

AUTOMATIC SUBJECTIVE ANSWER GRADING (FOR HINDI ESSAYS)

**Major Project Report 8th Semester (2019–2020)
Indian Institute of Engineering Science &
Technology, Shibpur**



Under Dr. Samit Biswas

Team members –

1. Mrityunjoy Kumar(510515013)
 2. Dyotana Das(510516014)
 3. Rituparna Biswas(510516017)
 4. Kumar Subhodeep Guin(510516045)
-

ACKNOWLEDGMENT

We hereby convey our sincere respect, appreciation, and gratitude to our mentor Dr. Samit Biswas, Prof. Dept. of Computer Science & Technology, Indian Institute of Engineering Science and Technology, Shibpur for allowing us to work on this thesis under his supervision and for his inspiration, ideas, and suggestions to improve this work. The project would not have been successful without his perseverance and constant encouragement.

CONTENTS

1. Abstract
2. Introduction
3. Proposed Work
4. Data Collection and Data Description
5. Data Preprocessing
6. Word2vec
7. Modeling
8. Long Short Term Memory(LSTM)
9. Training Phase
10. Result Analysis
11. Conclusion
12. References

ABSTRACT

The evaluation of answer papers considering semantics is a complex process that requires great intellectual effort from evaluators. The lack of availability of expert evaluators makes the evaluation more time-consuming. Nowadays, everything is automated. Hence, in order to reduce the effort during the evaluation of answer scripts an automated system is required to grade the answer scripts correctly. This paper presents a system for descriptive answer checking and grading application based on natural language processing and deep learning. Features are extracted to create a model from the human evaluated sample dataset of answer scripts. The proposed sequential model consists of the LSTM layer which sequentially takes the Word2vec vector representation in a sentence of each word and converts to an embedding vector representation. Embedding vectors corresponding to the Word2vec vector of the last word will be the representation of the entire sentence in its semantic form.

The model can assign scores of non-evaluated descriptive answers by comparing it with the answer key. This approach is very useful in the valuation of essays, descriptive answer scripts, document similarity checking, and plagiarism detection.

Till now, there is no such grading system for languages other than English, specially the ones localised for Indian languages such as Hindi. The main challenge of the grading system in Hindi starts right from the “Data Collection”. Very less data is available on the web to perform training of the model. More challenges include less tools for data-preprocessing due to the vast size of the language.

So, we have tried to develop such an essay grading model for Hindi language which does all this and is the first of its kind.

INTRODUCTION

One of the significant parts of education is the examination which is a measure of students learning the ability. After examination, the teachers spend most of their time evaluating the marks of the students and the evaluation takes bulk usage of human effort, time, and cost. An automated assessment evaluation system can reduce the efforts during the evaluation. Today, many automated evaluation systems exist and they analyze a piece of text based on semantics, spelling, and context. The evaluation of descriptive answers is still an open problem. A major problem among the existing systems is their efficiency. The subjective nature of the answer scripts evaluation corresponds to variations in awarding of grades by different human evaluators, which is seen as an unfair method of grading by students.

This difficulty of grading answer sheets can be rectified by answer script evaluation tools which grade answer scripts automatically. An automated assessment system must be capable of scoring the answer papers within the range of those awarded by human evaluators. It must be consistent in the way it grades the answer scripts and thus it can save the time and cost of the evaluation.

The objective of this model is to extract the semantics to efficiently represent the text in answer scripts and develop a model from the key and evaluated answer scripts to grade non evaluated answer scripts using deep neural networks. It is a combination of NLP and machine learning. The learning is done by using the Recurrent Neural Network and LSTM cells. Deep Neural networks are able to capture the semantics of text in order to and the similarity between texts. To efficiently represent the semantics of the sentences as embedding vectors we use LSTM.

In the previous semester, we have already developed an essay grading system for English language. It was quite successful with accuracy of approx 96%. In this project, we have tried to create a new grading system for Hindi essays, which is the first of its kind.

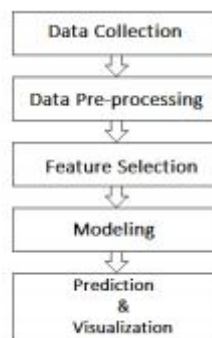
RELATED WORK

- In 1996, by the request from American College Board, Ellis Page developed the first AES system, Project Essay Grader. The Project Essay Grader analyzes the essays to determine a huge set of text features, e.g. to predict mark that human evaluators give, the fourth root of essay length uses an approach with regression. It **fails to detect the content-related features of an essay** by mainly aiming at the surface structures and ignoring the semantic aspect of essays.
- In 1995, Ranjit Biswas has explained an answer paper evaluation system based on fuzzy sets known as the Fuzzy Evaluation Method(FEM). The paper compares the traditional evaluation approaches and automated grading approaches. A value marking using vector technique is used by Fem which uses a fuzzy technique of computer system.
- In 2003, Leacock and Chodorow have implemented an automated short answer scoring system, C-rater. C-rater is used to score responses to content-based short answer questions. It uses predicate-argument structure, pronominal reference, morphological analysis, and synonyms to assign full or partial credit to a short answer question.
- in 2017, Philip E. Robinson et al. have presented an online learning platform for assessment, teaching, and learning of programming. Jamsheedh C. V has implemented a basic answer script evaluation system by using NLP and machine learning tools. Here, features are extracted from the human evaluated sample dataset of answer scripts, and high weight is given answer key.
- Ming Che Lee et al. [11] have presented a paper that contains grammar and semantic corpus-based similarity algorithm for natural language sentences. In order to surmount the problems addressed, the sentence similarity algorithm takes advantage of corpus-based ontology and grammatical rules.

PROPOSED WORK

Many schemes and methods are currently available for the evaluation of essays. But automatic evaluation and grading are not attempted successfully for descriptive answer scripts. Many concepts of NLP have been applied in finding the relevant words, extracting the meaning, and assigning a grade for different answers. The system takes the entire short answer as input and converts it into glove vector representation by using the embedding layer. The LSTM will learn the temporal data from the embedding layer and the embedding vector corresponding to the final glove vector will be the semantic representation of the entire answer. This is given as input to the dropout layer and then to the fully connected neural network layer with a RELU activation function.

The developed system is comprised of three stages **Preprocessing, Semantic Extraction, Classification, and Grading**. The preprocessing phase converts the answers in the dataset into a set of an index to glove vectors corresponding to each sentence. The Semantic Extraction phase makes the semantic representation of the entire answer by using the Embedding layer and LSTM-RNN layer. The embedding vector representation from the LSTM-RNN model is further used for the classification and grading phase.



Different steps in the proposed model are explained below:

DATA COLLECTION AND DATA DESCRIPTION

Supervised machine learning is totally dependent upon the condition of the data. A good and credible dataset can create the difference between good and bad machine learning agents. In our case, finding a large dataset of essays that could be used in our research wasn't easy. After much hassle, we have created the right usable dataset consisting of 600 Hindi essays. These essays are marked by two human raters. The dataset contains the following parameters :

STRUCTURE OF THE TRAINING DATASET:

TABLE-1

Name	Type	Description
essay_id	Numeric	A unique identifier for each individual essay set
essay_set	Numeric	A unique identifier for each individual essay in a set
essay	String	The ASCII text of a student's response
domain1_score	Numeric	Teacher 1 rates the answer script
domain2_score	Numeric	Teacher 2 rates the answer script
rater_score	Numeric	The human rater's score for the answer. This is the final score for the answer and the score that you are trying to predict.

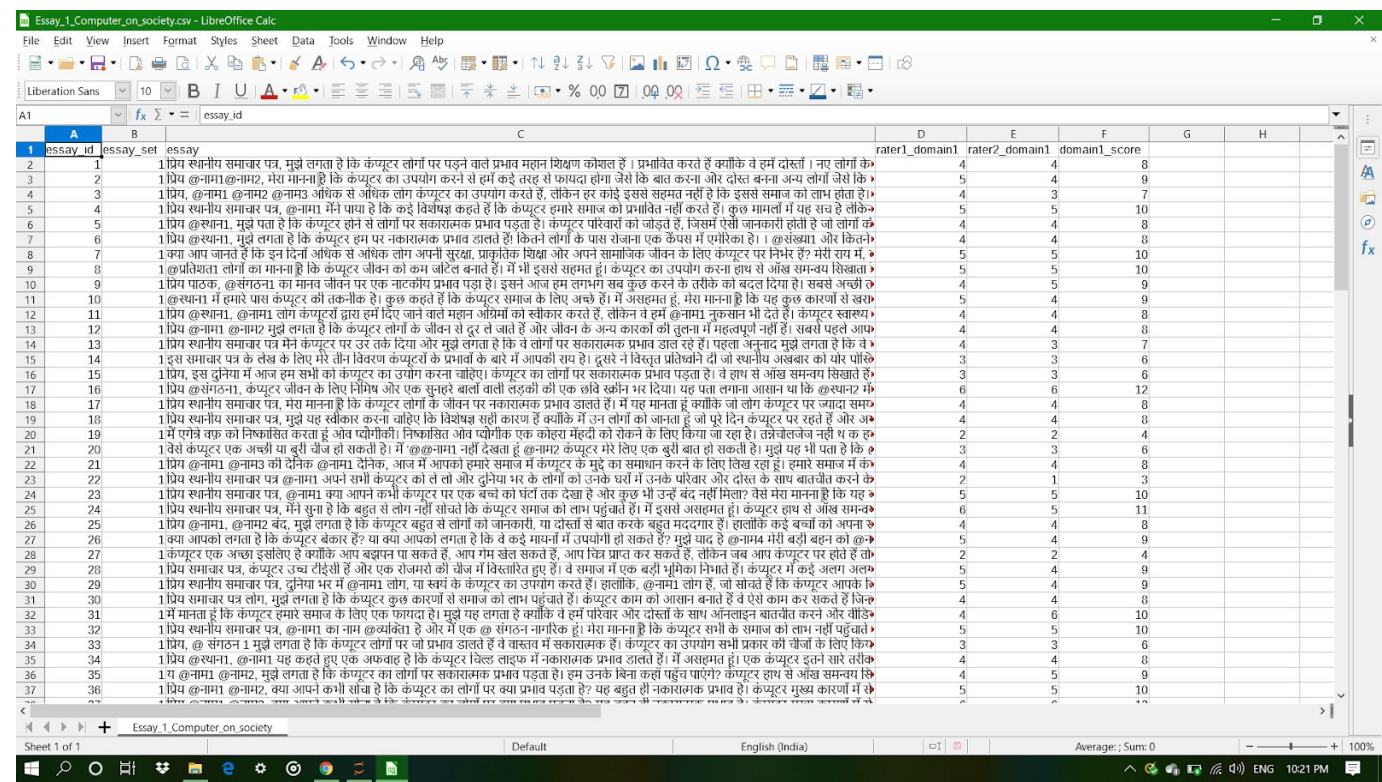
Testing Dataset is similar to the training dataset (as shown in Table-2).

STRUCTURE OF THE TEST DATASET:

TABLE-2

Name	Type	Description
essay_id	Numeric	A unique identifier for each individual essay set
essay_set	Numeric	A unique identifier for each individual essay in a set
essay	String	The ASCII text of a student's response

Screenshot of the dataset after collecting:



essay.head() gives the following output :-

-----Essay 1-----

	essay_id	essay_set	essay \
0	1	1	प्रिय स्थानीय समाचार पत्र, मुझे लगता है कि कंप...
1	2	1	प्रिय @नाम1@नाम2, मेरा मानना है कि कंप्यूटर का...
2	3	1	प्रिय, @नाम1 @नाम2 @नाम3 अधिक से अधिक लोग कंप...
3	4	1	प्रिय स्थानीय समाचार पत्र, @नाम1 मैंने पाया है...
4	5	1	प्रिय @स्थान1, मुझे पता है कि कंप्यूटर होने से...

	rater1_domain1	rater2_domain1	domain1_score
0	4	4	8
1	5	4	9
2	4	3	7
3	5	5	10
4	4	4	8

DATA PREPROCESSING

Data preprocessing is the first step of any data mining approach. Data preprocessing is needed to convert the raw unordered unusable data to a structured usable format. Also, the dataset contains a lot of outliers and other noises that could affect the research in a negative manner. However, the dataset that we have collected was very much in shape to be used in our research. A slight modification was made to remove the outliers and other noises like some missing values removal and so on. Later on, we have made the following modifications to our dataset while using them so that we could use them in a better way.

- The dataset contains answers with non-hindi characters and needs to be replaced by spaces. For every set of data we will replace non-alphanumeric characters by space because data makes references to experimental numerical data and those numerical data may influence scoring.
- We have preprocessed all answers and **converted them to feature vectors** so that they can be fed into the model as almost all Deep Learning Architectures are incapable of processing *strings* or *plain text* in their raw form.
- We have removed all leading space, punctuations, trailing spaces, stopwords, zero-width (\u200b) white spaces. Then converted sentences into a list of words which are the input to the Word2Vec.

To convert the text into vectors we have used **Word2Vec**.

The idea of Natural Language Processing is to do some form of analysis, or processing, where the machine can understand, at least to some level, what the text means, says, or implies.

Text Processing for Indian Languages using Python:

There are a handful of Python libraries we can use to perform text processing and build NLP applications for Indian languages. I've put them together:

Punkt

Punkt Sentence Tokenizer

This tokenizer divides a text into a list of sentences, by using an unsupervised algorithm to build a model for abbreviated words, collocations, and words that start sentences. It must be trained on a large collection of plaintext in the target language before it can be used.

The NLTK data package includes a pre-trained Punkt tokenizer for Hindi.

(Whitespace from the original text, including newlines, is retained in the output.) Punctuation following sentences is also included by default (from NLTK 3.0 onwards).

Stopwords

One of the major forms of pre-processing is to filter out useless data. In natural language processing, useless words (data), are referred to as stopwords.

A stop word is a commonly used word (such as “मैं”, “ऊँ”, “सो”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.

```
P = pd.read_csv( 'C:/Users/Chotu/Documents/hindi_stopwords.csv')
stop_words=P['stopwords']
stop_words
```

0	अगर
1	अत
2	अथवा
3	अंदर
4	अदि
...	...
299	होति
300	होती
301	होते
302	होना
303	होने

Name: stopwords, Length: 304, dtype: object

We can even modify the list by adding words of your choice in the hindi.txt. file in the stopwords directory.

Preprocessing code snippet :

```
1 import nltk
2 #nltk.download('punkt')
3 from string import digits
4 import re
5 import string
6 from nltk.tokenize import word_tokenize
7 sentenceEnders = re.compile('[!?\']')
8 ge = re.compile(' ')
9 wordy1 = []
10 wordy2= []
11 wordy3=[]
12 flag = 0
13 train_essay = D['essay']
14 for i in range(len(train_essay)):
15     m=[]
16     if i>=0:
17         essay_v = train_essay[i]
18         essay_v =re.sub("\u200b","",essay_v)
19         essay_v.strip()
20         X = sentenceEnders.split(essay_v)
21
22         wordy=[]
23         for sentence in X:
24             p = []
25             tokens = word_tokenize(sentence)
26             remove_digits = str.maketrans('', '', digits)
27             list = [i.translate(remove_digits) for i in tokens]
28             table = str.maketrans('', '', string.punctuation)
29             words = [w.translate(table) for w in list ]
30             for word in words:
31                 flag = 0
32                 for stop_word in stop_words:
33                     if word == stop_word or word == '':
34                         flag = 1
35                         #print(word+" "+stop_word)
36                 if flag == 0:
37                     #print(word)
38                     wordy.append(word)
39                     p.append(word)
40             if len(p)>1:
41                 wordy2.append(p)
42                 m.append(p)
43             wordy1.append(wordy)
44         wordy3.append(m)
```

Each essay after preprocessing is left with list of useful words as shown below :

```
600
wordy1
600
['प्रिय', 'स्थानीय', 'समाचार', 'पत्र', 'मुझे', 'लगता', 'कंप्यूटर', 'लोगों', 'पढ़ने', 'प्रभाव', 'महान', 'शिक्षण', 'कौशल', 'प्रभावित', 'क्योंकि', 'ह',
मे', 'दोस्तों', 'नए', 'लोगों', 'चैट', 'देते', 'हमें', 'ग्लोब', 'खगोल', 'विज्ञान', 'बारे', 'जानने', 'मदद', 'हमें', 'रोकते', 'बारे', 'आपको', 'लगता',
'किशोर', 'हमेशा', 'दोस्तों', 'फोन', 'रहता', 'आपको', 'कसा', 'लगेगा', 'चीजों', 'बारे', 'दोस्तों', 'व्यपर', 'साथी', 'चैट', 'खैर', 'कंप्यूटर', 'चै
ट', 'नया', 'तरीका', 'इंटरनेट', 'सारी', 'साइटें', 'संगठन', 'संगठन', 'फेसबुक', 'माईस्पेस', 'ईत्यादी', 'सोचिए', 'कंप्यूटर', 'बॉस', 'मिलना', 'शु
रू', 'किशोर', 'फोन', 'मस्ती', 'क्योंकि', 'इस्तेमाल', 'चाहते', 'आपने', 'देशों', 'राज्यों', 'बारे', 'सीखा', 'वैसे', 'मेरे', 'पास', 'कंप्यूटर', 'इंटर
नेट', 'हमारे', 'चल', 'जानने', 'नया', 'तरीका', 'सोच', 'बच्चा', 'कंप्यूटर', 'बिताता', 'उनसे', 'अर्थव्यवस्था', 'समुद्री', 'तल', 'फैलने', 'दिनांक',
'बारे', 'इतना', 'सवाल', 'पूछें', 'जानते', 'आश्चर्यचकित', 'रह', 'जाएंगे', 'कंप्यूटर', 'दिलचस्प', 'कक्षा', 'दिन', 'किताबों', 'पढ़ना', 'बच्चा', 'आप
के', 'कंप्यूटर', 'स्थानीय', 'पुस्तकालय', 'दोस्तों', 'फ्रेश', 'तैयार', 'बेहतर', 'सही', 'जानते', 'आपको', 'शायद', 'पता', 'होगा', 'बच्चा', 'अस्प
ताल', 'बिस्तर', 'झाड़वबाय', 'कारण', 'मना', 'बच्चे', 'बजाय', 'कंप्यूटर', 'सीखने', 'चैट', 'सिर्फ', 'गेम', 'खेलने', 'सामुदायिक', 'सुरक्षित', 'स्व
स्थ', 'रहें', 'मुझे', 'आशा', 'मुझे', 'समझने', 'उससे', 'सहमत', 'बिंदु', 'पहुंच', 'क्योंकि', 'कंप्यूटर', 'आपके', 'बच्चे', 'प्रभाव', 'डाल', 'क्
योंकि', 'हमें', 'दोस्तों', 'नए', 'लोगों', 'चैट', 'देता', 'हमें', 'ग्लोब', 'बारे', 'जानने', 'मदद', 'हमें', 'विश्वास', 'रखता', 'रखता', 'सुनने', 'धन्य
वाद']
```

WORD2VEC

Word embedding is the concept of mapping from discrete objects such as words to vectors and real numbers.

Word2vec is the most common approach used for unsupervised word embedding technique. It trains the model in such a way that a given input word predicts the word's context by using skip-grams.

TensorFlow enables many ways to implement this kind of model with increasing levels of sophistication and optimization and using multithreading concepts and higher-level abstractions.

Implementation in python package gensim:

```
34 num_features = 300
35 min_word_count = 10
36 num_workers = 4
37 context = 16
38 downsampling = 1e-1
39
40
41 print("Training Word2Vec Model...")
42 model=Word2Vec(sentences,workers=num_workers,size=num_features,min_count=min_word_count>window=context,sample=downsampling,
43
44 #print(model)
45 model.init_sims(replace=True)
46 model.wv.save_word2vec_format('word2vecmodel.bin', binary=True)
```

Size: the dimensionality of the feature vectors (commonly used: 200 or 300)

Window: the maximum distance between the current and predicted word within a sentence

Min_count: minimum number of occurrences of a word in the corpus to be included in the model

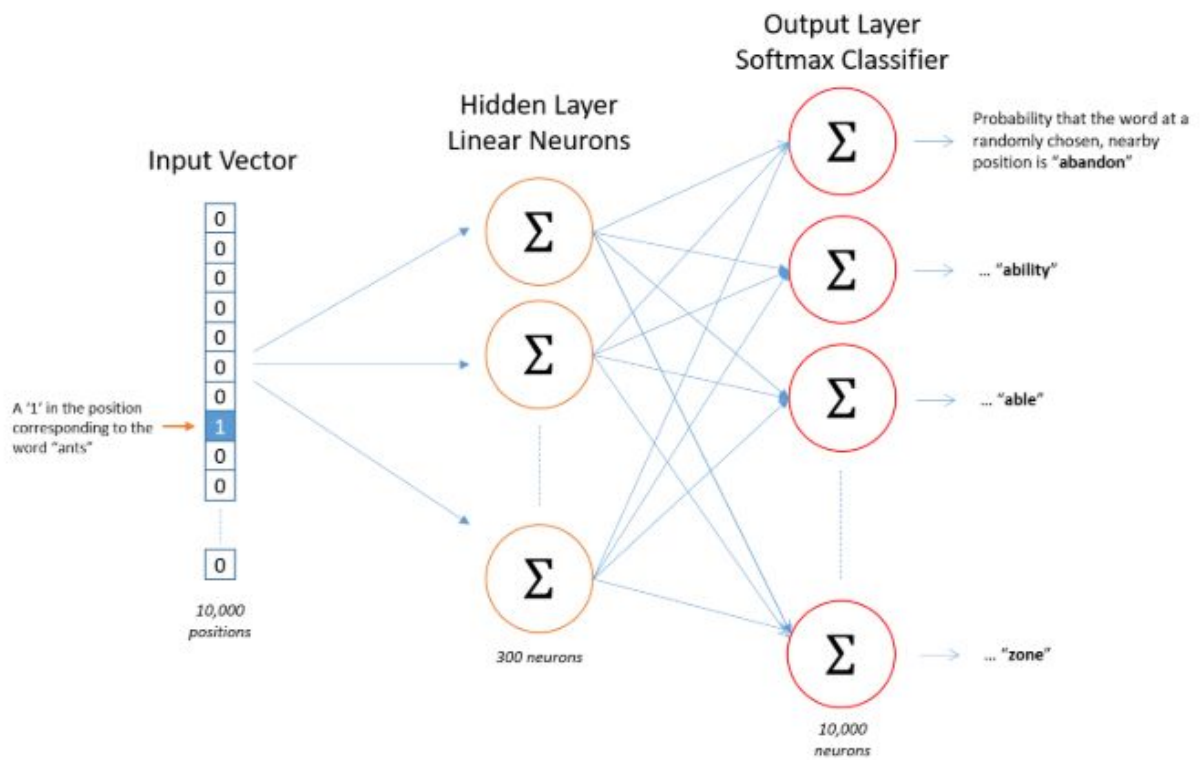
Workers: for parallelization with multicore machine

Word2vec model comes in two flavors: **Skip Gram Model** and **Continuous Bag of Words Model (CBOW)**. Here we have used the Skip Gram Model.

Advantages of Word2Vec:

Word2Vec has several advantages over a bag of words and TF-IDF schemes. Word2Vec retains the semantic meaning of different words in a document. The context information is not lost. Another great advantage of the Word2Vec approach is that the size of the embedding vector is very small. Each dimension in the embedding vector contains information about one aspect of the word.

We do not need huge sparse vectors, unlike the bag of words and TF-IDF approaches.



Neural Network for the Skip-Gram model

MODELING

The proposed system is using Long Short Term Memory. Long short-term memory (LSTM) is an artificial **recurrent neural network** (RNN) architecture used in the field of **deep learning**.

```
In [8]: 1 from keras.layers import Embedding, LSTM, Dense, Dropout, Lambda, Flatten
        2 from keras.models import Sequential, load_model, model_from_config
        3 import keras.backend as K
        4
        5 def get_model():
        6     """Define the model."""
        7     model = Sequential()
        8     model.add(LSTM(300, dropout=0.2, recurrent_dropout=0.2, input_shape=[1, 300], return_sequences=True))
        9     model.add(LSTM(64, recurrent_dropout=0.2))
       10     model.add(Dropout(0.2))
       11     model.add(Dense(1, activation='relu'))
       12
       13     model.compile(loss='mean_squared_error', optimizer='rmsprop', metrics=['mae'])
       14     model.summary()
       15
       16     return model
```

Using TensorFlow backend.

The Sequential model API is a way of creating deep learning models where an instance of the Sequential class is created and model layers are created and added to it.

The first layer is an LSTM layer with 300 memory units and it returns sequences. This is done to ensure that the next LSTM layer receives sequences and not just randomly scattered data.

A dropout layer is applied after the 1st LSTM layer to avoid the overfitting of the model. Finally, we have the last layer as a fully connected layer with a '*ReLU*' activation function.

A metric is a function that is used to judge the performance of your model. Metric functions are to be supplied in the **metrics** parameter when a model is compiled. We have used

Rmsprop optimizer:

The central idea of RMSprop is to keep the moving average of the squared gradients for each weight. And then we divide the gradient by the square root of the mean square. Which is why it's called RMSprop (root mean square).

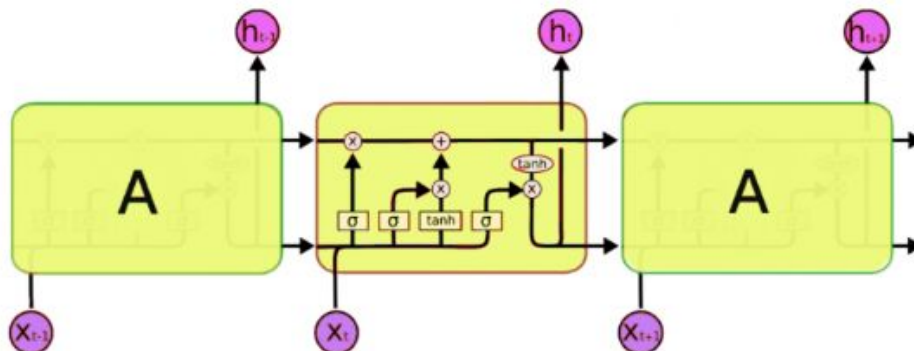
LONG SHORT TERM MEMORY

RNN remembers things for just small durations of time, i.e. if we need the information after a small-time it may be reproducible, but once a lot of words are fed in, this information gets lost somewhere. This issue can be resolved by applying a slightly tweaked version of RNNs – the **Long Short-Term Memory Networks**.

LSTMs make small modifications to the information by multiplications and additions. With LSTMs, the information flows through a mechanism known as cell states. This way, LSTMs can selectively remember or forget things. The information at a particular cell state has three different dependencies:

- i)The previous cell state (*i.e. the information that was present in the memory after the previous time step*)
- ii)The previous hidden state (*i.e. this is the same as the output of the previous cell*)
- iii)The input at the current time step (*i.e. the new information that is being fed in at that moment*)

The architecture of LSTMs:



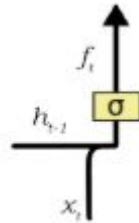
A typical LSTM network is comprised of different memory blocks called **cells**(the rectangles that we see in the image).

There are two states that are being transferred to the next cell:

1. The **cell state**
- 2.The **hidden state**.

The memory blocks are responsible for remembering things and manipulations to this memory is done through three major mechanisms, called **gates**.

FORGET GATE:



A forget gate is responsible for removing information from the cell state.

The information that is no longer required for the LSTM to understand things or the information that is of less importance is removed via the multiplication of a filter. This is required for optimizing the performance of the LSTM network.

This gate takes in two inputs; h_{t-1} and x_t .

H_{t-1} : The hidden state from the previous cell or the output of the previous cell. x_t : The input at that particular time step.

The given inputs are multiplied by the weight matrices and a bias is added. Following this, the sigmoid function is applied to this value. The sigmoid function outputs a vector, with values ranging from 0 to 1, corresponding to each number in the cell state.

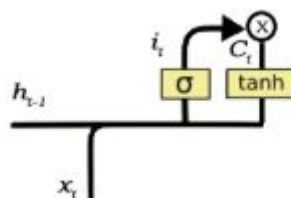
Basically, the sigmoid function is responsible for deciding which values to keep and which to discard. If a '0' is output for a particular value in the cell state, it means that the forget gate wants the cell state to forget that piece of information completely.

Similarly, a '1' means that the forget gate wants to remember that entire piece of information.

This vector output from the sigmoid function is multiplied to the cell state.

INPUT GATE:

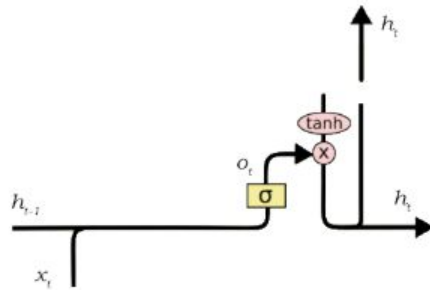
The input gate is responsible for the addition of information to the cell state. This addition of information is basically a three-step process.



1. Regulating what values need to be added to the cell state by involving a sigmoid function.

2. Creating a vector containing all possible values that can be added (as perceived from h_{t-1} and x_t) to the cell state. This is done using the \tanh function, which outputs values from -1 to +1.
3. Multiplying the value of the regulatory filter (the sigmoid gate) to the created vector (the \tanh function) and then adding this useful information to the cell state via addition operation.

OUTPUT GATE:



This job of selecting useful information from the current cell state and showing it out as output is done via the output gate.

1. Creating a vector after applying **the \tanh function** to the cell state, thereby scaling the values to the range -1 to +1.
2. Making a filter using the values of h_{t-1} and x_t , such that it can regulate the values that need to be output from the vector created above. This filter again employs a **sigmoid function**.
3. Multiplying the value of this regulatory filter to the vector created in step 1, and sending it out as output and also to the hidden state of the next cell.

TRAINING PHASE

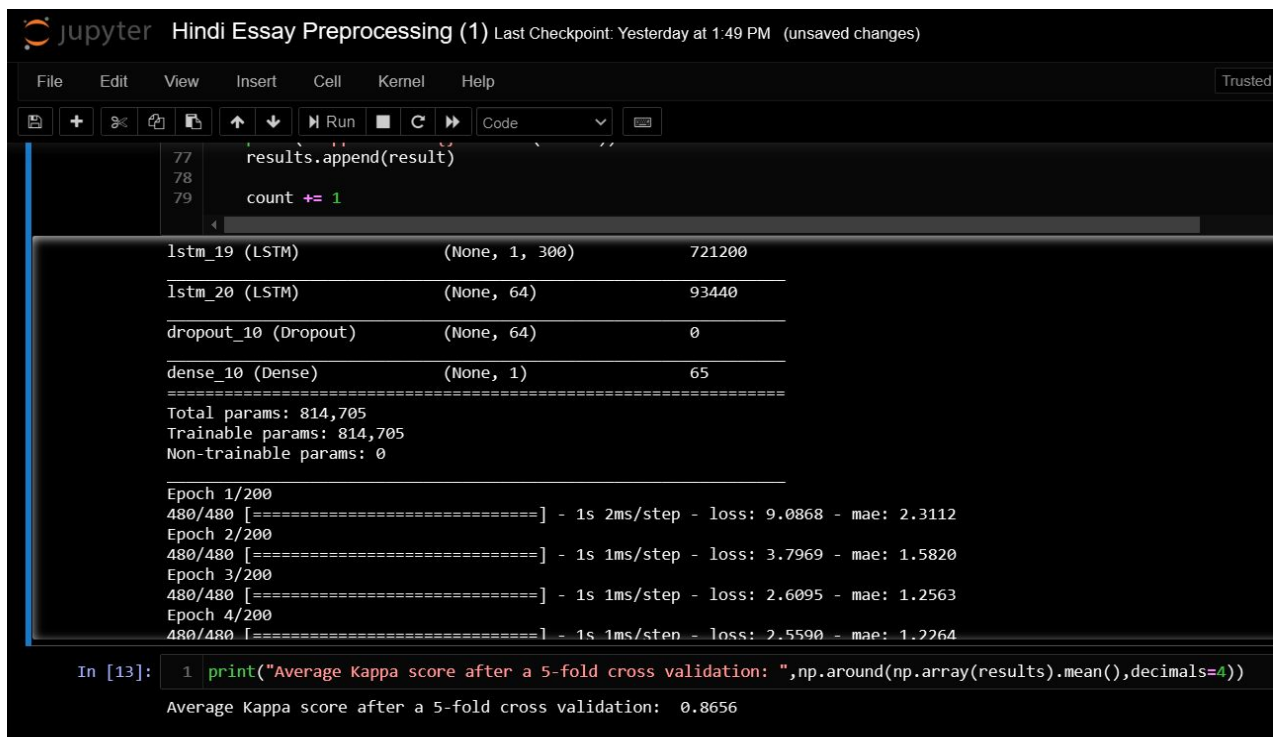
We have used 5-Fold Cross-Validation.

Inside each fold, we have trained the word2vec model from our training data. Then we have created the word embeddings of individual words in each essay. These word embeddings are then fed into the LSTM model for training. We have taken epochs as 200 and batch_size as 8. Now, we test the model with our test data and calculate the score using Quadratic Weighted Kappa.

Kappa is similar to Accuracy score, but it takes into account the accuracy that would have happened anyway through random predictions.

$$\text{Kappa score} = (\text{Observed Accuracy} - \text{Expected Accuracy}) / (1 - \text{Expected Accuracy})$$

Finally, we calculate the Average Kappa for all the folds.



```
77 results.append(result)
78
79 count += 1

lstm_19 (LSTM)              (None, 1, 300)          721200
lstm_20 (LSTM)              (None, 64)              93440
dropout_10 (Dropout)       (None, 64)              0
dense_10 (Dense)           (None, 1)               65
=====
Total params: 814,705
Trainable params: 814,705
Non-trainable params: 0

Epoch 1/200
480/480 [=====] - 1s 2ms/step - loss: 9.0868 - mae: 2.3112
Epoch 2/200
480/480 [=====] - 1s 1ms/step - loss: 3.7969 - mae: 1.5820
Epoch 3/200
480/480 [=====] - 1s 1ms/step - loss: 2.6095 - mae: 1.2563
Epoch 4/200
480/480 [=====] - 1s 1ms/step - loss: 2.5590 - mae: 1.2264

In [13]: 1 print("Average Kappa score after a 5-fold cross validation: ",np.around(np.array(results).mean(),decimals=4))
Average Kappa score after a 5-fold cross validation: 0.8656
```

According to Cohen's original article, values ≤ 0 as indicating no agreement and 0.01–0.20 as none to slight, 0.21–0.40 as fair, 0.41– 0.60 as moderate, 0.61–0.80 as substantial, and 0.81–1.00 as almost perfect agreement.

We have achieved an Average Cohen Kappa Score of 0.8656, which means our result is in almost perfect agreement.

RESULT ANALYSIS

The output of an AES system can be compared to the ratings assigned by human annotators using various measures of correlation or agreement (Yannakoudakis and Cummins, 2015). These measures include Pearson's correlation, Spearman's correlation, Kendall's Tau, and quadratic weighted Kappa (QWK).

The ASAP competition adopted QWK as the official evaluation metric. Since we use the ASAP data set for evaluation in this paper, we also use QWK as the evaluation metric in our experiments. The quadratic weighted Kappa is calculated as follows. First, a weight matrix W is constructed according to Equation 1:

$$W_{(i,j)} = (i - j)^2 / (N - 1)^2$$

where i and j are the reference rating (assigned by a human annotator) and the hypothesis rating (assigned by an AES system), respectively, and N is the number of possible ratings. A matrix O is calculated such that $O_{(i,j)}$ denotes the number of essays that receive a rating i by the human annotator and a rating j by the AES system. An expected count matrix E is calculated as the outer product of histogram vectors of the two (reference and hypothesis) ratings. The matrix E is then normalized such that the sum of elements in E and the sum of elements in O are the same.

Finally, given the matrices O and E , the QWK score is calculated according to Equation 2:

$$\kappa = 1 - ((W_{(i,j)} O_{(i,j)}) / (W_{(i,j)} E_{(i,j)})) \quad \text{for all } (i,j) \quad (2)$$

According to Cohen's original article, values ≤ 0 as indicating no agreement and 0.01–0.20 as none to slight, 0.21–0.40 as fair, 0.41–0.60 as moderate, 0.61–0.80 as substantial, and 0.81–1.00 as almost perfect agreement.

We have achieved an Average Cohen Kappa Score of 0.8656, which means our result is in almost perfect agreement.

CONCLUSION

In this project, we have proposed an approach based on the LSTM model to tackle the task of automated essay scoring for Hindi essays. Our method does not rely on any feature engineering and automatically learns the representations required for the task. We have explored a variety of neural network model architectures for automated essay scoring but we haven't found anything significant for Hindi essay scoring. Our system has created the baseline by getting a score of 0.8656 in terms of average quadratic weighted Kappa. Furthermore, an analysis of the network has been performed to get an insight into the LSTM model and we show that the method effectively utilizes essay content to extract the required information for scoring essays while preserving the context of words in the essay.

REFERENCES

- ❑ <https://www.analyticsvidhya.com/blog/2020/01/3-important-nlp-libraries-indian-languages-python/>
- ❑ <https://www.superdatascience.com/blogs/recurrent-neural-networks-rnn-long-short-term-memory-lstm/>
- ❑ <https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa>
- ❑ [1] Hanxiao Shi, Giodong Zhou and Peide Qian (2010),” An Attribute-based Sentiment Analysis System”, Information Technology Journal, PP 1607-1614.
- ❑ [2] Papri Chakraborty (2012),” Developing an Intelligent Tutoring System for Assessing Students' Cognition and Evaluating Descriptive Type Answer”, IJMER, PP 985-990.
- ❑ [3] Mita K. Dalal, Mukesh A. Zave (2011),” Automatic Text Classification: A Technical Review”, International Journal of Computer Applications, PP.37-40.
- ❑ <https://github.com/Kyubyong/wordvectors>
- ❑ <https://fasttext.cc/docs/en/crawl-vectors.html>
- ❑ <https://www.analyticsvidhya.com/blog/2020/01/3-important-nlp-libraries-indian-languages-python/>
- ❑ Towards Deep Learning in Hindi NER : An approach to tackle the Labelled Data Scarcity Vinayak Athavale*[1], Shreenivas Bharadwaj *[2], Monik Pamecha*[3], Ameya Prabhu [1] and Manish Shrivastava [1]