Language identification for Hindi language transliterated text in Roman Script using Generative Adversarial Networks

Deepak Kumar Sharma[1], Anurag Singh[2], and Abhishek Saroha[3]

[1] Netaji Subhas Institute of Technology,University of Delhi New Delhi, dk.sharma1982@yahoo.co.in
[2] Netaji Subhas Institute of Technology,University of Delhi New Delhi, anurags.it@nsit.net.in
[3] Netaji Subhas Institute of Technology,University of Delhi New Delhi, abhisheks.it@nsit.net.in

Abstract. This work aims to achieve a novel approach to identify text based content written in roman script which conveys meaning in Hindi language.The research work proposes a methodology to identify language based on semantic meaning of the text. The solution is approached by means of feature extraction which are eventually fed to Artificial Neural Network(ANN). The final output of the ANN is multiplied with the feature vector and then fed through a auto-encoder and a Generative Adversarial Network(GAN). Which then trains the model in a semi-supervised manner.The feature extraction defines a feature vector and ANN-model then detects the probability of language classified correctly. The data set was curated using scarping data from web and then cleaning it.Data set was used to train the model.Data from common chat applications was also used and curated to form the data set.

Key words: Transliteration,Natural Language Processing, Deep learning, Feature extraction,Generative Adversarial Networks,auto-encoders

1 Introduction

Language Identification(LID) is one of the key problems in any task corresponding to Natural Language Processing(NLP). It being one of the most primitive problem in NLP. For any processing with the text in a unknown language it is very important to classify which natural language does the text belong to.With the proliferation of internet in our lives we as humans have been surrounded by digital media and content in form of text. Content is written using of natural languages. The task however becomes complex when there is no knowledge of the script that is being used to convey the meaning of the language. Different languages are more or less complete in the set of phonemes by the virtue of sound their characters make or usually multiple syllables help realize all the phonetics present. Thus a text can be transliterated meaning it can written using script of some other language where using words and syllables phonetically

similar to mean something totally different in another language. The first step to translate such text is to detect which language does this transliterated text intend to convey. In context of Indian Languages and most specifically Hindi which is spoken and understood by majority of people in Indian Subcontinent. Meanwhile smart-phones and computers are becoming close to ubiquitous and majority of communication is done digitally. We use keyboards to generate content and express. Keyboards by default are in Roman Script i.e. the simple 26 English alphabets. Although there are keyboards in Devanagari they are tedious and cumbersome to use. This has caused people to type their languages in Roman script. Were the sentences make little or no sense in languages which use Roman script but are phonetically very similar to their own language.
Thus when read out the sentences they sound like they are being spoken in native language. In our case of concern more specifically Hindi. Detection of such text and to label them is a challenging and novel area of research.Detection and labelling of text which sounds phonetically similar to the language that it was intended to be written in is challenging because many features that can be used to easily classify a language are lost due to limited scope of different scripts. The area offers an opportunity to explore and apply myriad of concepts ranging from novel information retrieval techniques used in Natural language processing to Deep learning techniques and architectures in Machine learning. The research in this area will also shed light on novel ways to understand phonetics of texts and will help map languages based on phonetics also.Thus serving as interesting problem to Computational Linguists and Machine Learning community.
The current state-of the art models are either based on statistical calculations or either neural networks based transliteration models which are strictly supervised learning in nature. This work proposes to use new method which is semi-supervised in nature.Despite the current models being used extensively the current models still lack satisfaction and have room for improvement. The current neural machine transliteration systems use maximum likelihood estimation(MLE) principle for training the model.Which means to maximize the probability of a target ground truth sentence given the source sentence.There have been works to work around this.Instead Generative adversarial networks can be used for minimizing the distance between the features extracted and features build using the Generative Networks.The model aims to extract features from the actual data and also generate its own features using Generative Networks which are then compared with the original features using the Discriminator part of GANs which labels the feature vectors as real or fake. Meaning seeing the feature vectors they actually make a choice about which category they might have been generated from. The two parts of the Generative Adversarial Network play a Min-Max game with each other. Eventually making generator better at generating more real like data to fool the discriminator.The following construct is used to make the Artificial Neural Network train themselves of the particular task in hand as same gradients that flow for error correction in the Generator Network can flow back into the ANN helping the network to backpropagate its errors. The subsequent sections describe related work. Propose the model and

explain related concepts. Finally results are discussed,conclusion is made and paper sets scope for future works.

2 Related Works

In this section, the work closely related to our study is reviewed to give a idea of progress. Transliteration is a well documented problem with many researchers attempting to solve the problem fully. The transliteration of Indian languages has been discussed by scholars since late 19Th century. A loss-less romanization system for indic languages was conceptualized when Charles Trevelyan and many other scholars, together in the Transliteration Committee of the Geneva Oriental Congress,on Sept 1894 agreed for transliteration of Sanskrit test to be done using international alphabet for Sanskrit transliteration(IAST)[1].There have been multiple advancements in language transliteration since then. Devanagari transliteration that is Devanagari to roman script, often referred to as romanization is done using multiple approaches. The transliteration process uses language identification, named entity recognition and then the process of conversion into another language. There have been still no standard system for conversion of Devanagari to Roman Script[2].Malik Abbas et.al.[3] attempted something that is very different from romanization.They tried to build a system for Urdu Hindi machine transliteration using statistical models and finite state machines (FSM) which gives an novel idea for rule based transliteration. P koehn et. al. [10] also give statistical machine translation techniques which deal with the formulation of efficient data formats for translation in general. The use case similar to ours was attempted by the Yuxiang Jia et. al. [4] for Chinese languages. They use n-grams Markov models to achieve the task of English Chinese back transliteration. Our model is designed for back transliteration of English Hindi and uses Deep Learning (ANNs) instead. Generative Adversarial Networks[7][8] in Ian GoodFellow et.al. were shown as novel idea that could easily underline the generator distribution for any task. Since then GANs have been used by means of building various architectures around them for solving multiple problems. They have also proved very successful in the task of face generative , text to image generation to name a few. Generative Adversarial Networks are also thought to be applied to language translation task[9].There is very active research for machine translation going on in Google which resulted them being able to detect and translate web-pages making content on web more accessible to all by putting down language barriers for eg Britz and Kenny et al.[11] propose novel work which aims to remove the limitations of neural machine translation(NMT) which is that they are very expensive to train which eventually gives very practical advise based on empirical results they found for English to German translation task. This work will make neural machine translation default standard and ubiquitous for language detection and translation. Therefore this piece of research aims to contribute to achieving transliteration using Generative Adversarial Networks.The focus is on first part of transliteration that is language detection using generative adversarial networks.As a result of which this work aims to give a means of detection of transliterated Hindi text in Roman script using Generative Adversarial Networks.Li et. al[15] have shown that Generative Adversarial Networks can be used in dialogue generation.They modeled this task as a reinforcement learning where two systems,a generative model is used produce response sequences, and a discriminator is used to distinguish between the human-generated dialogues and the machine-generated ones are jointly trained.

3 Methodology

The task this research work aims to solve is of identification of Hindi in transliteration text in Roman Script. The approaches used to tackle the problem solving are based on Deep leaning techniques primarily Neural Networks. It can be noted that no domain-specific resources and/or tools were used for development because it was intended that the work must maintain domain-independence property. That means the architecture can be easily used for other language pairs provided novel features can be extracted within the text. The latter part of this section describes the approach that was followed in developing each part of the model.

3.1 language Identification

Language identification concerns itself with identifying the origin of a particular word or a group of words. The problem statement can be thought of as a case of classification problem, where a group of text or a sentence has to be assigned one of the labels denoting the language it is.Hindi being a original to a non Latin script the method proposed for language identification is based on supervised machine learning. Here certain features present in the text are extracted to build a feature vector. This vector is propagated into a artificial neural network by means of which detection of the language is done.

3.2 Feature Extraction

The sentences in the corpus are formatted, broken into tokens and checked for certain features which then helps the model build a feature vector. There are multiple approaches that are being followed and various features are used to identify languages.The features used for identification of language are described below.

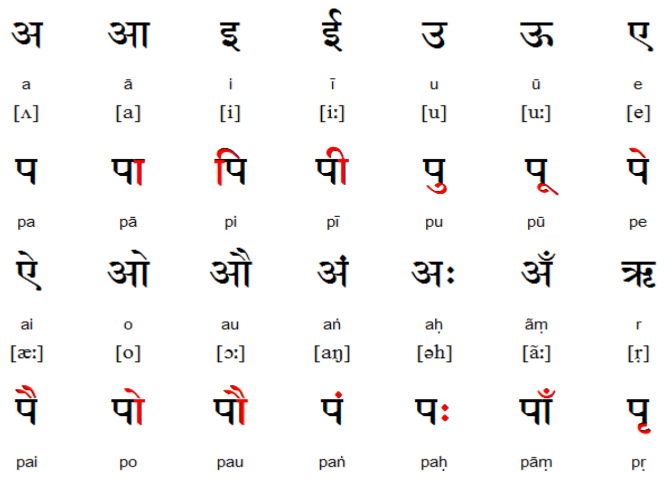
1. *Character n-gram*: Character n-gram are a continuous sequence of alphabets extracted. Here N in N-gram stands for the number of characters which are taken into consideration for defining the probability of next character.That is the size of a moving window which defines previous characters taken into consideration.The common n-grams that are generated for length one (uni gram), two(bi-gram) and three (tri-gram). These kind of feature were also used in previous works [2].

2. *First letter Capital*: The feature is a binary one. It checks for presence of a capital letter at start of every word in sentence.Words starting with capitals can be good indication of Named Entities and thus can be independent of language.
3. *Alphanumeric*: While parsing the sentences such tokens with syllables are looked for which start with a punctuation. For example {:P,:) ;)} do not belong to any language and must not be used to influence the feature vector.
4. *Finite Rule*:This research work helped develop a novel rule based on linguistic construct of Hindi language. The construct is although very specific to Hindi but can be used for all Indian Languages with similar features like Sanskrit,Oriya, Punjabi etc. Therefore this research work also contributes to language detection by proposing a novel feature which is alone sufficient to classify transliterated Hindi text in roman script as Hindi. The Hindi language is written in Devanagari script. With their being phonetic similarities between English vowels and Hindi vowels स्वर as they are usually similar one or two English vowels clubbed together. The Devanagari consonants have a "inherent a" sound. With each consonant the schwa sound must also be pronounced. And in majority of words have a common pattern within themselves. The consonant sounds are followed by a vowel sound. Which means in the transliterated text consonants will be followed by vowels very frequently. The below two images explain the construct of Hindi vowels and consonants and also show their close transliterate text.
   Its been discussed that transliteration and identification of transliterated text relies on the understanding of the native script the language uses. Hindi has twice as many vowels as English. Where vowels can be classified into two particular categories. *Long vowels* and *Short vowels* which are articulated on basis of duration of time. The longer vowels are usually transliterated to English using syllables as a combination of vowels. For example (ईइ) the ई is present in feet is "ee" whereas इ is "i" type in bin.
   In figure 1 and 2 we can see that the phonetic mapping is attempted. Before NMT One of the first steps was to create a mapping of Hindi phones and English phones for getting the transcriptions of Hindi characters. As an transitory step from the Hindi characters one could generate English using older version of Google Transliteration which used the International Alphabet of Sanskrit Transliteration (IAST) scheme. English recognizer could then be used to search through Hindi for English phonemes that correspond to the Hindi syllables.

3.3 Review of auto-encoders and GAN

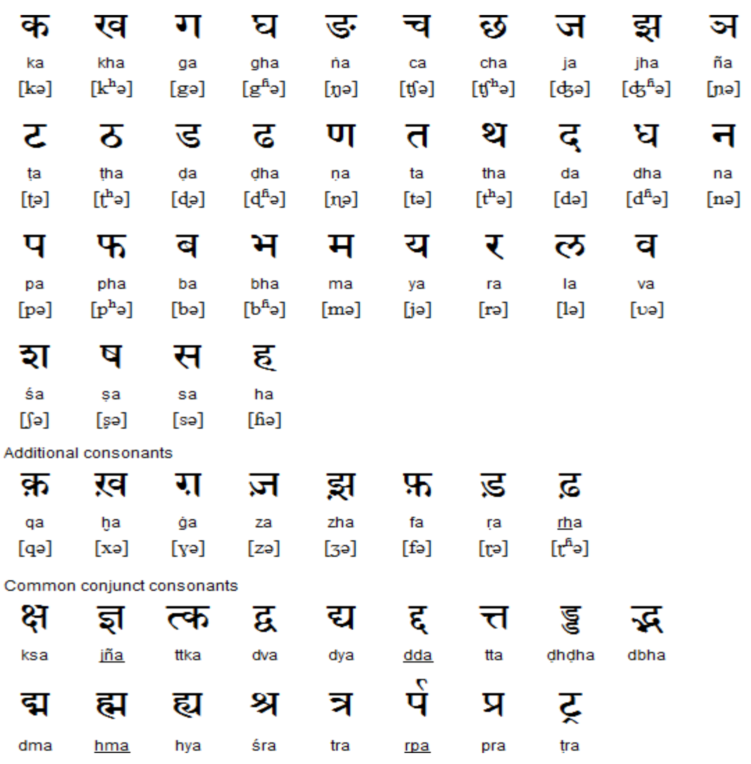
The main components of the model that are used as the building blocks are discussed below.

Artificial Neural Networks A neural network is a computation graphical model that is build using many smaller units of computation. These small units called neurons are "hooked" with each other to create a graph.Each neuron is



अ	आ	इ	ई	उ	ऊ	ए
a	ā	i	ī	u	ū	e
[ʌ]	[a]	[i]	[iː]	[u]	[uː]	[e]
प	पा	पि	पी	पु	पू	पे
pa	pā	pi	pī	pu	pū	pe
ऐ	ओ	औ	अं	अः	अँ	ऋ
ai	o	au	aṅ	aḥ	āṃ	ṛ
[æː]	[o]	[ɔː]	[aŋ]	[əh]	[ãː]	[ɽ]
पै	पो	पौ	पं	पः	पाँ	पृ
pai	po	pau	paṅ	paḥ	pāṃ	pṛ

Fig. 1: Hindi Vowels and their use with consonants
Source: https://www.omniglot.com/writing/hindi.htm



क	ख	ग	घ	ङ	च	छ	ज	झ	ञ
ka	kha	ga	gha	ṅa	ca	cha	ja	jha	ña
[kə]	[kʰə]	[gə]	[gʱə]	[ŋə]	[ʧə]	[ʧʰə]	[ʤə]	[ʤʱə]	[ɲə]
ट	ठ	ड	ढ	ण	त	थ	द	ध	न
ṭa	ṭha	ḍa	ḍha	ṇa	ta	tha	da	dha	na
[ʈə]	[ʈʰə]	[ɖə]	[ɖʱə]	[ɳə]	[t̪ə]	[t̪ʰə]	[d̪ə]	[d̪ʱə]	[nə]
प	फ	ब	भ	म	य	र	ल	व	
pa	pha	ba	bha	ma	ya	ra	la	va	
[pə]	[pʰə]	[bə]	[bʱə]	[mə]	[jə]	[rə]	[lə]	[ʋə]	
श	ष	स	ह						
śa	ṣa	sa	ha						
[ʃə]	[ʂə]	[sə]	[ɦə]						
Additional consonants									
क़	ख़	ग़	ज़	झ़	फ़	ड़	ढ़		
qa	ḫa	ġa	za	zha	fa	ṛa	ṛha		
[qə]	[xə]	[ɣə]	[zə]	[ʒə]	[fə]	[ɽə]	[ɽʱə]		
Common conjunct consonants									
क्ष	ज्ञ	त्क	द्व	द्य	द्द	त्त	ड्ढ	द्भ	
ksa	jña	ttka	dva	dya	dda	tta	ḍhḍha	dbha	
द्म	ह्म	ह्य	श्र	त्र	र्प	प्र	ट्र		
dma	hma	hya	śra	tra	rpa	pra	ṭra		

Fig. 2: Hindi Consonants and inherent a with them
Source: https://www.omniglot.com/writing/hindi.htm

a computational unit which takes input vector and applies a dot product with the weights of the graph to produce the output which is then squashed between 0 to 1 using a sigmoid or some other activation function.

$$o(x_1, .., x_n) = \begin{cases} 1 & if w_0 + w_1x_1 + ... + w_nx_n \geq 0 \\ -1 & otherwise \end{cases}$$

Above is a output of a perceptron unit. For a neuron the output is usually in form of

$$\frac{1}{1 + exp(\sum_j w_jx_j + b_j)} \tag{1}$$

Then error loss is calculated and weight update procedure takes place for the network using backpropagation algorithm.

Auto-Encoders Auto-encoders are a particular case of neural networks. The model consists of two parts that go into the auto-encoder to complete it. The encoder part that encodes the given data into a latent compact representation. The next half of the auto-encoder acts as the "decoder" generating a close representation to data using the latent representation. Within the auto-encoders is a variational auto-encoder(VAE) which acts as a generative model and not a model that can memorize the datasets by mapping them to latent representations. The constraint on encoder is put i.e. model defines a posterior distribution on the observed data given the latent representation. Let $e \sim p(e)$ where $p(e)$ is usually taken to be unit Gaussian distribution.Also x and $q(e|x)$ are the observed data and probability of e given x respectively. Now any latent representation when sampled from unit Gaussian may be decoded to generate images. The loss function which is being used is a two sum. It deals with the trade-off of reconstruction and how nicely does the latent representation match Gaussian distribution.

$$-log(\frac{p(x|e)p(e)}{q(e|x)}) = log(p(x|e)) + D_{KL}(q(e|x)||p(e)) \tag{2}$$

Generative Adversarial Networks The Generative Adversarial Networks or GANs are a set of two competing neural networks. The two halves being generator and the discriminator respectively. The Generator Network generates the counterfeit data close to the actual data in its attempt to mimic the real data. Whereas the Discriminator network tries to find differences in the real and the fake data.That is the data from the dataset and the data generated by the Generator network. Thus both networks play a "game" with each other training themselves to get better at what they do. Eventually the generator becomes good enough to fool the discriminator network generating images very similar to the dataset images. The learning is formulated by following min-max operation. The $\hat{x}$ is the generated data. Where the D that is the Discriminator is trained to maximize the probability of correct classification and G that is the Generator is trained to minimize the same.Here D(.) defines the final output of discriminator and G(.) defines final output of the generator. So the first term in the equation is the term responsible for improving the accuracy of correct classification and second term in equation is responsible for reducing the accuracy of incorrect classification so for discriminator to be better trained we need to maximize the inner expression where as the generator works adversarially to minimize the experssion that is to fool the discriminator.

$$\min_G \max_D [E_x(log(D(x)) + E_e(log(1 - D(\hat{x}))] \tag{3}$$

3.4 Propagation in Model

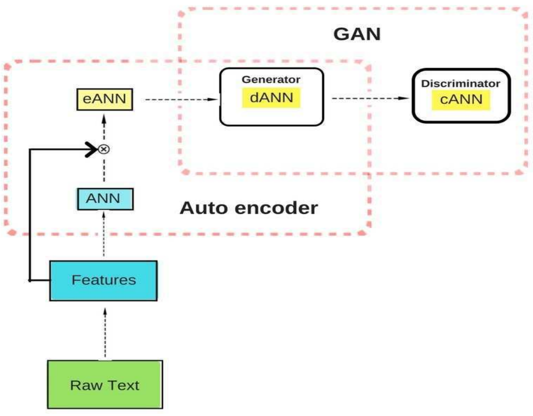


Fig. 3: Block diagram of Model

This section aims to explain the model and also deals with brief discussion of main components in our model which are made using the encoder *eANN* and the decoder *dANN*. The GAN part consists of generator *dANN* and the discriminator *cANN*. As we can see that the decoder part of the auto-encoder is same as the generator part of the GAN. Our goal is illustrated in 3 where we tell that we select features and then make a final layer feature vector which is basically a score vector based on those features. Now to train such network what do we do is we generate the score vector that is made using the Generator network. Then we compare the two score vectors which can then be used to train the network.
Now considering the raw text we extract the set of features x. The feature vector x is then forward propagated through a ANN giving us the score vector s which is basically a set of values of giving indication to which language does the text belong to $s = \{s_i : s_t \in [0, 1]\}$. The score vector is then multiplied to the feature vector x resulting in a matrix. The matrix is the re-sized into a long vector which is then fed to the encoder. The encoder produces a deep feature e specifically for every feature vector. The decoder ANN or the dANN takes the e as the

input and reconstructs the feature vector $\hat{x}$. The discriminator then is aimed to distinguish between the original feature vector x and the feature vector $\hat{x}$ generated by the generator network i.e. dANN. The classifier can be thought of as assigning different class labels to the feature vectors x and $\hat{x}$ provided they can be distinguished.The generator and discriminator are trained adversarial until the discriminator is not able to distinguish between real and generated feature vectors.

3.5 Training the model

This section specifies our training of our neural network and our auto-encoder parameters $\{w_s, w_e, w_d\}$, defining the score ANN or the sANN, the encoder or the eANN and the decoder or the dANN respectively. There is also training of the GAN parameters $\{w_d, w_c\}$, defining the generator network or the dANN and the discriminator network or the cANN. It can be observed that the same dANN is part of both autoencoder as the decoder and acts as the generator in GAN.
Our training is based on four loss functions. Loss of Gan $\mathcal{L}_{GAN}$ Loss of autoencoder $\mathcal{L}_{reconstruct}$ prior loss $\mathcal{L}_{prior}$ and regularization loss $\mathcal{L}_{sparsity}$. The idea behind training our model is to have a additional score vector s_p which is taken from some prior distribution $s_p \sim p(s_p)$. Now multiplying the input feature vector to the s_p gives us the input to eANN producing the encoded representation e_p. Given e_p we reconstruct the vector similar to input feature vector as $\hat{x}_p$. Now $\hat{x}_p$ can be used as a means to regularize the cANN such that it can very accurately label $\hat{x}_p$ to be of the summary class. That also gives the model a threshold to train upto. That the cANN can classify $\hat{x}_p$ as summary but gives $\hat{x}$ original label. The training algorithm thus iteratively updates the given three sets of parameters.

- for parameters $\{w_s, w_e\}$ the gradient descent is calculated using $\mathcal{L}_{prior} + \mathcal{L}_{reconstruct}$
- for parameters w_d the gradient descent is calculated using $\mathcal{L}_{GAN} + \mathcal{L}_{reconstruct}$
- for parameters w_c the gradient descent is calculated using $\mathcal{L}_{GAN}$

Reconstruction Loss $\mathcal{L}_{reconstruct}$ The current standards for learning mechanism in autoencoders use euclidean distances for calculating the difference in actual and reconstructed output.Nevertheless there have been certain recent findings demonstrating the shortcomings of using euclidean distance as a means to calculate loss[9]. Therefore the reconstruction loss $\mathcal{L}_{reconstruct}$ is defined using the output of the last hidden layer of the discriminator ANN. The loss is modelled as expectation of log likelihood of $p(\rho(x)/e)$ where $\rho(x)$ represents the feature vector in the hidden layer of discriminator ANN and e is the encoded output of encoder ANN.

$$\mathcal{L}_{reconstruct} = E[p(\rho(x)/e)] \tag{4}$$

Loss of Gan $\mathcal{L}_{GAN}$ Taking inspiration from [9] the goal to train the discriminator is that it can classify the reconstructed feature vector $\hat{x}$ as 'fake' and original feature vector sequence x as 'original'. For regularization of the model being trained it is additionally enforced that the model learns to classify set of randomly generated feature vectors $\hat{x}_p$ as 'fake'. Therefore the loss is expressed as:

$$\mathcal{L}_{GAN} = log(cANN(x)) + log(1 - cANN(\hat{x})) + log(1 - cANN(\hat{x}_p)) \quad (5)$$

Here cANN(.) denotes the output of the discriminator ANN. Given the definitions of $\mathcal{L}_{reconstruct}$ and $\mathcal{L}_{GAN}$ we update the parameters w_s, w_e, w_d, w_c using stochastic gradient descent. The algorithm given below summarizes each step for training the model. The capital letter notation is used to show that variable stands for mini-batch of corresponding small alphabets used in the paper.

Algorithm 1 Training the model

```
 1: function UPDATE PARAMS        ▷ where input is the feature vector secquence and
    output is learned parameters $w_s, w_e, w_d, wc$
 2:     for m doax number of iterations do
 3:         $X = MiniBatchOfFeatureSequences$
 4:         $S = sANN(X)$                                              ▷ select frames
 5:         $E = eANN(X, S)$                                           ▷ encoding
 6:         $\hat{X} = dANN(E)$                                        ▷ Reconstruction
 7:         $S_p = DrawSamplesFromUniformDistribution$
 8:         $E_p = eANN(X, S_p)$                                       ▷ encoding
 9:         $X_p dANN(E_{S_p})$                                        ▷ Reconstruction
10:         $\{w_s, w_e\} = \{w_s, w_e\} - \nabla(\mathcal{L}_{reconstruct} + \mathcal{L}_{prior})$   ▷ weight updates
11:         $\{w_d\} = \{w_d\} - \nabla(\mathcal{L}_{reconstruct} + \mathcal{L}_{GAN})$
12:         $\{w_c\} = \{w_c\} + \nabla(\mathcal{L}_{GAN})$                  ▷ Maximization Update
```

4 Results

4.1 Training results

The data set is scraped from reliable sources and then was verified through Google Detection API [12] by sending an HTTP request using a URL[13].The verified data set was divided into training and testing data sets in the ratio 80:20.The training data set was further divided into training and validation data sets in the ratio 80:20.Our model is trained on a batch size of 10000 words and an accuracy vs epochs graph is plotted where x-axis represents the number of epochs and y-axis represents accuracy of the model.The blue line denotes the accuracy on validation data set and red line denotes the accuracy on training data set.Accuracy of the model can be seen increasing on subsequent epochs from the graph on both training as well as on validation data. In 5 we talk

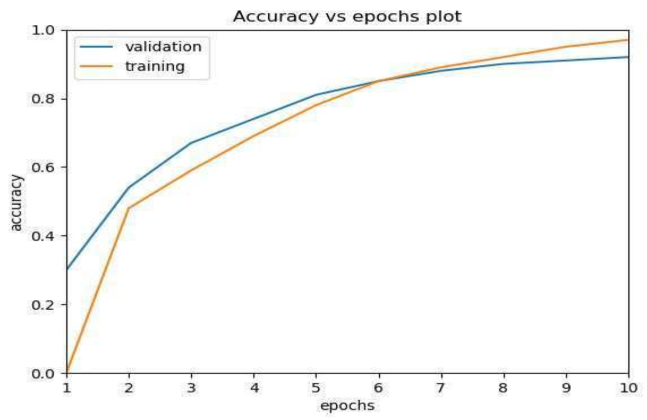


Fig. 4: Graph of Accuracy over Training data and validation data

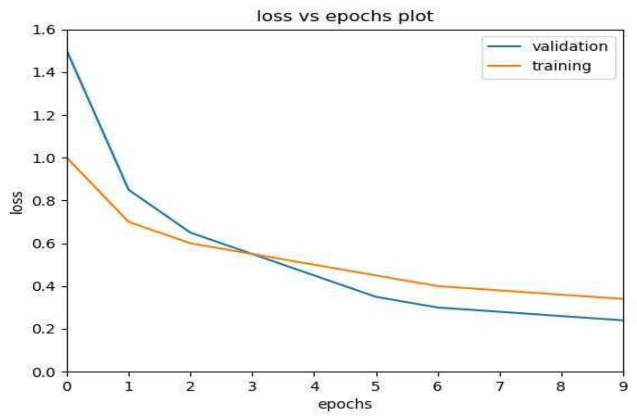


Fig. 5: Graph of Loss over Training data and validation data

about loss.the loss taken is the sum of all individual losses and then scaled using a constant.

4.2 Comparative results

Google detector is used for comparison of accuracy with our model.The test data was divided into different categories-1 word,2 words,5 words,10 words and more than 10 words each having a batch size of 10000 examples.Both ours as well as Google model was tested on these 5 categories and results were plotted on the graph,where x-axis represents number of words and y-axis represents accuracy of model.The blue line denotes the accuracy of Google detector and green line denotes the accuracy of proposed model.For data set of 1 words category,both Google detector and our model gives relatively low accuracy as it includes both unambiguous and ambiguous words.Ambiguous words are those which have same spelling in different languages like "arre" is detected as English while it is more commonly used in Hindi.Similarly words like "kahan","jab" are also detected as English.Hence the relatively lower accuracy of 1 word data set is justified due to ambiguity.For data set of 2 words category the accuracy of both Google and proposed model increases rapidly as ambiguity is much less than 1 word.On increasing the word count per example in data set to 5 words per testing example the Google detector achieves 100 percent accuracy.It can be seen from the graph that our proposed model is giving sufficient accuracy when the number of words are increased above 5 and it almost achieves 100 percent accuracy when word count of test data per example increases above 10 words.

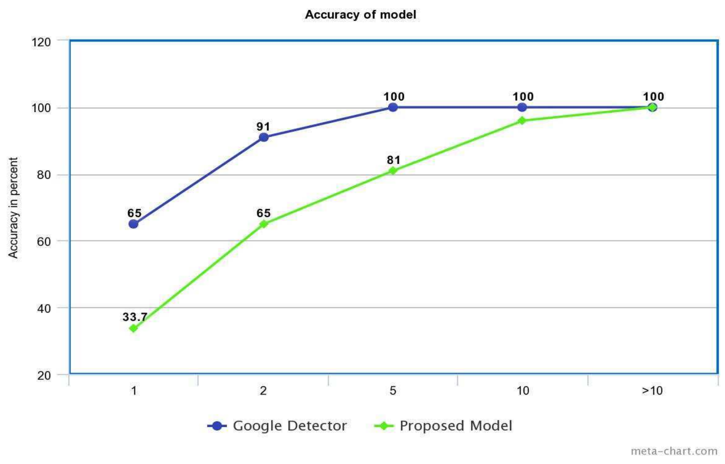


Fig. 6: Comparative Results of accuracy over number of words in input

5 Conclusion

The model proposed performs with reliable accuracy when number of words in data set per example are more than 10 and with sufficient accuracy when number of words are less than 10.Our proposed model gives comparable results with Google detector.It can be concluded from the results that semi-supervised learning models such as ours can use Generative Adversarial Networks for Hindi language identification and for transliterated text identification in general with great accuracy.Thus the comparison with state of the art Google Language detection establishes that the model using feature extraction and then unsupervised learning of parameters is also at par with the current standards in language detection. It also opens up new frontiers were experiment with different types of deep learning models can be done which train in completely unsupervised fashion.Eventually even removing the need to engineer features in such models is a open challenge which future research can look into.

References

1. Monier-Williams, Monier (1899). A Sanskrit-English Dictionary (PDF). Oxford: Clarendon Press. pp. xxx.
2. Daya Nand Sharma, Transliteration into Roman and Devangar of the languages of the Indian group, Survey of India, 1972
3. Malik, Abbas, et al. "A hybrid model for Urdu Hindi transliteration." Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration. Association for Computational Linguistics, 2009.
4. Jia, Yuxiang, Danqing Zhu, and Shiwen Yu. "A noisy channel model for grapheme-based machine transliteration." Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration. Association for Computational Linguistics, 2009.
5. Michael C. Shapiro (2008). A Primer of Modern Standard Hindi. Motilal Banarsidass Publishers Private Limited, 2008 Reprint
6. ΨKyungHyun Cho and Bart van Merrienboer and Dzmitry Bahdanau and Yoshua Bengio the Properties of Neural Machine Translation: Encoder-Decoder Approaches arXiv:1409.1259
7. Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems. 2014.
8. Yoshua, Courville. "Generative adversarial networks."
9. Wu, Lijun, et al. "Adversarial Neural Machine Translation." arXiv preprint arXiv:1704.06933 (2017)
10. Koehn, Philipp, et al. "Moses: Open source toolkit for statistical machine translation." Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions. Association for Computational Linguistics, 2007.
11. Britz, Denny, et al. "Massive exploration of neural machine translation architectures." arXiv preprint arXiv:1703.03906 (2017).
12. https://cloud.google.com/translate/docs/
13. https://translation.googleapis.com/language/translate/v2/detect
14. Jiwei Li, Will Monroe, Tianlin Shi, Alan Ritter,and Dan Jurafsky. Adversarial learning for neural dialogue generation.arXiv preprint arXiv:1701.06547, 2017.