin and context can play a role as well. One other challenge is that the models also fail on common names.

A source language word can have more than one valid transliteration in target language. For example, for the Hindi word below four different transliterations are possible: गौतम - gautam, gautham, gowtam, gowtham Therefore, in a CLIR context, it becomes important to generate all possible transliterations to retrieve documents containing any of the given forms. Transliteration is not trivial to automate, but we will also be concerned with an even more challenging problem going from English back to Hindi, i.e., back-transliteration. Transforming target language approximations back into their original source language is called back-transliteration. The information-losing aspect of transliteration makes it hard to invert. Back-transliteration is less forgiving than transliteration. There are many ways to write a Hindi word like मीनाक्षी (meenakshi, meenaxi, minakshi, minaakshi), all equally valid, but we do not have this flexibility in the reverse direction.

2. Existing Systems

G.S.Josan((2011)) first used a base line method as a character to character matching approach and then compared it with a statistical method for transliteration. They used a Noisy channel model for the purpose. They also concluded that their system can be improved by using some tuning in the language model in terms of alignment heuristics, maximum phrase length etc. and by defining a better syllable similarity score.

S.Reddy,(2009)a substring based transliteration model and used conditional random fields (CRF) sequential model which use substrings as the basic token unit and pronunciation data as the token level features. They considered source and target language strings as non-overlapping substring sequences. For alignment they have used Giza++ toolkit. They trained the system for English to Hindi, English to Tamil and English to Kannada transliteration and got accuracy of 41.8%, 43.5% and 36.3% respectively.

T.Rama,(2009)a phrase based translation problem for English to Hindi transliteration and used Moses and Giza++. In case of transliteration, phrases are basically the letters of the words. The authors varied the maximum phrase length from 2-7 and changed the order of language model from 2-8 and observed that on training the language model on 7-gram and using alignment heuristic grow-diag-final gives the best results. They got an accuracy of 46.3%.

V.B.Sowmya,(2009)a transliteration based method for typing Telugu using Roman script. They have used Edit-distance based approach using Levenshtein Distance and considered three Levenshtein distances : Levenshtein distance between the two words, between the consonant sets of the two words and between the vowels set of the two words They have concluded that Levenshtein distance gives good results because of the relation between Levenshtein Distance and nature of typing Telugu using English. They used three databases: general database, countries and place names and person names.

V.Goyal,(2009)a rule based approach for transliteration from Hindi to Punjabi. With the character level mapping of Hindi and Punjabi the authors define approximately 55 rules for transliteration and got an accuracy of 98%.

A.Finch,(2008)phrase based techniques of machine translation for transliteration of English to Japanese words for speech to speech machine translation system. They expressed transliteration as a character level machine translation problem and achieved correct or phonetically equivalent correct words in approximately 80% of cases.

H.Surana,(2008)transliteration from English to Hindi and English to Telugu is done by authors using mapping and fuzzy string matching. Firstly, authors detected the origin of a word in terms of Indian / Foreign word. For foreign words, they mapped English Phonemes to letters of Indian Language script. For Indian words, they mapped Latin segments of the words to Indian language letters or to a combination of letters and then used fuzzy string matching for final transliteration and got a precision of 80 % for English-Hindi and 71% for English-Telugu.

T.Sherif,(2007)substring based transliteration from Arabic to English text. They implemented the method using dynamic programming and finite state transducers. They evaluated four approaches - a deterministic mapping algorithm (base-line method); a letter based transducer; Viterbi substring decoder with obtained optimal substring length as 6; and substring based transducer with obtained best length of substring as 4. The authors then compared results of all these four methods with a fifth approach, viz., manual transliterator. They concluded that substring based transliteration gives better results.

P.Pingali,(2006)cross-language retrieval from Hindi and Telugu to English language was done with translations. Authors also used transliteration for proper names and non- dictionary words. They used phoneme mapping, metaphone algorithm and Levenshtein’s approximate string matching for transliteration.

J.H.Oh,(2002)transliteration of English words to Korean words, authors used phonetic information (phoneme and context) and orthographic information for transliteration. They divided English words into two categories - pure English words and those with Greek origin and found that usually pure English words can be transliterated using phoneme and English words with Greek Origin can be transliterated using character matching. After dividing the words in two categories on the basis of origin (E or G) they converted English phonemes to Korean alphabet. They claimed that their results show an increment of about 31% in word accuracy in comparison to previous works for transliteration.

3. Proposed Methodology

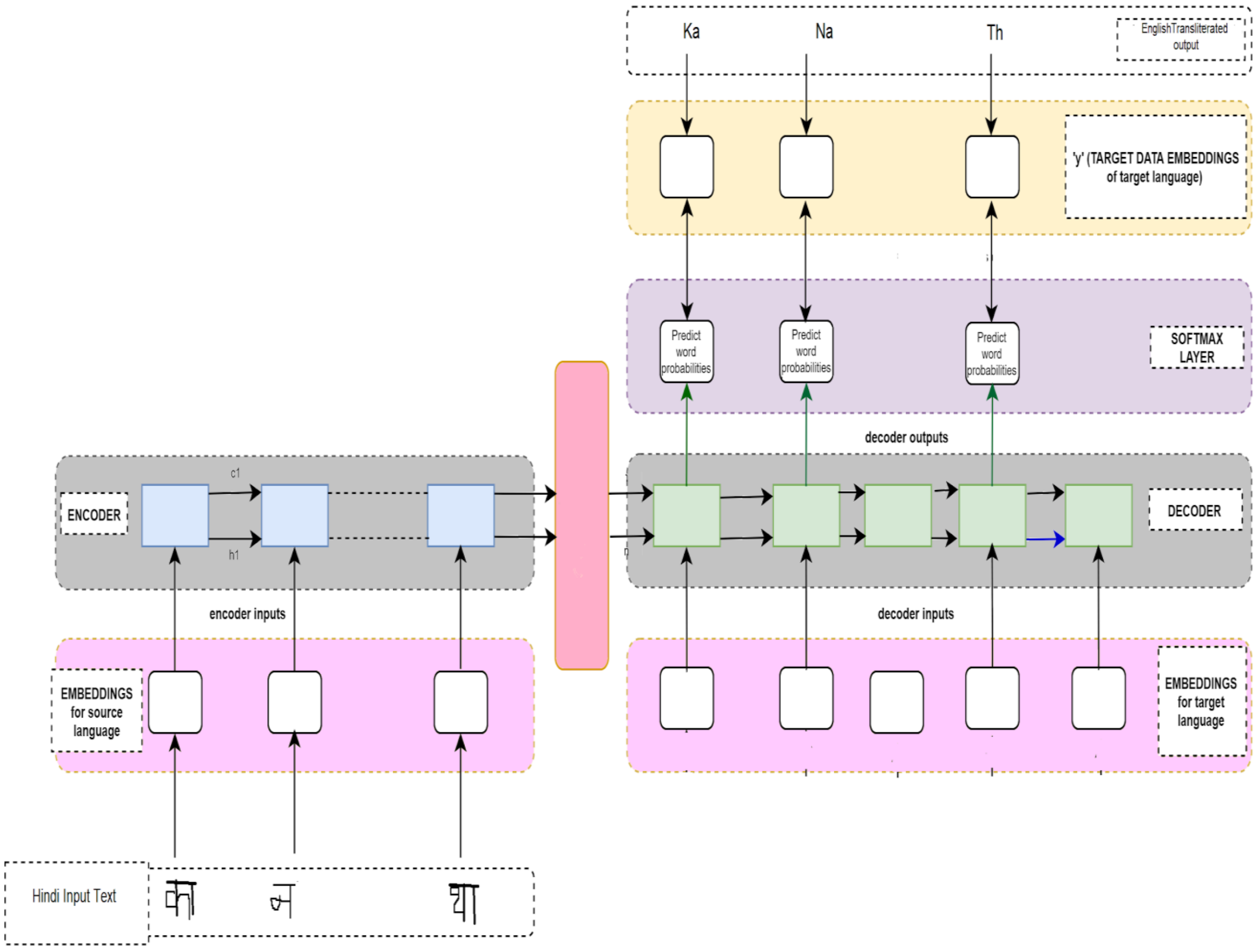
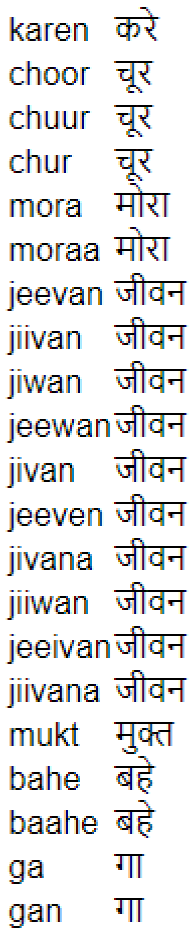

Ka
Na
Th
EnglishTransliterated output
'y' (TARGET DATA EMBEDDINGS of target language)
Predict word probabilities
Predict word probabilities
Predict word probabilities
SOFTMAX LAYER
decoder outputs
ENCODER
c1
h1
DECODER
encoder inputs
decoder inputs
EMBEDDINGS for source language
EMBEDDINGS for target language
Hindi Input Text
क
न
था

Fig 1: Flowchart of our proposed Methodology

3.1 Sample Dataset

Our Sample Dataset



karen	करे
choor	चूर
chuur	चूर
chur	चूर
mora	मोरा
moraa	मोरा
jeevan	जीवन
jiivan	जीवन
jiwan	जीवन
jeewan	जीवन
jivan	जीवन
jeeven	जीवन
jivana	जीवन
jiiwan	जीवन
jeeivan	जीवन
jiivana	जीवन
mukt	मुक्त
bahe	बहे
baahe	बहे
ga	गा
gan	गा

Fig 2: Screenshot of the data set

3.2 Pre-processing of Dataset

We are given a dataset that contains both Hindi and their equivalent English word in a single line. We separate those words and create two different List for English and Hindi words and then we calculate the most frequent characters existing and then create a vocabulary with the help of most frequent words and assign different indexes accordingly. Then we encode input samples with the help of our vocabulary and feed them into embedding layer.

3.3 Training and Testing

We use Encoder Decoder combined with RNN to transliterate our input sample. Create character embeddings for Hindi words. These will be the inputs to the encoder and the decoder. Feed character by character embeds into the encoder till the end of the Hindi word sequence. Obtain the final encoder states (hidden and cell states) and feed them into the decoder as its initial state. At every step of the decoder, the output of the decoder is sent to softmax layer that is compared with the target data. Prepare the embeds for encoder input, decoder input and the target data embeds. We will create one-hot encoding for each character in English and Hindi separately.

The decoder output is passed through the softmax layer that will learn to classify the correct French character.

The BLEU loss is obtained by comparing the predicted values from softmax layer with the *target data*.

Now the model is ready for training. Train the entire network for the specified number of epochs.

Once we predict the character using softmax, we now input this predicted character along with the updated *states val* (updated from the previous decoder states) for the next iteration of the *while* loop. Note that we reset our *target seq* before we create a one-hot embed of the predicted character every time in the *while* loop.

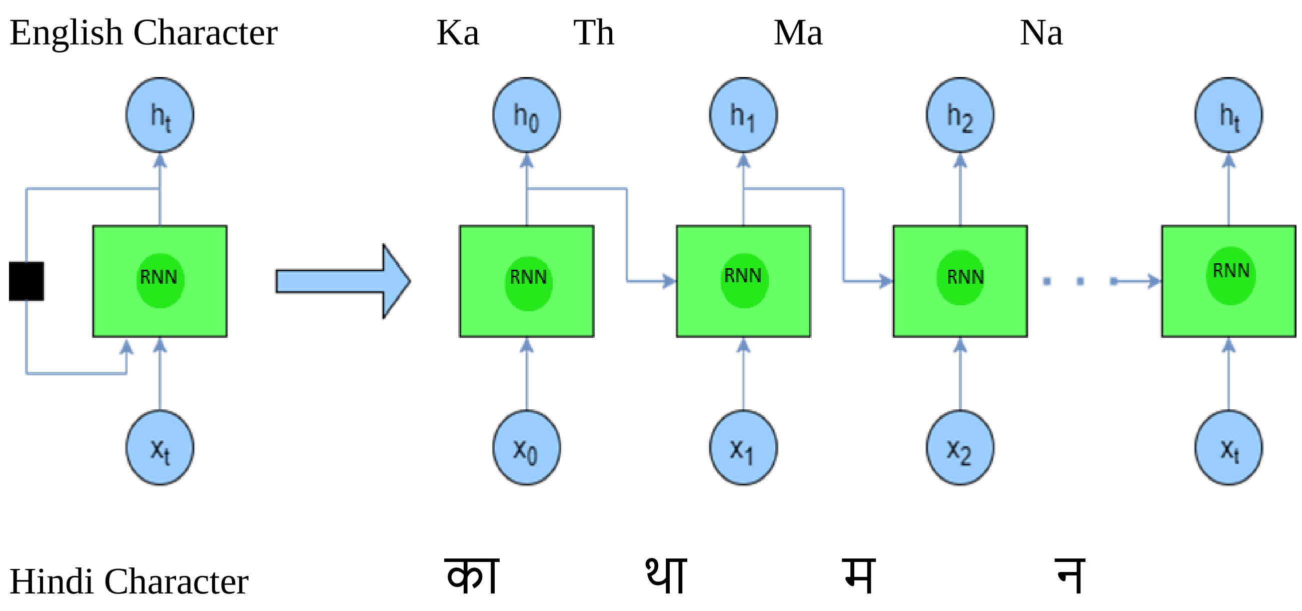


Fig 3: Illustration of Proposed Method

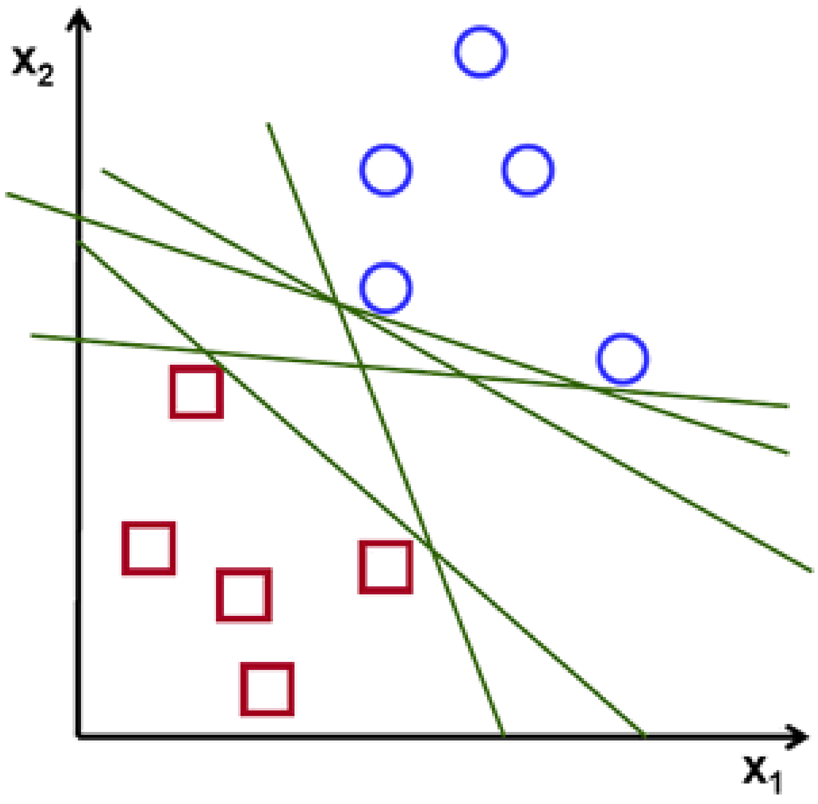
4. Algorithms and Techniques

With increasing globalization, information access across language barriers has become important. Given a source term, machine transliteration refers to generating its phonetic equivalent in the target language. This is important in many cross-language applications. This project explores Hindi to English transliteration. It is followed by a brief overview of the overall project, i.e., ’transliteration involving English and Hindi languages’. The dataset is provided in the form of CSV file. We will be working on algorithms like Sequence to Sequence modelling, RNN.

4.1 Support Vector Machine (SVM)

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N - the number of features) that distinctly classifies the data points.

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.



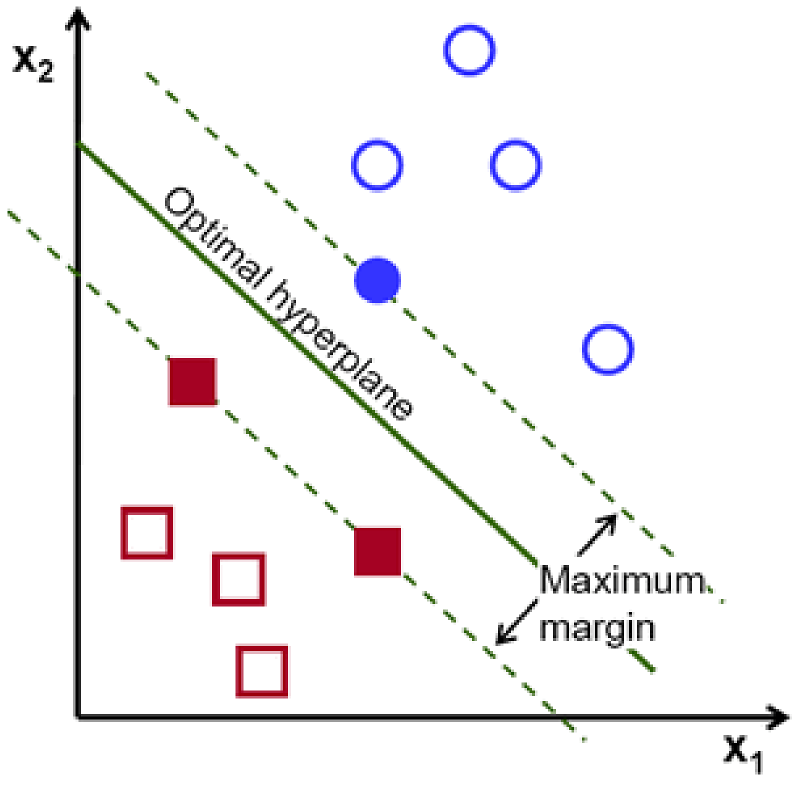


Fig 4: Representation of linear and Non-Linear SVM

4.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) add an interesting twist to basic neural networks. A vanilla neural network takes in a fixed size vector as input which limits its usage in situations that involve a ‘series’ type input with no predetermined size.A single input item from the series is related to others and likely has an influence on its neighbors. Recurrent Neural Network remembers the past and it’s decisions are influenced by what it has learnt from the past.Basic feed forward networks “remember” things too, but they remember things they learnt during training. For example, an image classifier learns what a “1” looks like during training and then uses that knowledge to classify things in production.While RNNs learn similarly while training, in addition, they remember things learnt from prior input(s) while generating output(s). It’s part of the network. RNNs can take one or more input vectors and produce one or more output vectors and the output(s) are influenced not just by weights applied on inputs like a regular NN, but also by a “hidden” state vector representing the context based on prior input(s)/output(s). So, the same input could produce a different output depending on previous inputs in the series.

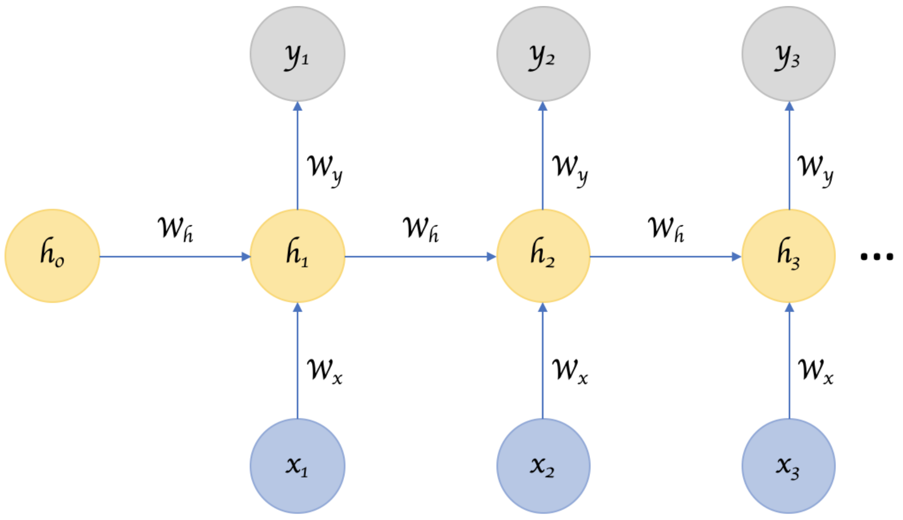


Fig 5: A Recurrent Neural Network, with a hidden state

In summary, in a vanilla neural network, a fixed size input vector is transformed into a fixed size output vector. Such a network becomes “recurrent” when you repeatedly apply the transformations to a series of given input and produce a series of output vectors. There is no pre-set limitation to the size of the vector. And, in addition to generating the output which is a function of the input and hidden state, we update the hidden state itself based on the input and use it in processing the next input.

4.3 ENCODER DECODER SEQUENCE TO SEQUENCE RNNs

Encoder Decoder or Sequence to Sequence RNNs are used a lot in translation/transliteration services. The basic idea is that there are two RNNs, one an encoder that keeps updating its hidden state and produces a final single “Context” output. This is then fed to the decoder, which translates this context to a sequence of outputs. Another key difference in this arrangement is that the length of the input sequence and the length of the output sequence need not necessarily be the same.The model consists of 3 parts: encoder, intermediate (encoder) vector and decoder.

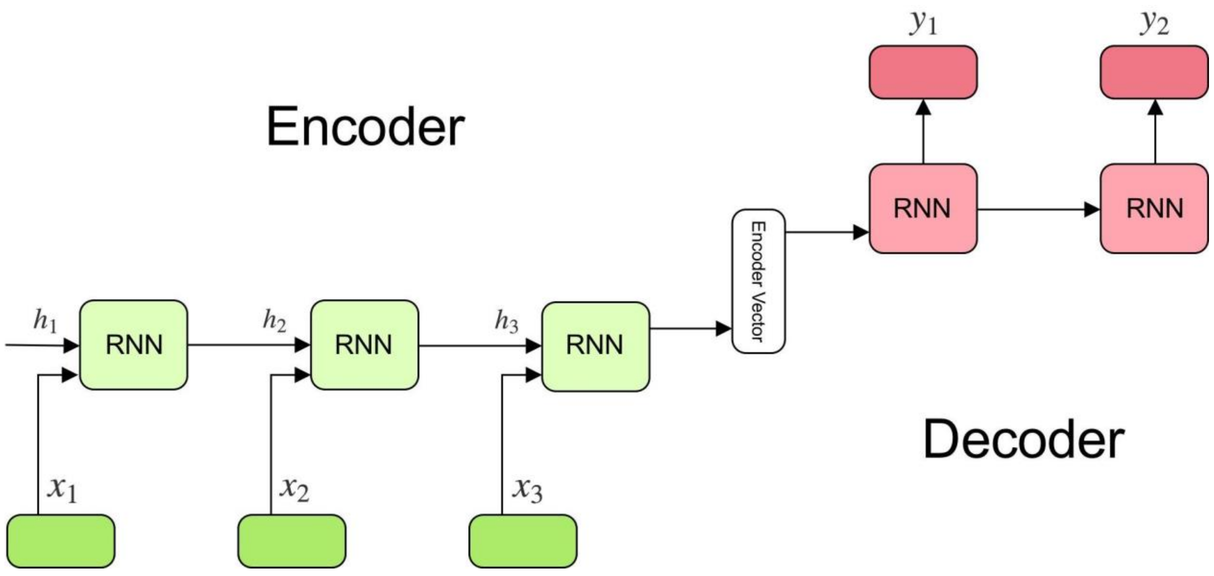


Fig 6: Encoder-decoder sequence to sequence model

Encoder

- A stack of several recurrent units (LSTM or GRU cells for better performance) where each accepts a single element of the input sequence, collects information for that element and propagates it forward.
- The encoder simply takes the input data, and train on it then it passes the last state of its recurrent layer as an initial state to the first recurrent layer of the decoder part.
- In question-answering problem, the input sequence is a collection of all words from the question. Each word is represented as *x_i* where *i* is the order of that word.
- The hidden states *h_i* are computed using the formula:

$$h_t = f(W^{(hh)} h_{t-1} + W^{(hx)} x_t)$$

This simple formula represents the result of an ordinary recurrent neural network. As you can see, we just apply the appropriate weights to the previous hidden state *h_(t-1)* and the input vector *x_t.*

Encoder Vector

- This is the final hidden state produced from the encoder part of the model. It is calculated using the formula above.
- This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.

- It acts as the initial hidden state of the decoder part of the model.

Decoder

- The decoder takes the last state of encoder's last recurrent layer and uses it as an initial state to its first recurrent layer
- A stack of several recurrent units where each predicts an output *y_t* at a time step *t*.
- Each recurrent unit accepts a hidden state from the previous unit and produces and output as well as its own hidden state.
- In the question-answering problem, the output sequence is a collection of all words from the answer. Each word is represented as *y_i* where *i* is the order of that word.
- Any hidden state *h_i* is computed using the formula:

$$h_t = f(W^{(hh)} h_{t-1})$$

As you can see, we are just using the previous hidden state to compute the next one.

- The output *y_t* at time step *t* is computed using the formula:

$$y_t = softmax(W^S h_t)$$

We calculate the outputs using the hidden state at the current time step together with the respective weight W(S). Softmax is used to create a probability vector which will help us determine the final output.

The power of this model lies in the fact that it can map sequences of different lengths to each other. As you can see the inputs and outputs are not correlated and their lengths can differ. This opens a whole new range of problems which can now be solved using such architecture.

- In the world of deep learning, the RNN is considered as the go-to model whenever the problem requires sequence-based learning and this has propelled the research community to come up with interesting improvements over the vanilla RNN. One such prominent improvement is the introduction of gated RNNs: the LSTM and GRU.

LSTM

- Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They work tremendously well on a large variety of problems, and are now widely used.

- LSTMs are explicitly designed to avoid the long term dependency problem. Remembering information for long periods of time is practically their default behaviour, not something they struggle to learn!

- All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

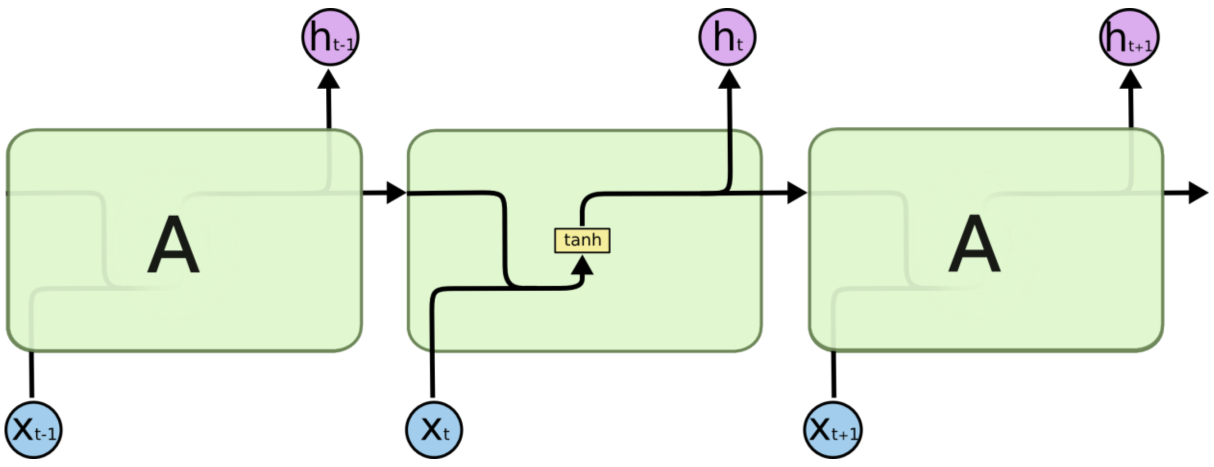


Fig 7: The repeating module in a standard RNN.

- LSTMs also have this chain-like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

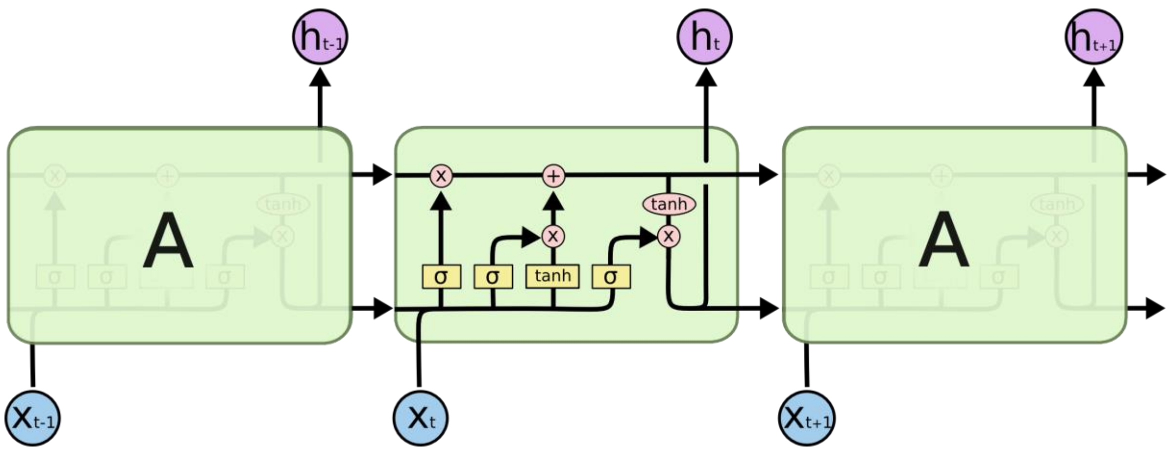
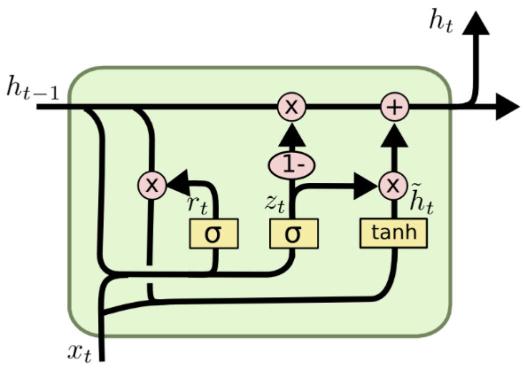


Fig 8: The repeating module in an LSTM.

GRU

The Gated Recurrent Units have the above-mentioned modification along with few others due to which it has an edge over LSTM and is getting adopted by most of them in the practical world.

GRU uses the so called, **update gate and reset gate**. The Sigma notation below represents those gates: which allows a GRU to carry forward information over many time periods in order to influence a future time period. In other words, the value is stored in memory for a certain amount of time and at a critical point pulling that value out and using it with the current state to update at a future date.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Fig 9: GRU Architecture

z_t- update gate

r_t- reset gate

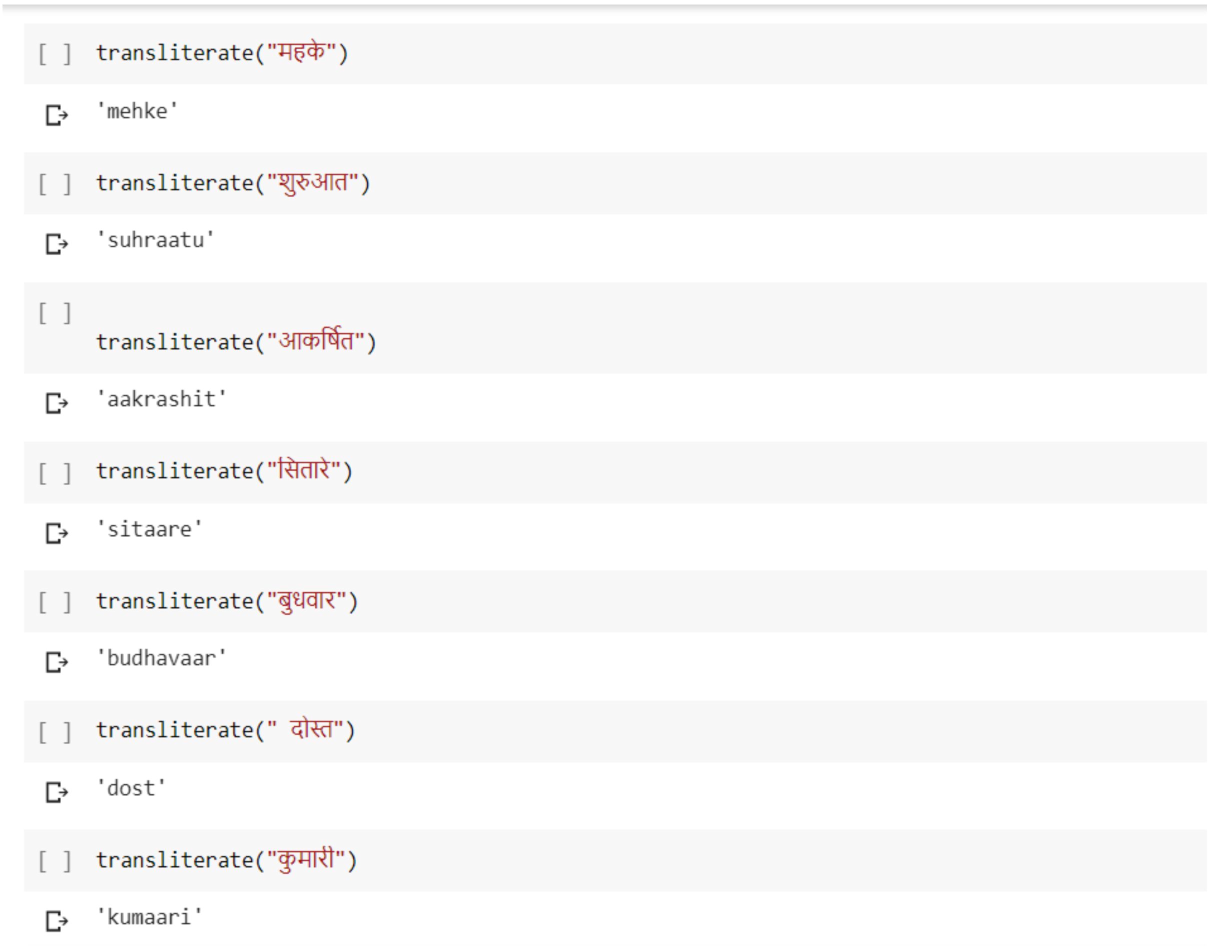
$\underline{h}_t$- Current memory content

h_t-Final memory at current time step

GRUs using the internal memory capability are valuable to store and filter information using their update and reset gates. That said the issues faced by RNNs (vanishing gradient problem) are eliminated offering us a powerful tool to handle sequence data.

5. EXPERIMENTAL RESULTS

We applied two different models namely RNN, Support Vector Machine on our training data since our original data was very huge hence, we filtered out words based on their frequency giving us 30823 English-Hindi words. Using these words, we generated our training data set and the following results were obtained for the given models.



```
[ ] transliterate("महके")
'mehke'
[ ] transliterate("शुरुआत")
'suhraatu'
[ ]
    transliterate("आकर्षित")
'aakrashit'
[ ] transliterate("सितारे")
'sitaare'
[ ] transliterate("बुधवार")
'budhavaar'
[ ] transliterate(" दोस्त")
'dost'
[ ] transliterate("कुमारी")
'kumaari'
```

Fig 10: Screenshot of the Output

BLEU Score

BLEU is controlled by a number of parameters:

• N-gram order, N. Most often, we use N=4.

• Case sensitivity. By default, we compute case insensitive BLEU scores to evaluate a translator. Case sensitive BLEU should be used when evaluating true-case models.

The following is the BLEU scores after final iteration:

BLEU Score: 0.692

BLEU Score: 0.699

BLEU Score: 0.703

BLEU Score: 0.700

6. Tools Used

Due to ease of understanding and implementation, we used the widely popular and well-known Python 3 as our programming environment.

The tools which were used in the development of this project were as follows:

Jupyter Notebook

The *Jupyter Notebook App* is a server-client application that allows editing and running notebook document via a web browser. The *Jupyter Notebook App* can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet. In addition to displaying/editing/running notebook documents, the *Jupyter Notebook App* has a "Dashboard" (Notebook Dashboard), a "control panel" showing local files and allowing to open notebook documents or shutting down their kernels.

Natural Language Tool Kit

The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. It was developed by Steven Bird and Edward Loper in the Department of Computer and Information Science at the University of Pennsylvania. NLTK includes graphical demonstrations and sample data. It is accompanied by a book that explains the underlying concepts behind the language processing tasks supported by the toolkit, plus a cookbook.

NLTK is intended to support research and teaching in NLP or closely related areas, including empirical linguistics, cognitive science, artificial intelligence, information retrieval, and machine learning. NLTK has been used successfully as a teaching tool, as an individual study tool, and as a platform for prototyping and building research systems. There are 32 universities in the US and 25 countries using NLTK in their courses. NLTK supports classification, tokenization, stemming, tagging, parsing, and semantic reasoning functionalities.

SciKit-learn

It is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

One Hot Encoder

It is very useful but it can cause the number of columns to expand greatly if you have very many unique values in a column. For the number of values in this example, it is not a problem. However you can see how this gets really challenging to manage when you have many more options.

7. Conclusion

We described a number of design considerations that one must address when building a robust, multilingual named entity transliteration system. Our results can be extended and potentially improved in a number of ways. A way is by exploring
other recurrent network architectures, such as adaptive computation time networks or classifier combination via boosting or other methods. Another way is by combining the neural network cost with a target language model cost.

Since transliteration involves phonetic features, it might be useful to run a separate pronunciation model on the input string and then provide both the grapheme and the phoneme string as input to the transliteration model, mirroring previous non-neural approaches to transliteration.

Perhaps one of the most important areas of improvements is that of training data. Right now, transliteration research (including the described work) performs training and evaluation on plain correspondences between strings in two orthographic systems. Such an approach disregards word frequencies, and treats predictions involving alternative, valid transcriptions as errors. Improvements
to both the training datasets and the mechanisms for handling multiple predictions will likely result in significant improvements in model performance and correlate more with human evaluations. In addition to improving datasets, our work also points out the need for understanding the relative importance of character and word error rates in evaluating transliterations, since they appear to vary independently.

8. References

1. [Rosca and Breuel[2016] Mihaela Rosca and Thomas Breuel. 2016. Sequence-to-sequence neural network models for transliteration. arXiv preprint arXiv:1610.09565
2. [Cho et al.2014] Kyunghyun Cho and Bart van Merrienboer and Caglar Gulcehre and Dzmitry Bahdanau and Fethi Bougares and Holger Schwenk and Yoshua Bengio.2014.Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.arXiv preprint arXiv:1406.1078
3. [Sutskever et al.2014] 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.
4. [Merhav and Ash2018]Yuval Merhav and Stephen Ash, "Design Challenges in Named Entity Transliteration", *arXiv.org*, 2018
5. [A.Finch,2008] Finch, Andrew, and Eiichiro Sumita, "Phrase-based machine transliteration" in Proceedings of the Workshop on Technologies and Corpora for Asia-Pacific Speech Translation (TCAST), pp. 13-18. 2008.