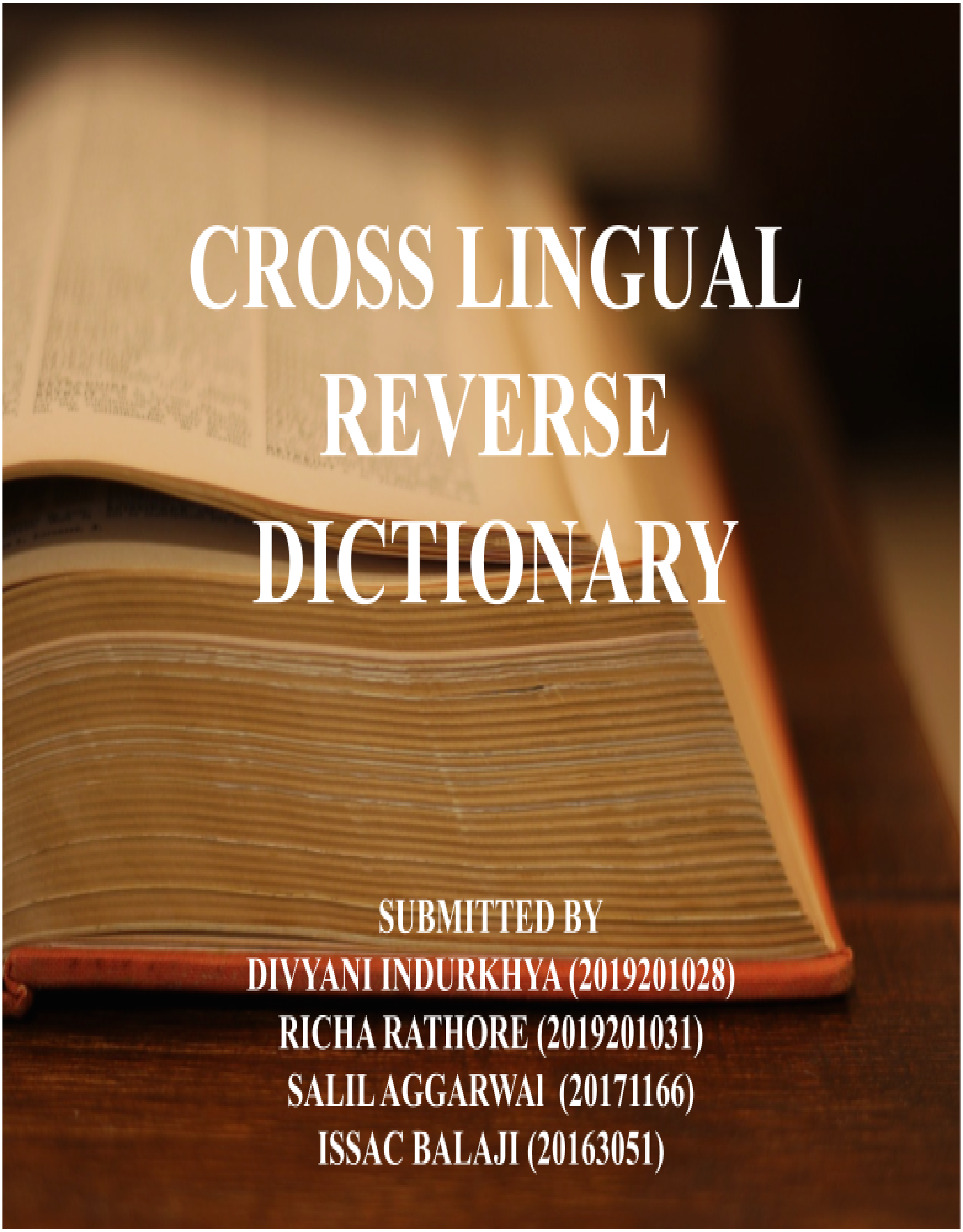


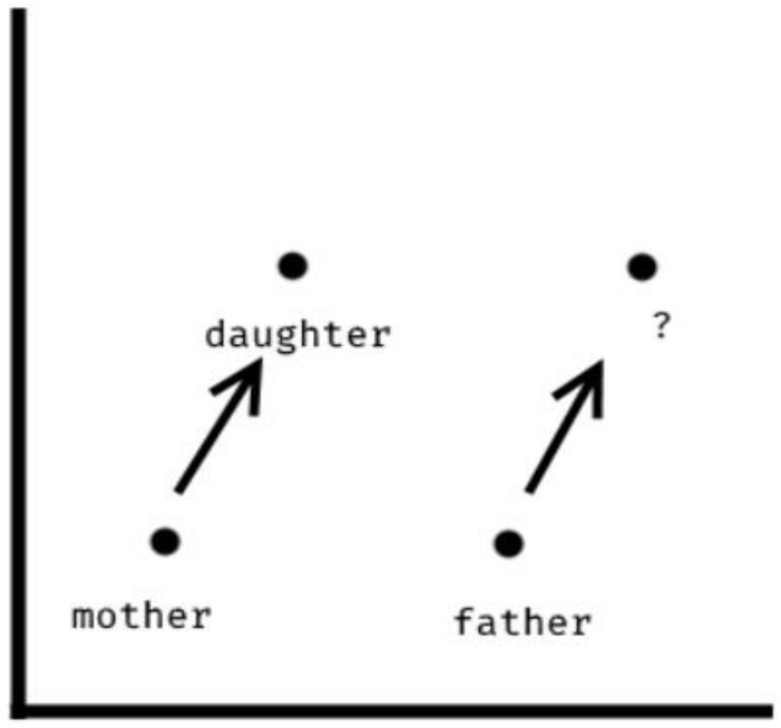
CROSS LINGUAL REVERSE DICTIONARY

SUBMITTED BY
DIVYANI INDURKHYA (2019201028)
RICHA RATHORE (2019201031)
SALIL AGGARWAI (20171166)
ISSAC BALAJI (20163051)

Cross Lingual Reverse Dictionary

Word2Vec

A reverse dictionary is simply a dictionary in which you input a definition and get back the word that matches that definition. For the baseline, we implemented reverse dictionary by using the Word2Vec embeddings model. For the cross lingual part, we used Google translate API to convert the word into the desired language.



Why it works?

The system can find the word indicated by ? simply because it can add, from the vector for father, the difference between the two given word vectors (mother and daughter). The system captures the relationship between words. We used Word2Vec embeddings to find a word that is most like the combination of input words, the definition. This will work because the system uses vector mathematics to find the word that is closer to the set of words given as our input.

For example, if you input group of relatives, it should find family. We are also able to use negated words in the definition to help identify a word. For example, group of -relatives resolves to organization.

Data for our Word2Vec Model?

For practical reasons, we used pre-trained model shared by Google based on Google News data: GoogleNews-vectors-negative300.bin.gz. It is 1.6GB compressed around 4GB in compressed form.

On one hand, the Google News model is great, it has been generated by a very large dataset, so it is quite accurate. It allows us to get much better results than what we would get building our own model. However, since it is based on news, it also contains a lot of misspellings and, more importantly, a model for entities that we do not need.

In other words, it does not contain just individual words, but also stuff like the names of buildings and institutions that are mentioned in the news. And since we want to build a reverse dictionary, this could hinder us. So, we filtered all the items output by our model to ensure that only commonly used words that you can find in a dictionary are shown to the user. Our list of such words comes from SCOWL (Spell Checker Oriented Word Lists).

Results

Well, this approach worked quite well for descriptive definitions, i.e., if we use a definition that describes the word we are looking for. By that, we mean that you will probably find the correct word in the list returned by the model. It might not be the top word, but it should be there. It does not work that well for logical definitions, e.g., female spouse. So, it is certainly not a perfect solution, but it works quite well for something so simple and acts as a good baseline.

BERT

We used dense vector representations for sentences and paragraphs (also known as sentence embeddings). The pretrained models used were based on transformer networks like BERT and are tuned such that sentences with similar meanings are close in vector space. We used semantic search which seeks to improve search accuracy by understanding the content of the search query. In contrast to traditional search engines, that only finds documents based on lexical matches, semantic search can also find synonyms.

We implemented domain specific reverse dictionary using BERT sentence embeddings. We used around 1200 job professions and their definitions. The jobs were manually converted into Hindi and Punjabi for the cross lingual part. We then computed the cosine-similarity between the query and all entries in the corpus.

For that, we computed the embeddings for the corpus as well as for our query. We then used the util.pytorch_cos_sim() function to compute the cosine similarity between the query and all corpus entries. Then we used torch.topk to only get the top k entries.

RESULTS

For testing our model, we manually created complex descriptions for 20 jobs and tried on our model. The model gave 95 percent accuracy on our data. So, BERT performed very well on small datasets and we can say that for building domain specific reverse dictionary, BERT is an excellent choice.

MODEL: Neural Network with LSTM

Implementation of Cross Lingual Reverse Dictionary i.e. Given a set of keywords in a language, the output is displayed in another language.

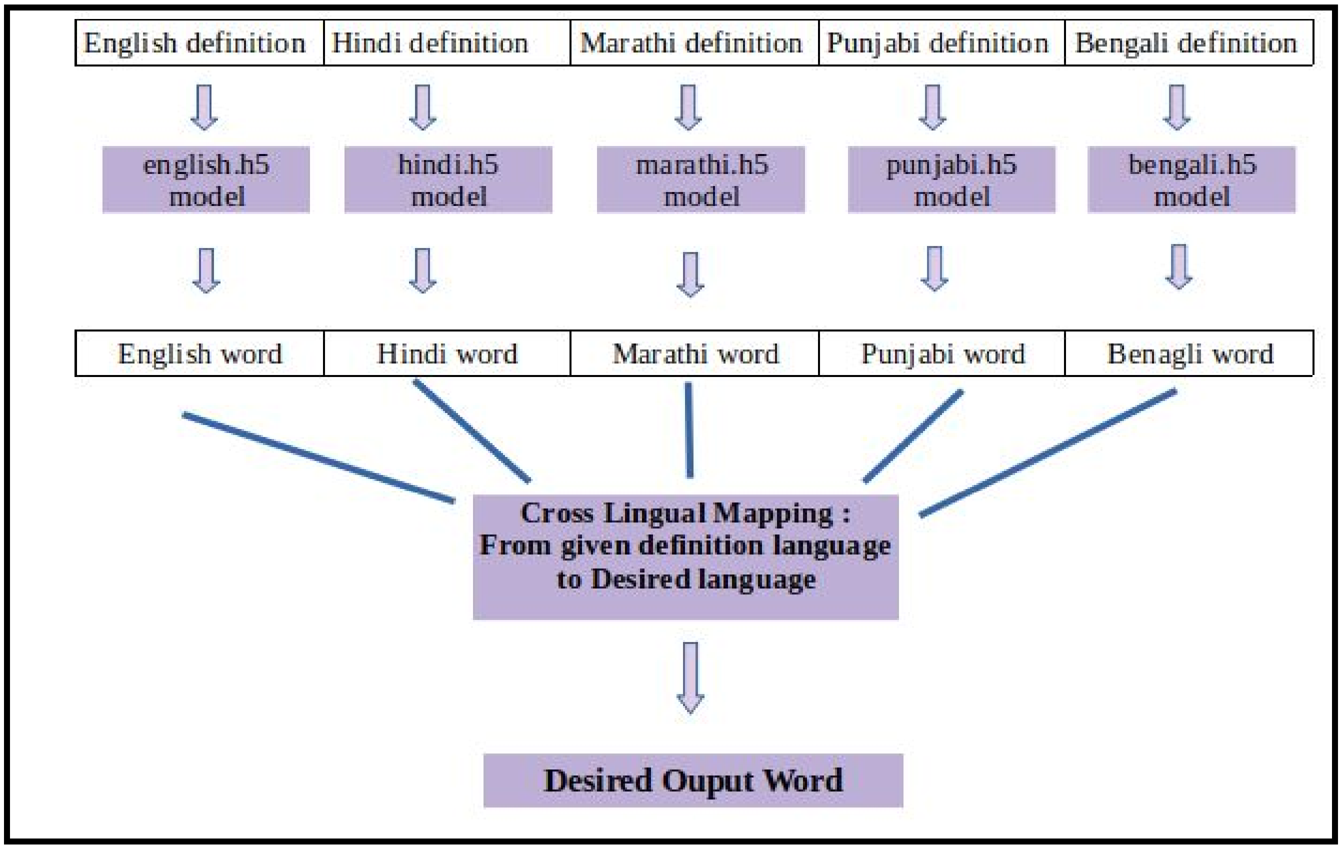
Before Jumping to the Description and Implementation of the model. We have specified the entire statistics of the model.

UNPROCESSED DATASET SIZE : ~11k word definition pair in 5 languages

VOCAB SIZE: The vocabulary size of each language depends upon the total distinct words used in the sentence of each language.

English	Hindi	Marathi	Punjabi	Bengali
13778	16725	21854	18009	17455

TRAINED MODEL: Each language model is trained separately and 5 different models are stored with their weights in .h5 format. This is to reduce the run time and query processing time.



Model: Our project is divided into three phases and it works on 5 languages namely, Hindi, English, Punjabi, Marathi and Bengali.

1. Data Collection and Preprocessing
2. Neural Network with LSTM model Training
3. Resultant language mapping

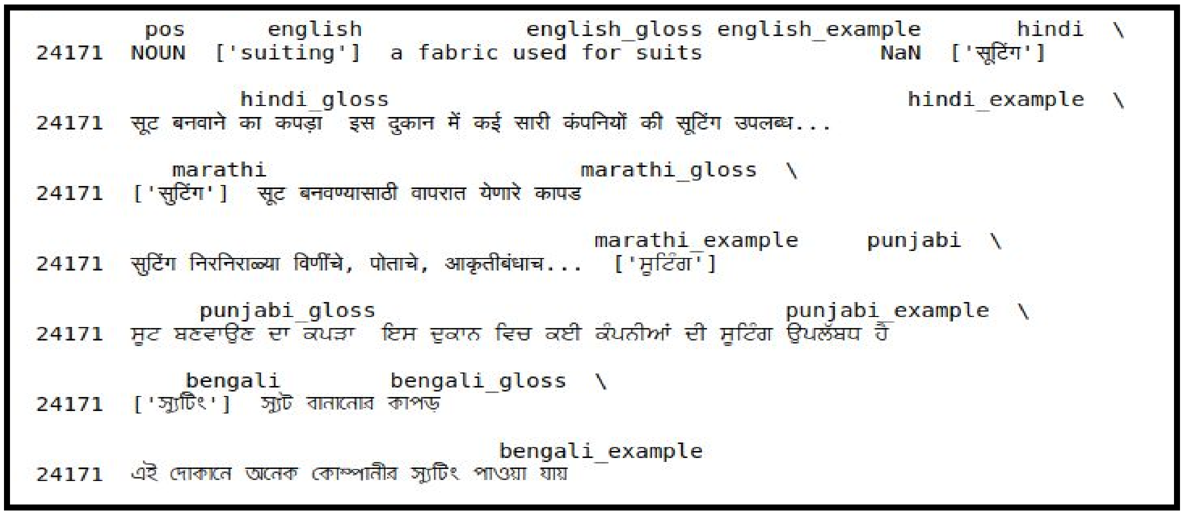
PHASE 1: Data Collection and Preprocessing

This model is highly based upon the quality as well as quantity of the data. Thus we need to ensure proper clean data and structured data. The data collection phase implements various python scraping techniques to extract data from various websites in all the desired languages.

Data Collection: We used IndoWordnet website to extract the wordlist with
its definition and synonyms are extracted in 5 languages.

1. Hindi
2. English
3. Marathi
4. Punjabi
5. Bengali

The dataset consists of around 11k word definition pairs in all the languages. Below image shows the first row of the dataset.



```
        pos      english              english_gloss english_example       hindi  \
24171  NOUN  ['suiting']  a fabric used for suits             NaN  ['सूटिंग']

              hindi_gloss                                hindi_example  \
24171  सूट बनवाने का कपड़ा  इस दुकान में कई सारी कंपनियों की सूटिंग उपलब्ध...

          marathi                           marathi_gloss  \
24171  ['सुटिंग']  सूट बनवण्यासाठी वापरात येणारे कापड

                              marathi_example     punjabi  \
24171  सुटिंग निरनिराळ्या विर्णींचे, पोताचे, आकृतीबंधाच...  ['ਸੂਟਿੰਗ']

           punjabi_gloss                              punjabi_example  \
24171  ਸੂਟ ਬਣਵਾਉਣ ਦਾ ਕਪੜਾ  ਇਸ ਦੁਕਾਨ ਵਿਚ ਕਈ ਕੰਪਨੀਆਂ ਦੀ ਸੂਟਿੰਗ ਉਪਲੱਬਧ ਹੈ

          bengali        bengali_gloss  \
24171  ['স্যুটিং']  স্যুট বানানোর কাপড়

                            bengali_example
24171  এই দোকানে অনেক কোম্পানীর স্যুটিং পাওয়া যায়
```

<u>Data Preprocessing:</u>
This is further divided into multiple steps:

STEP 1: Word Definition and Cross Lingual Mapping Creation

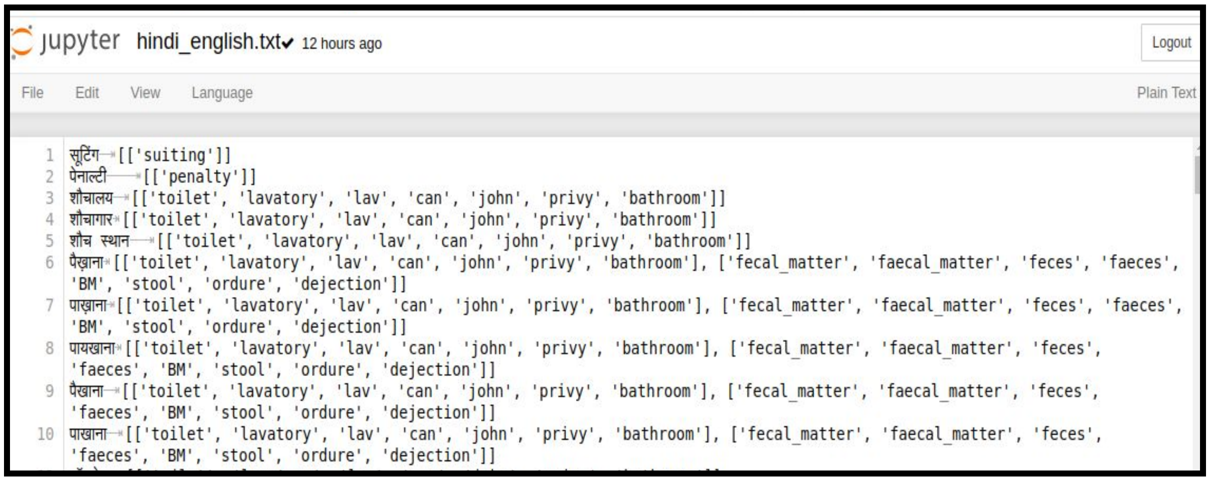
- Initially, we extract each language word_definition pair and store it in the corresponding language_word_definiton.txt
  Each word is mapped uniquely with its definition.

Word list: 'चवालीसवाँ', 'चौआलिसवाँ', 'चौंआलिसवाँ'
↓ ↓ ↓
Definition गणना में चवालीस के स्थान पर आनेवाला चवालीसवें खूंटे पर फूल बना है ।

- A list of 5 files is created [‘english_word_def.txt’ ,‘hindi_word_def.txt’ , ‘marathi_word_def.txt’, ‘punjabi_word_def.txt’, ‘bengali_word_def.txt’]. These files contain the unique word and its corresponding definition.

- After this, to implement cross linguality, each language word is associated with other language words.

- As can be seen from the figure below, every hindi word is mapped to all the english words with the same meaning. This way we can display multiple matching words for any definition.

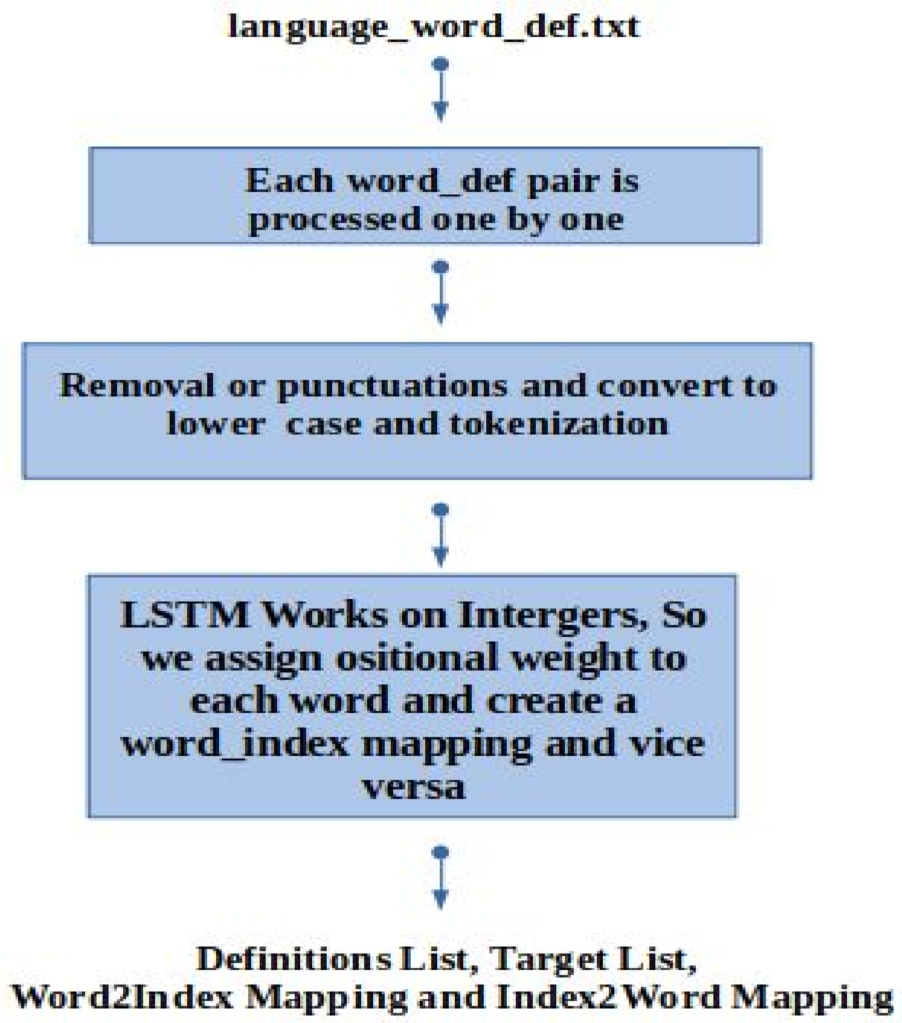


```
सूटिंग→[['suiting']]
पेनाल्टी→[['penalty']]
शौचालय→[['toilet', 'lavatory', 'lav', 'can', 'john', 'privy', 'bathroom']]
शौचागार→[['toilet', 'lavatory', 'lav', 'can', 'john', 'privy', 'bathroom']]
शौच स्थान→[['toilet', 'lavatory', 'lav', 'can', 'john', 'privy', 'bathroom']]
पैख़ाना→[['toilet', 'lavatory', 'lav', 'can', 'john', 'privy', 'bathroom'], ['fecal_matter', 'faecal_matter', 'feces', 'faeces',
'BM', 'stool', 'ordure', 'dejection']]
पाख़ाना→[['toilet', 'lavatory', 'lav', 'can', 'john', 'privy', 'bathroom'], ['fecal_matter', 'faecal_matter', 'feces', 'faeces',
'BM', 'stool', 'ordure', 'dejection']]
पायख़ाना→[['toilet', 'lavatory', 'lav', 'can', 'john', 'privy', 'bathroom'], ['fecal_matter', 'faecal_matter', 'feces',
'faeces', 'BM', 'stool', 'ordure', 'dejection']]
पैखाना→[['toilet', 'lavatory', 'lav', 'can', 'john', 'privy', 'bathroom'], ['fecal_matter', 'faecal_matter', 'feces',
'faeces', 'BM', 'stool', 'ordure', 'dejection']]
पाखाना→[['toilet', 'lavatory', 'lav', 'can', 'john', 'privy', 'bathroom'], ['fecal_matter', 'faecal_matter', 'feces',
'faeces', 'BM', 'stool', 'ordure', 'dejection']]
```

Figure: hindi word mapping with english word list

STEP 2: Processing word definitions and Creation of Model Input

- Once individual files are created, Each language file is sent through a data processing function. The following steps take place


language_word_def.txt
Each word_def pair is processed one by one
Removal or punctuations and convert to lower case and tokenization
LSTM Works on Intergers, So we assign ositional weight to each word and create a word_index mapping and vice versa
Definitions List, Target List, Word2Index Mapping and Index2Word Mapping

- The Output of data processing is a tokenized word list of definition, Target word list, Word2 index positional weight mapping and Index to word positional Weight mapping
- The above process is carried out for all the 5 languages.

PHASE 2: MODEL DESIGN

LSTM:

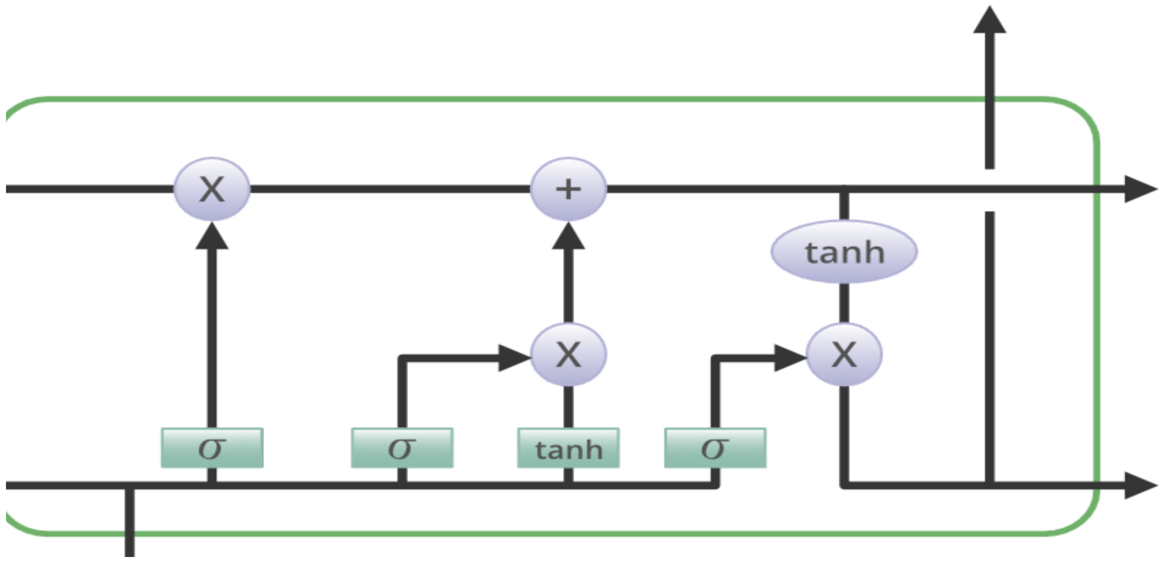
Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. This is a behaviour required in complex problem domains like machine translation, speech recognition, and more.

LSTMs have an edge over conventional feed-forward neural networks and RNN in many ways. This is because of their property of selectively remembering patterns for long durations of time.

LSTM can maintain constant error, which allows them to continue learning over Numerous time-steps and backpropagation through time and layers. LSTM can by default retain the information for a long period of time. It is used for processing, predicting and classifying on the basis of time series data.

STRUCTURE OF LSTM

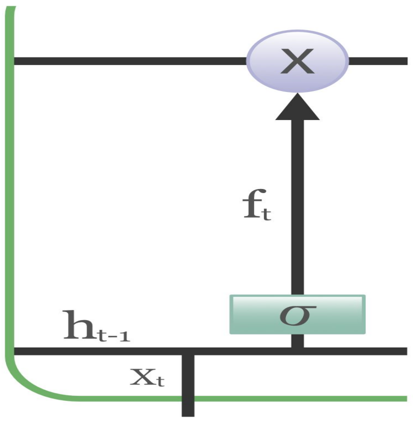
LSTM has a chain structure that contains four neural networks and different memory blocks called cells.



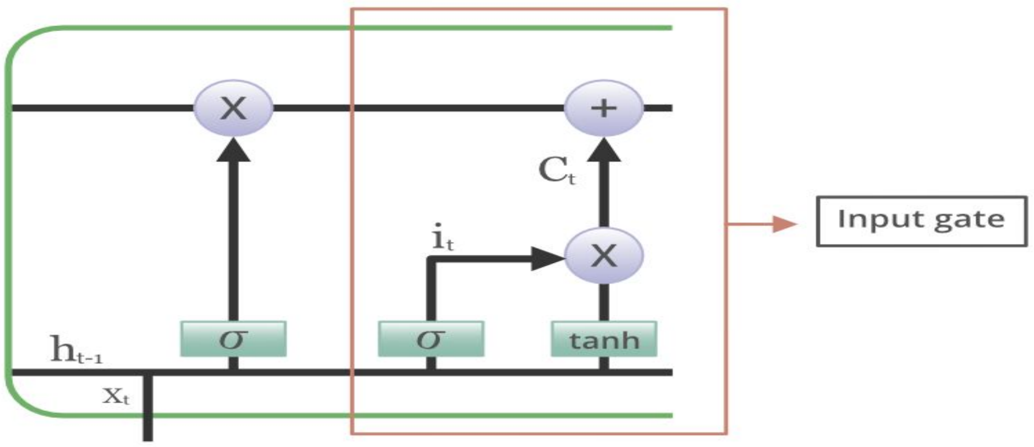
Information is retained by the cells and the memory manipulations are done by the **gates.** There are three gates –

1. **Forget Gate:** The information that is no longer useful in the cell state is removed with the forget gate. Two inputs *x_t* (input at the particular time) and *h_t-1* (previous cell output) are fed to the gate and multiplied with weight matrices

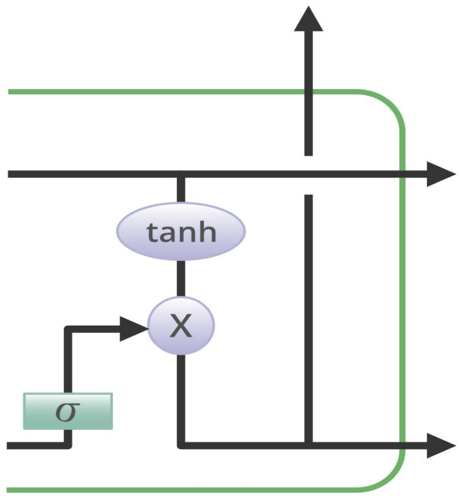
followed by the addition of bias. The resultant is passed through an activation function which gives a binary output. If for a particular cell state the output is 0, the piece of information is forgotten and for the output 1, the information is retained for the future use.



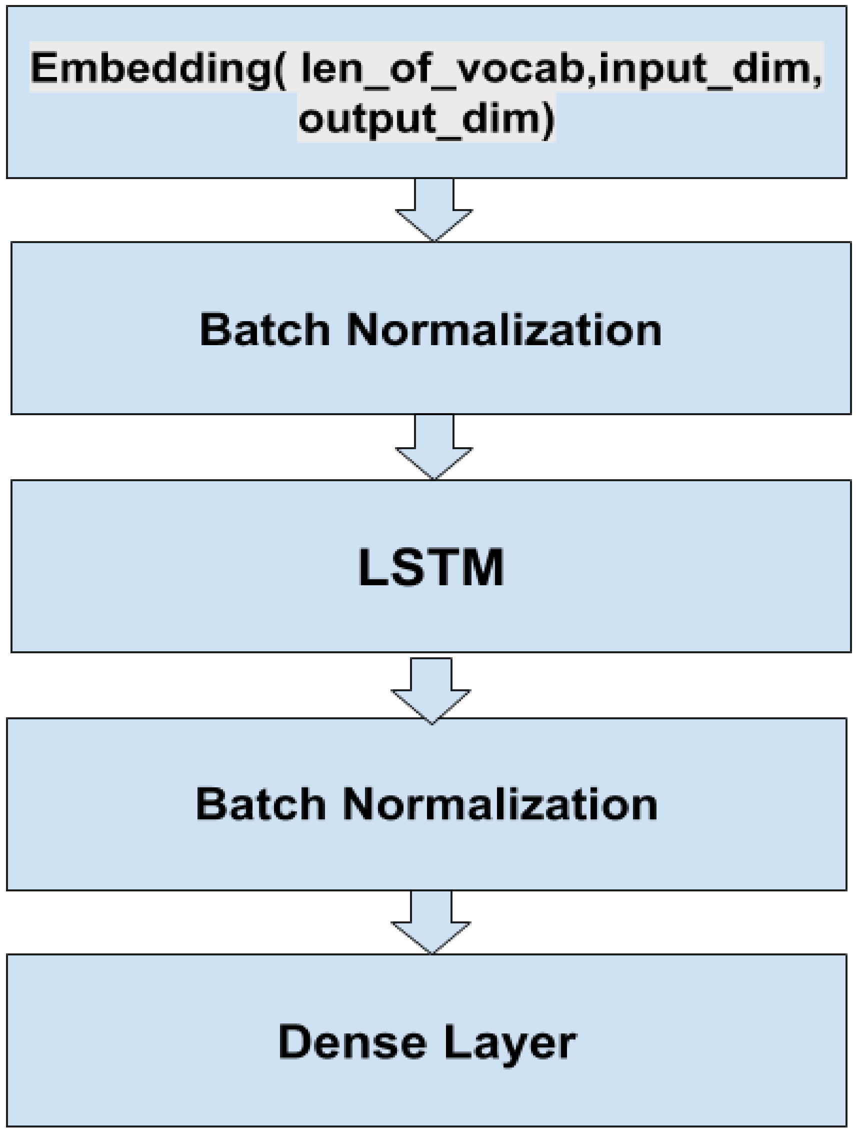
2. **Input gate:** Addition of useful information to the cell state is done by input gate. First, the information is regulated using the sigmoid function and filter the values to be remembered similar to the forget gate using inputs *h_t-1* and *x_t*. Then, a vector is created using the tanh function that gives output from -1 to +1, which contains all the possible values from h_t-1 and *x_t*. Atlast, the values of the vector and the regulated values are multiplied to obtain the useful information



3. **Output gate:** The task of extracting useful information from the current cell state to be presented as an output is done by the output gate. First, a vector is generated by applying tanh function on the cell. Then, the information is regulated using the sigmoid function and filters the values to be remembered using inputs *h_t-1* and *x_t*. Atlast, the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell.



MODEL USED IN OUR PROJECT



We have initialised a Sequential model followed by the below mentioned layers. A Sequential model is appropriate for **a plain stack of layers** where each layer has **exactly one input tensor and one output tensor**.

Embedding Layer: The first layer of our model is an Embedding layer. The Embedding layer in keras turns positive integers (indexes) into dense vectors of fixed size. This layer can only be used as the first layer of the model. In this step we have used parameters as the length of our word2indx mapping and 100 as input dimension. The output of this layer will be the dense vectors, each of dimension 100.

Batch Normalization Layer: The output of the Embedded layer was then feeded to the Batch Normalization layer. The Batch Normalization layer in keras normalizes the input. Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. Batch normalization is a layer that allows every layer of the network to do learning more independently. It is used to normalize the output of the previous layers. The activations scale the input layer in normalization. Using batch normalization learning becomes efficient also it can be used as regularization to avoid overfitting of the model. The layer is added to the sequential model to standardize the input or the outputs.

LSTM: After applying normalization on input, we have added an LSTM layer with 128 as dimensionality of the output space. The LSTM transforms the vector sequence into a single vector of size 128, containing information about the entire sequence.

Batch Normalization Layer: After applying a LSTM model layer, we have again added a Batch Normalization layer to normalize the output produced by the LSTM layer.

Dense Layer: The output of Normalization layer was then feeded to the Dense layer with **softmax** as an activation function. Softmax function outputs numbers that represent probabilities, each number's value is between 0 and 1 valid value range of probabilities. The range is denoted as [0,1]. The numbers are zero or positive. The entire output vector sums to 1. That is to say when all probabilities are accounted for, that's 100%. Thus, we will get one output vector representing probabilities of different words to be an answer. The word corresponding to highest probability will then be chosen as the answer.

MODEL TRAINING

The resultant list of sentences and target is passed through the model.
EPOCHS= 100
BATCH SIZE=100
The final model weight is stored in .h5 format to avoid retraining.

1. **english.h5**

   Sentences: 7115 Vocab Size: 13778 Batches: 71

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 100)         1377800
_________________________________________________________________
batch_normalization_1 (Batch (None, None, 100)         400
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               117248
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512
_________________________________________________________________
dense_1 (Dense)              (None, 13778)             1777362
=================================================================
Total params: 3,273,322
Trainable params: 3,272,866
Non-trainable params: 456
_________________________________________________________________
None
```

2. **hindi.h5**

   Sentences: 9416 Vocab Size: 16725 Batches: 94

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 100)         1672500
_________________________________________________________________
batch_normalization_3 (Batch (None, None, 100)         400
_________________________________________________________________
lstm_2 (LSTM)                (None, 128)               117248
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
dense_2 (Dense)              (None, 16725)             2157525
=================================================================
Total params: 3,948,185
Trainable params: 3,947,729
Non-trainable params: 456
_________________________________________________________________
None
```

3. **marathi.h5**
Sentences: 9368 Vocab Size: 21854 Batches: 93

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, None, 100)         2185400
_________________________________________________________________
batch_normalization_5 (Batch (None, None, 100)         400
_________________________________________________________________
lstm_3 (LSTM)                (None, 128)               117248
_________________________________________________________________
batch_normalization_6 (Batch (None, 128)               512
_________________________________________________________________
dense_3 (Dense)              (None, 21854)             2819166
=================================================================
Total params: 5,122,726
Trainable params: 5,122,270
Non-trainable params: 456
_________________________________________________________________
None
```

4. **punjabi.h5**
Sentences: 9873 Vocab Size: 18009 Batches: 98

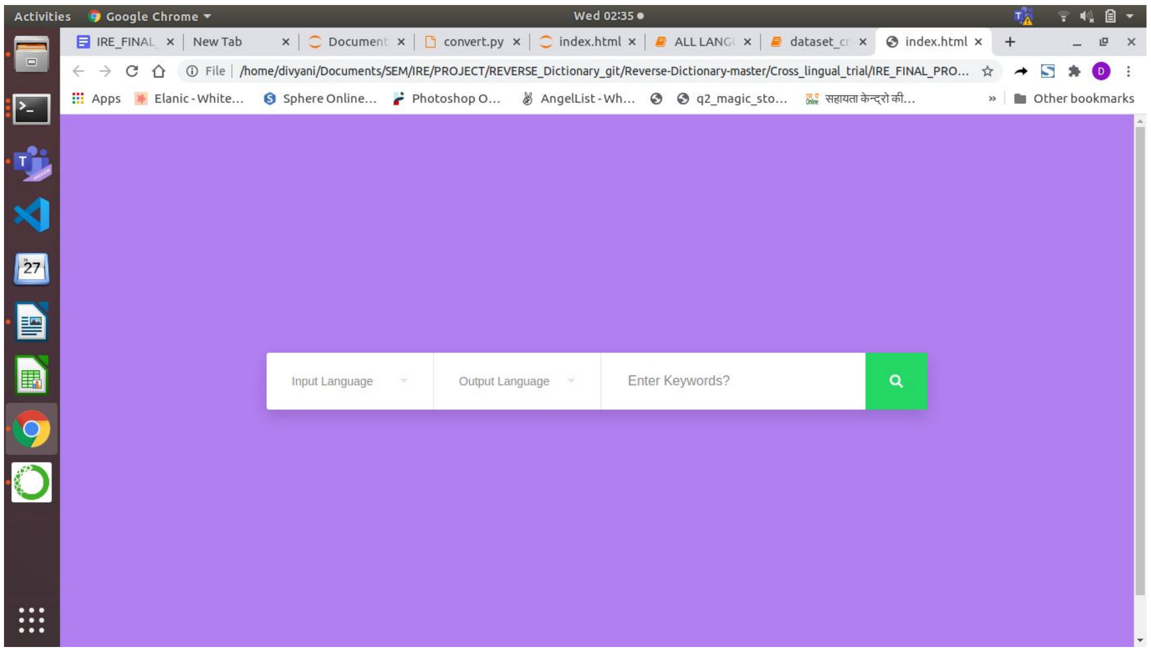
```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, None, 100)         1800900
_________________________________________________________________
batch_normalization_7 (Batch (None, None, 100)         400
_________________________________________________________________
lstm_4 (LSTM)                (None, 128)               117248
_________________________________________________________________
batch_normalization_8 (Batch (None, 128)               512
_________________________________________________________________
dense_4 (Dense)              (None, 18009)             2323161
=================================================================
Total params: 4,242,221
Trainable params: 4,241,765
Non-trainable params: 456
_________________________________________________________________
None
```

5. **bengali.h5**
Sentences: 9262 Vocab Size: 17455 Batches: 92

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, None, 100)        1745500
_________________________________________________________________
batch_normalization_9 (Batch (None, None, 100)        400
_________________________________________________________________
lstm_5 (LSTM)                (None, 128)              117248
_________________________________________________________________
batch_normalization_10 (Batc (None, 128)              512
_________________________________________________________________
dense_5 (Dense)              (None, 17455)            2251695
=================================================================
Total params: 4,115,355
Trainable params: 4,114,899
Non-trainable params: 456
_________________________________________________________________
None
```

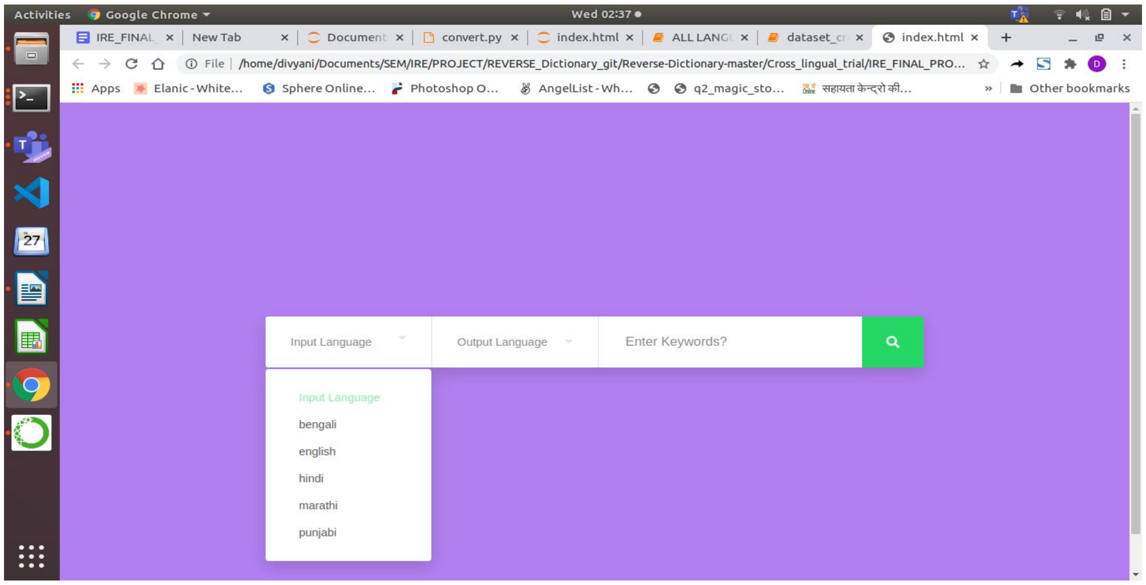
PHASE 3: Search Query Keyword Processing and displaying Result

The models are trained and the weights are stored so that they can directly be used at runtime. We have created a GUI along with python using Flask and HTML.



The above webpage takes 3 inputs:

1. Input Language: This is the language in which the definition or set of keywords will be searched
2. Output Language: This is the language for the desired output word.
3. Enter Keywords: Set of keyword for searching



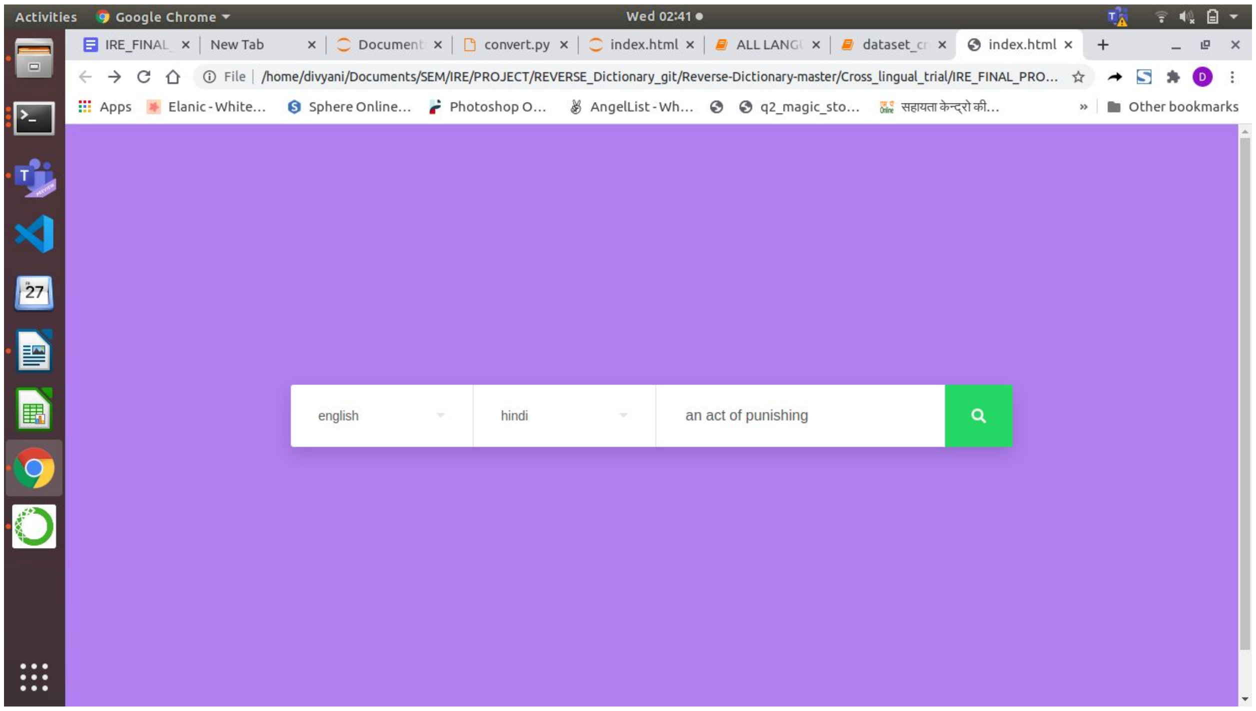
STEP 1: Processing the definition: The definition is processed using the same data processing function.

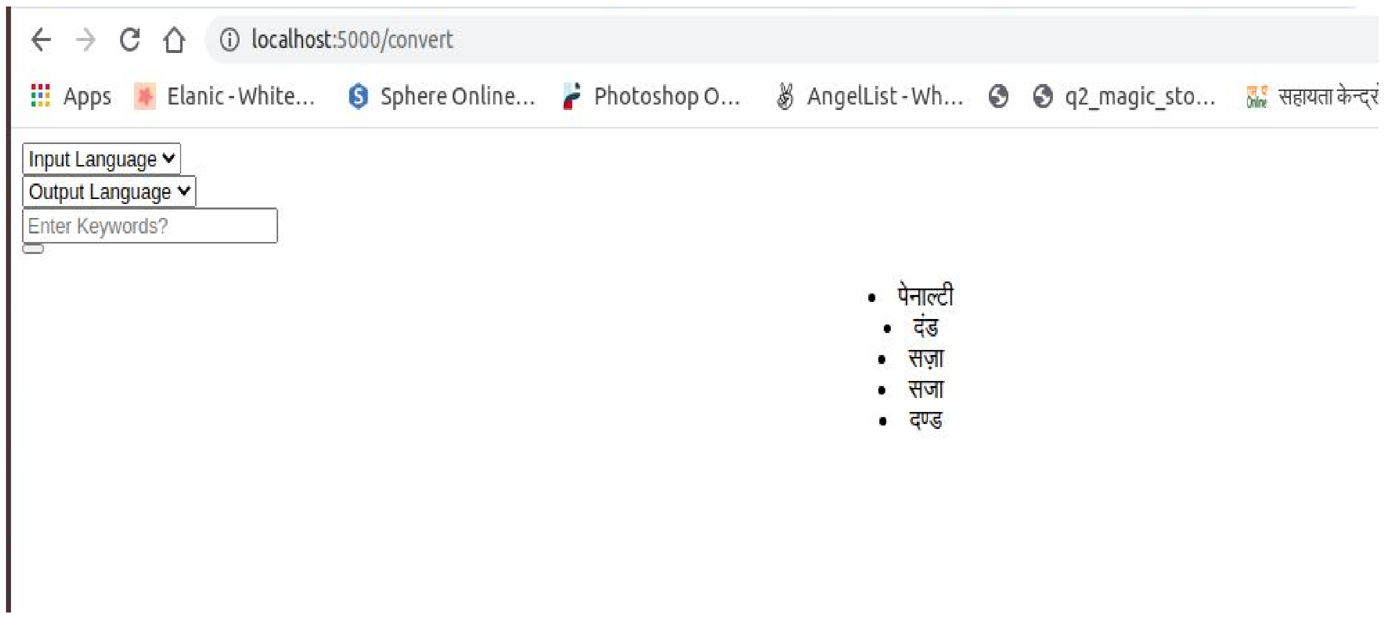
STEP 2: Model Prediction: the list of words is passed through the model. The model gives a vector of probabilities for each index. Each index is mapped with the word list.

STEP 3: Using Argmac on the resultant vector, we obtain the desired index.

STEP 4: USING the to_lang variable we map the from_lang word to to_lang word using a prestored dictionary.

STEP 5: Returning the list to the HTML page and Displaying the output.





- पेनाल्टी
- दंड
- सज़ा
- सजा
- दण्ड

As can be seen from the above images, we entered the sentence "An act of Punishing" and we got the results in the above image, which are very accurate.

RESULTS

For testing our model, we created a test data of mutually exclusive word_defintion pairs. and tried on our model.
We tested the above model on a test data of mutually exclusive word_defintion pairs.
TEST DATA SIZE 848 words
Accuracy : 21 %
The above accuracy is based on the matching of synonyms as well as main words. As can be seen on a small training dataset the accuracy is 21% when no definition is common. As we increase the dataset and provide much more definitions for the same word the accuracy can increase significantly.

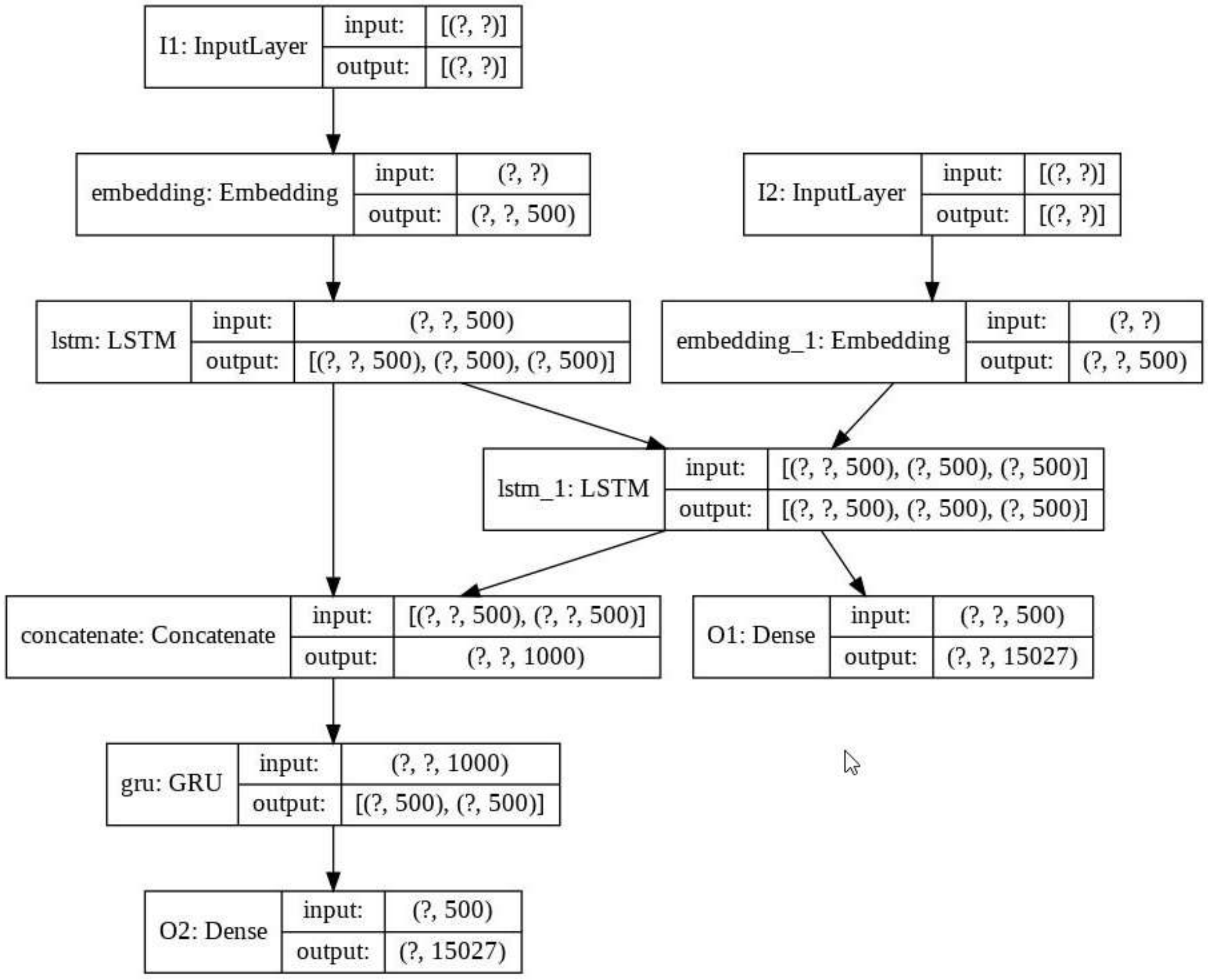
This code trains a sequence to sequence (seq2seq) model for English to Hindi translation with reverse dictionary feature.

After training the model we will be able to input a English sentence, such as *"someone who maintains and audits business accounts"*, and return the Hindi translation: *"START_ वह कर्मचारी जो आय-व्यय आदि का हिसाब लिखता या रखता हो _END" and word "**गणक**"*

The translation quality is reasonable, but the generated attention and its corresponding word is perhaps more interesting.

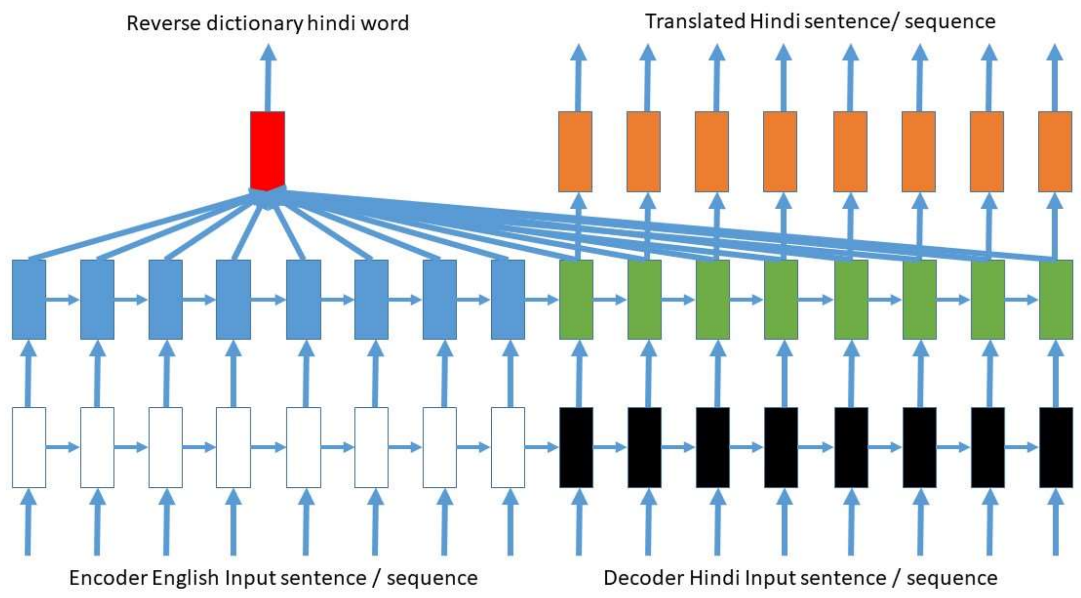
Encoder-Decoder model

Implement an encoder-decoder model with attention, which is similar to Neural Machine Translation (seq2seq) tutorial.



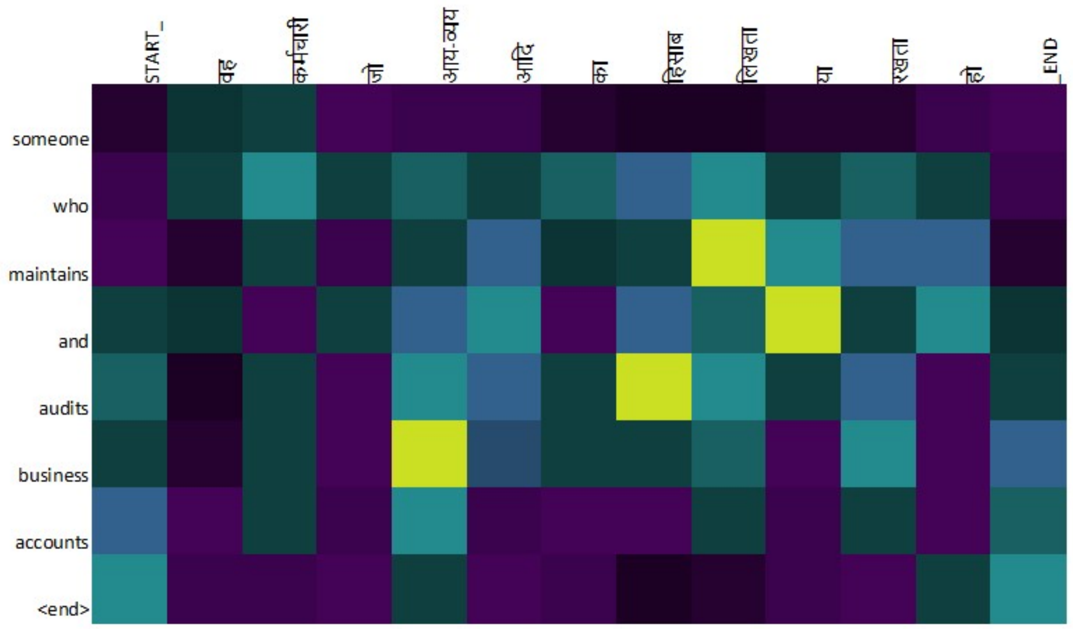
I1: InputLayer | input: [(?, ?)] | output: [(?, ?)]
embedding: Embedding | input: (?, ?) | output: (?, ?, 500)
I2: InputLayer | input: [(?, ?)] | output: [(?, ?)]
lstm: LSTM | input: (?, ?, 500) | output: [(?, ?, 500), (?, 500), (?, 500)]
embedding_1: Embedding | input: (?, ?) | output: (?, ?, 500)
lstm_1: LSTM | input: [(?, ?, 500), (?, 500), (?, 500)] | output: [(?, ?, 500), (?, 500), (?, 500)]
concatenate: Concatenate | input: [(?, ?, 500), (?, ?, 500)] | output: (?, ?, 1000)
O1: Dense | input: (?, ?, 500) | output: (?, ?, 15027)
gru: GRU | input: (?, ?, 1000) | output: [(?, 500), (?, 500)]
O2: Dense | input: (?, 500) | output: (?, 15027)

Which can be realized as


Reverse dictionary hindi word
Translated Hindi sentence/ sequence
Encoder English Input sentence / sequence
Decoder Hindi Input sentence / sequence

Here the reverse dictionary word is predicted along with translated sequence.

This shows which parts of the input sentence has the model's attention while translating:



Thus a word to word mapping is realized though this model and though the translation are not so perfect but gives or shows a significant direction towards the right word with much more complex dataset. Currently we use the data set from a parallel dataset of ~11k samples from IndoWordNet in 5 languages - English, Hindi, Marathi, Punjabi and Bengali.