LSTM & TRANSFORMER BASED LANGUAGE MODELS FOR ODIA TEXT CLASSIFICATION

Ishan Das - 10532854

Applied Research Project submitted in partial fulfilment of the requirements for the degree of

MSc. Business Analytics

Dublin Business School

Supervisor: Abhishek Kaushik

August 2020

Declaration

I declare that this Applied Research Project that I have submitted to Dublin Business School for the award of MSc Business Analytics is the result of my own investigations, except where otherwise stated, where it is clearly acknowledged by references. Furthermore, this work has not been submitted for any other degree.

Signed: Ishan Das

Student Number: 10532854

Date: 20th Aug 2020

Acknowledgement

Foremost, I would like to express my sincere gratitude to my supervising professor, Abhishek Kaushik at Dublin Business School for guiding me through the field of NLP and providing a strong foundation for the commencement of my project. I would like to thank him for his continuous support throughout the research and for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the times of research and writing this dissertation. He provided encouragement and insightful comments.
Besides my supervisor, I would like to thank Dublin Business School for providing me the opportunity to complete my MSc.

Abstract

Text classification is a significant part of NLP and with products and services moving online, a lot of data is now available including language such as Odia. There is no denying that NLP tasks like text classification, sentiment analysis etc provide actionable insights to the businesses. With recent surge in deep-learning models NLP research has reached state-of-the art results. This includes architectures like Transformers and built on top the same – BERT models. Although substantial amount of research has been done in NLP using deep-learning models (LSTM & Transformers / BERT), very less progress has been made on languages like Odia. Odia being the native language of Odisha (a state in eastern India) is spoken / written by around 55 million natives and is now majorly used across online platforms. This paper builds Odia language models for text classification using LSTM and custom Transformer architecture. It then outlines a comparative analysis of these models with Keras embedding and FastText embedding used as part of the experiments.

Keywords – **NLP, Text Classification, LSTM, Transformers, Keras, FastText**

Contents

List of Tables

List of Figures

1 Introduction

Text classification is a significant part of NLP. According to IBM (Schneider, 2016) 80% of worlds data is unstructured, i.e. in form of text of web like reviews and blogs. With the world moving online people express their thoughts over comments, reviews, and blogs. Review/news polarity classification or sentiment analysis are a type of text classification and is a significant part of consumer behaviour. Sites like Yelp and Trip Advisor have millions of users and reviews of businesses. It is valuable for a business to understand the active feedback and valuable for customers to understand the product/services offered. (Luca, 2011) investigated this impact, using Yelp dataset and concluded that "a one-star increase in Yelp rating leads to a 5-9 percent increase in revenue."

Hence, text classification research has grown substantially in recent years. Useful insights can be generated from these texts and classification is a way to achieve this. Such analysis can help in understanding the opinion of the author or subject. This can help across product recommendations, ad placements, formulating business strategies etc.

Across the years, NLP has made significant progress for building strong language models with neural networks like RNN, LSTM, CNN like discussed in papers from (Ma and Hovy, 2016) for LSTM and (Kim, 2014) for CNN. However, with LSTM and RNN based seq2seq models, the time to train is relatively higher as the sentence is processed sequentially. In addition to that Novel attention-based transformer architecture has emerged as a viable alternative with state-of-the-art results in the initial paper by (Ashish Vaswani *et al.,* 2017). Bi-directional pretrained models like BERT (Devlin *et al.,* 2018) and TransformerXL (Dai *et al.,* 2019) are designed on top of this idea.

The paper outlines language models for Odia text classification, which can be evolved, modified, and trained for various NLP task helping business to do sentiment analysis on customer comments/reviews.

1.1 Motivation

Although significant progress has been made in English and multi-lingual NLP, very few efforts has been made to build a language model for Odia. Odia is the native language for about 55 million people (Language census India, 2015) and with access to internet and Unicode standards, much of the reviews and comments or news are available in this language over the internet. Few papers have been published on this language, which includes text mining using CS5.0 algorithm for a tourism application (Nanda *et al.,* 2015) and senti-wordnet for Odia (Mohanty *et al.*, 2018). Odia sentiment analysis was conducted on movie reviews with Naïve Bayes, Logistic regression and SVM models, with logistic regression being the superior model (Sahu *et al.,* 2016). However, no significant experiments have been conducted using LSTM or relatively new Transformer model on Odia text classification.

1.2 Aim & Objective

This study aimed to build language model, one with LSTM and another with custom Transformer architecture to compare the performance and trade-offs. It dealt with Odia text classification for news headlines collected from news website. With a limited number of works on Odia language-based NLP, this study has the potential to help NLP software in supporting local business. The architecture build as part of this study can be used to train with sentiment polarity, NER and many other applications.

1.3 Research Question

- **How well does LSTM & Transformer models perform on Odia language with Keras embedding?**

  The paper explores results on LSTM and Transformer models without using any specific word embeddings like word2Vec or FastText and using Keras text to sequence with default embedding.

- **Do models perform significantly better on Odia language with FastText embeddings?**

  The paper also explores results on LSTM and Transformer models with FastText word embedding pre trained on Odia wiki corpus.

- **Do the models provide a substantial result on Odia text classification?**

  Statistical testing to evaluate substantial difference between the models based on text classification accuracy.

1.4 Hypothesis

Null hypothesis – All models, LSTM including its variations and Transformer based behave the same for Odia text classification.

1.5 Scope

- The scope of this master thesis is to build text classification language model for Odia.
- Improve area of NLP for Odia.
- Build a foundation in deep learning language models for Odia.
- Experiment with various LSTM architecture and Transformer architecture.
- Evaluate models based on Odia news classification.

The rest of the paper is structured as follows. Section 2 provides a concise literature review of previous work aligning it with rationale behind this work. Section 3 describes the working knowledge and methodology that was used as part of this paper. Section 4 describes the project architecture and dataset details. Section 5 describes the experimental set up and results. Section 6 outlines the conclusion and points for future work.

2 Related Work

2.1 LSTM

Text classification problems have been a great area of research. Starting from normal machine learning models to deep learning models and recently Transformers. Strong language models with neural networks like RNN, LSTM, CNN are discussed in papers from (Ma and Hovy, 2016) for LSTM and (Kim, 2014) for CNN. LSTM has shown promising result (Mohanty et al., 2018) for sequential processes since long time (Hochreiter and Schmidhuber, 1997) overcoming the vanishing gradient problem with regulating the cell state. In addition to that, word embeddings have played a crucial role in boosting the creation of better models. Semantic feature extraction with word embedding like Word2Vec (Mikolov *et al.,* 2013) achieved results by mapping nearby words in vector space with a bit of shortcoming of learning new

representations as its pre-defined dictionary based. Character-level word embeddings was proposed with FastText (Mikolov *et al.,* 2018) in a state-of-the-art word embedding (Bojanowski et al., 2016) have been effective in training models, which is what the paper tried to design in one of the model experiments embedding.

Research on Odia language model have been limited to senti-wordnet creation (Mohanty *et al.*, 2018) and models with Logistic regression, SVM & Naïve Bayes (Sahu *et al.,* 2016) and no deep learning models are created. Hence, this paper decided to investigate and explore further on the idea of building a deep learning model for Odia using LSTM.

2.2 Transformer

Although LSTM models are well established in NLP research, it inherently depends on recurrence and sequential computation with previous works concluding that LSTM NLP models use on average of 200 words in context (Khandelwal *et al.,* 2018) having room for improvement. With Attention mechanism sequence modelling was done without the distance dependencies in input and output sequence (Kim *et al.,* 2017).

Based on which, Transformer architecture was born, depending solely on attention mechanism, and enabling more parallelization for achieving language translations (Vaswani et al., 2017). The input embeddings were added with positional encoding, so that each input has a relative positional sense with respect to other inputs. The paper used multi-head attention layer with a feed forward network for the encoder and similar setup for the decoder. The multi-head attention layer is essentially scaled-dot product attentions running in parallel. It enabled the model to jointly attend information from different subspaces at different positions.

The success of Transformers was followed by state-of-the-art models like BERT. The architecture is based on self-attention with around 12 heads. It used a masked language model which randomly masks some tokens in input and predicting the masked word based on the context. This is essentially based on pre-training of the model on unlabelled data around 3.3B words from English Wikipedia and other datasets and fine-tuning it for downstream NLP tasks. They have extensively described pre-training of BERT for Masked Language Modelling and Next sentence prediction. And fine-tuning the model by adding a simple classification layer to the pre-trained model (Devlin et al., 2018). Apart from English, BERT was trained for specific language or multi-lingual BERT was introduced which is trained on Wikipedia corpus. This is a model trained on all Wikipedia pages of 104 different languages, however, it does not include Odia. Many researchers then took forward the task of pre-training BERT as a mono-lingual model. Models for French language, CamemBERT (Martin *et al.,* 2020), ParsBERT for Parsi (Farahani *et al.,* 2020), BERTJe for Dutch (de Vries *et al.,* 2019), AraBERT for Arabic (Antoun, Baly and Hajj, 2020) were developed and achieved state-of-the-art results in their respective language NLP tasks like – Sentiment Analysis, Named Entity Recognition (NER), Text Classification. However, it is noticed that the amount of data and computational resources required for pre-training is substantial. BERTJe was trained on 39GB of text data, around 6.6 billion words. Similary, CamemBERT used 138GB of raw text and AraBERT used 24GB of text, around 70 million sentences collected from different sources.

Similar other models with Transformer architecture have been researched like the Spanish irony detection model trained on 87million tweets and with a 300-dimension embedding layer using word2vec. The paper only used encoder part of the Transformer with multi-head attention and global average pooling for the classification task (González *et al.,* 2019).

Another multilingual sentiment classifier was built on basis of contextual embedding and self-attention (Biesialska *et al.,* 2020). English, Polish, and German language was used as part of this experiment. The word embedding inputs passed through self-attention layer which then passed through a bi-attention layer. This layer helps in determining the interdependencies of the representation. A single LSTM layer computes the pooled representation and then passed through fully connected network. Finally, a Softmax layer was used here to determine the classification.

Multi-Input Transformer Encoder is another language model built by (Zhu *et al.,* 2019) for thought mining on Chinese language. It used multiple inputs as in – word vector, POS and sentiment lexicon to get the word representation which was then used in multi-head self-attention (scaled dot product) and feed forward network. As part of decoder, fully connected networks are used which can classify sentiments with Relu function was used. They have achieved a F1 score of 84 with around 30K data.

With limited time and resources this paper rather than pre-training BERT, explores the possibility of building custom based architecture using multi-head attention.

3 Related Theory

3.1 LSTM

Long Short-Term Memory networks are a special form of RNN, capable of learning long-term dependencies (Hochreiter and Schmidhuber, 1997). The secret to the LSTMs is the cell state that is like a conveyor belt. It runs straight down the entire chain, with just a few small linear interactions. It is quite convenient for information to just keep moving along. The standard LSTM network consists of several memory blocks called cells.

In principle, the cell state can carry relevant information during the processing of the sequence. But even details from earlier time steps will make the way for later time steps to reduce the impact of short-term memory. When the cell state goes on its path, information is added or removed to the cell state through gates. The gates are specific neural networks that determine which information is allowed on the state of the cell.

There are two states that transfer to the next cell, the cell state, and the hidden state. Memory blocks are responsible for recalling items and manipulation of this memory is achieved by three main mechanisms, called gates.

Forget Gate

Takes two inputs one from the hidden state from the previous cell or the output of the previous cell and another is the input at that time step. Weights and bias are added with addition to sigmoid function. It is responsible for deciding which values to keep and which to discard.

Input Gate

Information is added in here to the cell state. As perceived from previous two inputs, tanh function is applied here, which outputs to -1 to +1.

Output Gate

The output gate determines what to do with the next secret state. The previous hidden state and the current input is passed to the sigmoid function. Newly modified cell state to the tanh function. Tanh output is multiplied with the sigmod output to decide which information the hidden state should carry. Output is a hidden state. The new cell state and the new secret state are then transferred to the next stage.

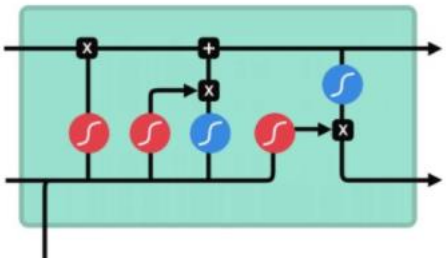


FIGURE 1: LSTM (SOURCE: TOWARDSDATASCIENCE)

3.2 Transformer

Transformer (Vaswani *et al*., 2017) utilize attention mechanism to check dependency between tokens, applying hidden representation to perform downstream tasks. Both Encoder and Decoder are made up of modules that can be stacked on top of each other multiple times. The modules consist mainly of Multi-Head Attention and Feed Forward layers. Another important part of the model is positional encoding which details the positions of the sequences that are fed to the model. As there are no recurrent networks that can remember how sequences are fed into a model, each word / part of the sequences is given a relative location, by this positional encoding. These positions are applied to the built-in representation (n-dimensional vector) of each word. The positional encoding (1) (2) vectors that were defined by (Vaswani *et al*., 2017) as follows. 'pos' is the position of the word in the sequence and 1 ≤ i ≤ d is the dimension in the positional encoding vector. Sinusoidal positional embeddings create embeddings using sin and cos functions. Using the equation shown below, the author hypothesized it allows learning for relative positions.

$$PE_{pos,2i} = \sin(\frac{pos}{1000^{2i/d}}) \qquad (1)$$

$$PE_{pos,2i+1} = \cos(\frac{pos}{1000^{2i/d}}) \qquad (2)$$

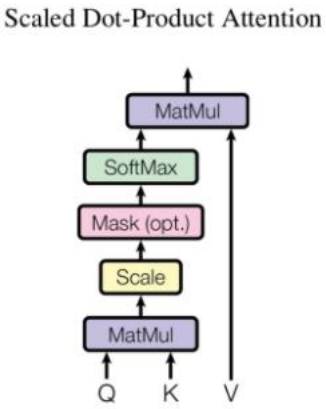


FIGURE 3 SCALED DOT PRODUCT

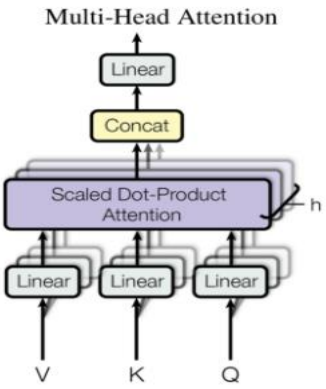


FIGURE 2 MULTI-HEAD ATTENTION (FROM 'ATTENTION IS ALL YOU NEED' BY VASWANI *ET AL.*)

(FROM 'ATTENTION IS ALL YOU NEED' BY VASWANI *ET AL.*)

$$Attention(Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad (3)$$

Attention (3) is described as mapping a query (Q) and a set of key (K) and value(V) pairs to an output, where all are vectors. Essentially, it is a scaled dot-product attention to compute hidden state of inputs. Scaled dot-product computes the dot products of the query with all keys, then apply a softmax function to obtain the weights on the values. For encoders and decoders, multi-head attention modules, V consists of the same word sequence as Q. Nevertheless, V is different from the sequence defined by Q for the attention module that considers the encoder and the decoder sequences.

The weights that is multiplied with V are determined by how much each word in the sequence (Q) is influenced by other words (K) in the sequence. The Softmax function helps the distribution to be in 0 and 1.

Figure 2 illustrates how this attention-mechanism can be paralleled to several mechanisms that can be used side by side. The attention mechanism is repeated multiple times with the linear projections of Q, K and V. This allows Transformer to learn from the different representations of Q, K and V. Such linear representations are rendered by the multiplication of Q, K and V by the weight of W matrixes that are taught during preparation.

3.3 Proposed model

The paper experiments with two basic models.

- LSTM
- Custom Transformer based.

The LSTM models are then built as either Single LSTM, Double LSTM or Bi-Directional LSTM modes.

3.3.1 LSTM

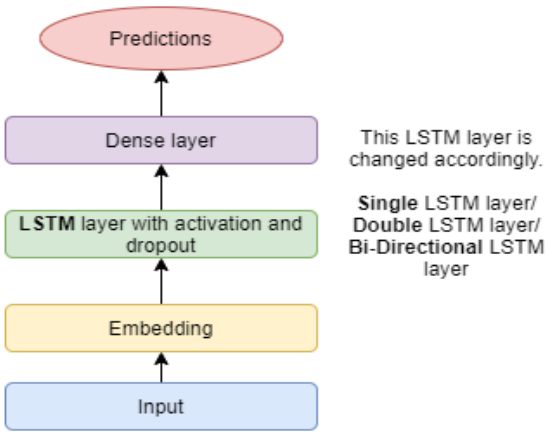


FIGURE 4 LSTM (SOURCE: AUTHOR)

The LSTM deep-learning model is built with Keras sequential modelling. The structure is as defined in figure 4 below. The input is passed through Keras embedding layer. This embedding

layer can be the in-built Keras embedding layer with multi-dimensional vectorization or FastText embeddings for the language. This is then passed through the LSTM layers.

3 types of models are aligned as part of this paper.

- Single LSTM
- Double Layer LSTM

  In-terms of stacking another layer of LSTM, the main reason is to make the model more complex. The LSTM phase at each point, in addition to the recurrent data now as the output of the LSTM layer before it, the new LSTM will construct a more complex representation of the current input.

- Bi-Directional LSTM

  Unidirectional LSTM only preserves information from the past as the only inputs that went through are from the past. Using bidirectional will run the inputs in two directions, a forward and backward pass. One from past to future and one from future to past, and what makes this method different from unidirectional is that in the LSTM that runs backwards, it retains information from the future and use the two hidden states combined to retain information from the past and the future at any time. Bi-LSTM tend to show very good results as they understand the context.

Output from the LSTM layer is passed through a Dense layer with 3 nodes and softmax activation which is essentially the text classifiers. In this layer each neuron receives input from all the previous neurons, thus densely connected. The layer has a weight matrix W, a bias vector b, and the activation function of softmax.

3.3.2 Transformer

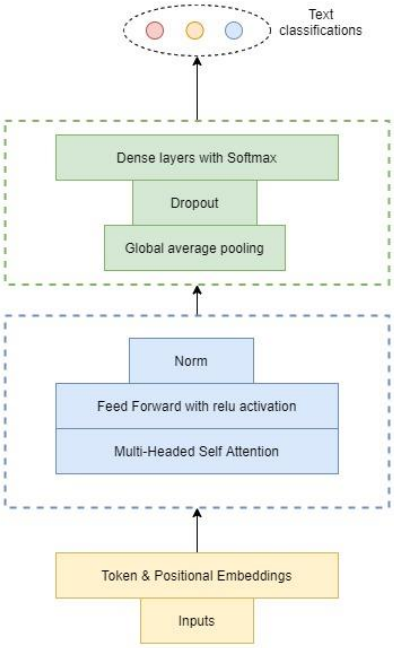


FIGURE 5 TRANSFORMER BASED CUSTOM MODEL (SOURCE: AUTHOR)

In case of the second model, it is designed as per figure 5. The inputs are passed through token embedding (either Keras or FastText). The input of the model is a headline

$$H = x_1, x_2, \ldots . x_t : x_i \in \{0 \ldots . V\}$$

T is the maximum length of the headline and V is the vocabulary size. This headline is sent to a d-dimensional fixed embedding layer, on which weights are initialized by the embedding layer. Positional information is also added with the sine and cosine functions described in (1) (2).

A Transformer block is designed as encoder which relies on multi-head scaled dot-product attention. The function is designed for 'N' number of heads. However, the paper uses two heads. The next layer includes a hidden feedforward layer which is essentially in this case are 2 dense layers with a feed forward dimension of '32' and 'relu' activation, the layer normalization (Ba, Kiros and Hinton, 2016).

The output from the above is fed into a global average pooling layer which computes the mean value of each feature map and sends it forward. This is then taken forward with a dropout layer which is adjusted as part of hyperparameter tuning and ends up in a dense layer of 3 nodes and softmax activation just like our LSTM models output. This block of the model can be termed as the decoder part, which makes it a custom Transformer built with multi-head self-attention. Essentially, the token embedding is suitable for both Keras and FastText pre-trained embedding to be added.

3.4 Methodology

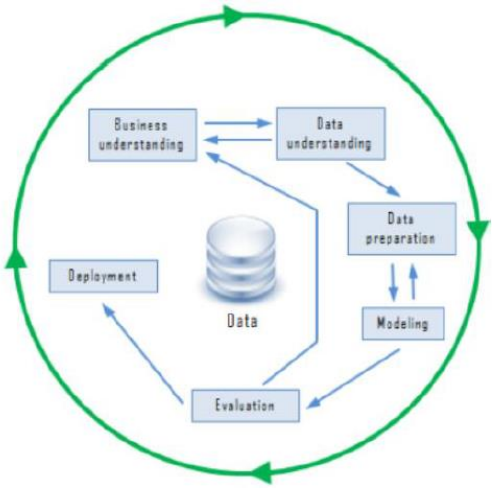


FIGURE 6 CRISP DM (WIRTH AND HIPP, 2000)

The crisp-DM (Wirth and Hipp, 2000) is clear and pragmatic — and helpful to explain the theoretical mechanism behind the work of data science projects. That is also the mechanism that paper uses for the project.

Business Understanding

In order to formulate a particular business issue that can be solved by a predictive model , it is important to consider the data sources and the context of the data itself, and to formulate

a collection of hypotheses regarding the variety of market "questions" that can be answered predictively. That is an iterative process — the original review of the data contributes to the actual interpretation of the market problem and the theory of how this problem can be overcome.

This paper outlines the use cases of text classification on Odia language. From a business standpoint the models built are validated against each other to find the trade-offs. These models will ultimately be helpful as part of various text classification outcomes like sentiment analysis on product and services.

Data understanding

The paper discusses more on data collection in experimental set up section. The data used for the train and test is collected from headlines of news website across three segments- State, Sports and Business news. Text classification deep-learning model on Odia language might help in opening possibility for other NLP tasks.

Data Preparation

This step will involve data cleaning and merging as per the problem definition and requirements. In our case, that will have the process of data preparation which involves eliminating unwanted characters, tokenization, elimination of stop words, stemming etc. Giving the classifications unique numerical classes. All the steps that is needed to reach a better feature extraction will be taken care in this step. Details of data preparation is described in 5.2 section.

Modelling

The process typically begins by choosing one or more models from that are the most "promising" based on the hypotheses generated earlier. Systematic training and tuning leads to a gradual improvement in the performance of the models. It is an iterative operation. Based on the success of the simulations, the team would need to go through and review the conclusions. The paper, as discussed in earlier section used LSTM and custom Transformers as deep learning models. Which are tuned based on the nodes, dropout, activation function

Evaluation

Evaluation of the projects and evaluation of whether it is sufficient to achieve the market objectives. For example, a 70 percent precision forecast model may be "good enough" for some applications but not for others. If the precision is found to be inadequate the team goes back to the data preparation / modelling process for things that can be tuned. Evaluation metrics for the paper is defined in section 5.3.

Deployment

Finding the right model that satisfies the business need of achieving the accuracy will lead to deployment of it. It can be expected as an optimal model that can be used for business classification of the text classification. The paper concludes in statistical testing of the outputs for the models. And model deployment is not in scope of the same.

4 Experimental Setup

The experiment is divided mainly into two broad categories.

- A LSTM based set up with
    - Default Keras embedding
    - FastText Odia word embedding

The LSTM based model is then further designed with Single layer, Double layer and Bi-directional LSTM.

- Transformer based set up with
    - Default Keras embedding
    - FastText Odia word embedding

The proposed task is to train and test the models and evaluate in terms of accuracy / F1 score. Google online platform Colab is used as part of setting up and running the models.

4.1 Dataset

The dataset is collected from Odia news website (sambad.in) from three sections – Business, State and Sports news. The dataset collected is of size **20,740** headlines and the distribution of headlines across three sections is as below.

TABLE 1 DATA COUNT

Category	Count
State	6,981
Sports	6,792
Business	6,967

The dataset is primarily the headline and the label (section) of news. The same is sampled as in figure 7 below.

headings	label
ନୂଆପଡ଼ାର ମାଓପ୍ରବଣ ୮ଟି ପଞ୍ଚାୟତରେ ଥିବା ଅସ୍ଥାୟୀ ସ...	state
ରାଜ୍ୟର ସୀମା ଜଗିବା ଦୁଃସାଧ, ପଞ୍ଚାୟତମାନେ ଚାହିଁ...	state
ଶଙ୍କରାଚାର୍ଯ୍ୟଙ୍କୁ ଭେଟି ସ୍ନାନଯାତ୍ରାକୁ ନିମନ୍ତ୍ର...	state
ରାଜ୍ୟର ଆର୍ଥିକ ସ୍ଥିତି ଉପରେ କରୋନା ମାଡ଼ : କେନ୍ଦ୍...	state
କଟକ ବଡ଼ ଡାକ୍ତରଖାନା ହେବ ବିଶ୍ୱସ୍ତରୀୟ : କ୍ୱାର୍ଟର୍...	state

FIGURE 7 DATASET

4.2 Data Preparation

As part of data preparation, the headlines are cleaned first. This includes removal of stop words. Typically, words with length less than 3 are removed, as stop words in Odia like 'ଆଉ', 'ଓ' which does not add much to the context of headlines. Removing special characters, number, html tags and garbage characters was also considered essential in data cleaning phase. Since the data was collected sequentially one section at a time, it also needed shuffling. As part of the process, one hot encoding is also applied to the categorical labels, so that categorical cross entropy can be applied as part of the deep learning model.

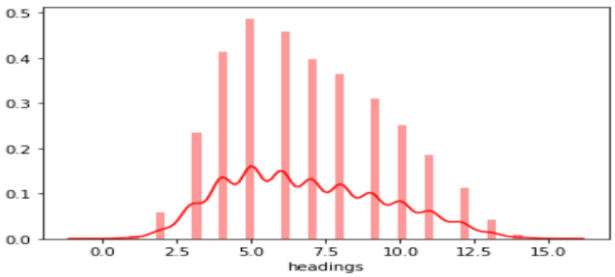


FIGURE 8 HEADLINES LENGTH DISTRIBUTION (SOURCE: AUTHOR)

The sentence length is first summarised and based on the column diagram (Figure 8) we observed the distribution of it. The headline length is concentrated around 5-6 words. Experiments are conducted on taking variant input lengths, padded to a specific number as an input to the models. It is found that, when we used top-k tokens for sentences i.e. around 10-12 and padded the shorter ones, the model performed better. Anything beyond 12 and below 5 the model performance decreased significantly because of information redundancy or low information. It is hence decided to keep the input sequence length at 12 while feeding it to the model with padding the sequences lower than 12.

TABLE 2 DATA COUNT IN SETS

Data	Count
Train	14,588
Test	3,647
Validation Set1	500
Validation Set2	500
Validation Set3	500
Validation Set4	500
Validation Set5	500

As part of data preparation, Keras Tokenizer is used for tokenization, text to sequence and padding sequences. 5 separate datasets are extracted out of the primary set for model testing. These 5 datasets are the text sequences which the model never used as part of train and test. These were used as separate set for validations results on which “model.evaluate” is run. With this extraction the table below showcases the number of data as part of the process.

The train-test split after extraction of validation set is done on 80-20 basis.

4.3 Model training and Evaluation Metrics

After data processing and making the input ready, various models that are discussed above are fed through. All the models are trained and validated with the Train-test and validation sets on Google Colab GPU.

With Keras Embedding

This is the embedding which is part of the Keras embedding layer. The input sequence on conversion of text to sequence is passed to this layer directly. On this paper, the embedding layer added to all the models mentioned below is of dimension 100. So essentially, the embedding layer will be creating an output of having 100 dimensions based on the input document that contains one vector for each word in the sequence specified. Below are the models that are trained with it.

- Single LSTM
- Double Layer LSTM

- Bi-Directional LSTM
- Transformer

With FastText Embedding

This is the word vectors from FastText that was generated from common crawl and wikipedia corpus, which are trained using Continuous Bag of words method with position weights and a dimension of 300. This Odia word embedding has 324,930 vectors. This is used with the Keras embedding as weights in an embedding matrix. The embedding matrix is created by using the word_index that was created with help of Keras Tokenizer. Below are the models that are trained with it.

- Single LSTM
- Double Layer LSTM
- Bi-Directional LSTM
- Transformer

4.3.1.1 Model Evaluation

The above-mentioned models are primarily evaluated based on F1 score, that considers precision and recall ratio. Our dataset is slightly imbalanced in terms of sports headlines, that makes F1 a better evaluation critic rather than accuracy as it avoids false high results.

$$P = \frac{TP}{TP+FP} \qquad (3)$$

$$R = \frac{TP}{TP+FN} \qquad (4)$$

$$F1 = \frac{2* P* R}{P+R} \qquad (5)$$

F1 (5) is the harmonic mean of precision and recall, where FN is the false negative, TP is true positive, and FP is the false positives in the above equations.

4.3.2 Hyperparameter Optimization

4.3.2.1 Number of nodes

The K number of nodes are essentially selected based on a trial and error run for the models. A basic formula (6) mentioned below is used as part of number of nodes selection for the models.

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o)} \quad (6)$$

Ni = input neurons number.

No = output neuron numbers.

Ns = Training dataset samples.

α = Scaling factor that is ideally between 2 and 10.

8 different numbers can be calculated based on this and find an optimal model based on the resulting validation loss. Based on the experiments performed upon modifying the number of nodes below are the number of nodes that is found to give an optimal result for the dataset used. Table below outlines the same.

TABLE 3 MODEL HYPERPARAMETER - NODES

Single LSTM	Double Layer LSTM	Bi-Directional LSTM	Transformer
64 Nodes	64 and 32 Nodes	32 Nodes	2 Attention heads & 32 node feedforward layer

These numbers are kept relatively close for Keras and FastText embedding models to have a better comparative view.

4.3.2.2 Dropout

This is a simple yet powerful regularization technique for deep learning models (Srivastava *et al.*, 2014). Helps in avoiding overfitting by randomly ignoring neurons during the training phase where contribution to downstream neurons are removed and weights updates are ignored in back pass.

When the neurons are arbitrarily deleted from the network during preparation, the other neurons will have to step in to perform the interpretation needed to make predictions for the missing neurons. In turn building multiple separate internal representation. A dropout is varied from 20-50% in the experiments. A dropout too low has chances of the model to overfit, while a dropout too high can result in model under-learning. The table below outlines the dropout applied across the models.

TABLE 4 MODEL HYPERPARAMETER - DROPOUT

Single LSTM	Double Layer LSTM	Bi-Directional LSTM	Transformer
LSTM layer dropout 0.2	LSTM layer dropout 0.2	No LSTM layer dropout	Transformer block dropout 0.1 Model layer dropout 0.25

4.3.2.3 Optimizer

As part of the models, the Adam optimization algorithm (Kingma and Ba, 2014) is used. This is an extension to stochastic gradient descent. It combines the best practices of AdaGra and RMSProp to provide optimization. 'learning_rate' is adjusted based on the runs and was finally set to '1e-5'.

4.3.2.4 Activation function

The input to the neurons performs a linear transformation on it with weight and bias. On top of that activation functions are applied, output of which is passed to next layers of neurons. Our experiments started with using ReLU function for the hidden layers. However better results were achieved using softmax. Softmax returns the probability of a datapoint aligned to specific class. Dense layers in the model were configured with Softmax and LSTM layers were configured with 'tanh' activation. Hidden layer in Transformer block is 'ReLU' activated.

4.3.2.5 Epochs

This defines the number of times the learning model will work through the entire training dataset. In these experiments performed for the models, the max epoch was set to 100 or 200. However, with deep learning there is a chance of overfitting when the training is done for too many epochs. As part of the experiments, Early stopping method is used that allows the model to stop once the model performance halts improving. This was configured as part of Keras callbacks. Early stopping was configured to monitor 'val_loss' (Validation loss) with a

mode as min, seeking a minimum for validation loss. The table below showcases the early stop triggers for models.

TABLE 5 MODEL HYPERPARAMETER - EPOCHS

	With Keras Embeddings	With FastTextEmbedding
Single LSTM	43	100
Double Layer LSTM	53	200
Bi-Directional LSTM	45	138
Transformer	100	200

4.4 Results

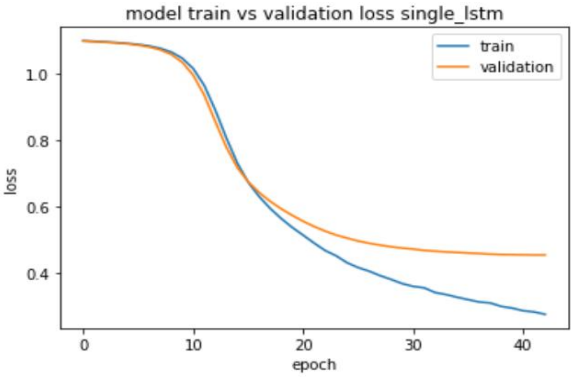
This section outlines results from the model. The models are built as discussed above in section 3 and hyperparameters are tuned as per section 4.3.2.

4.4.1 With Keras Embedding

Below are the observations from models with Keras embedding.

4.4.1.1 Single LSTM

The single LSTM model was configured with 64 nodes and an activation of tanh. Epochs of 100 and batch of 100 was set for the model train. The model ran for 43 epochs before early stop with an average F1 score of 82.1 and execution time of 115secs. It is observed from train/validation loss and train/validation accuracy graph (Figure 10) around epoch number 30 the model starts to overfit. Figure (11) outlines the F1 scores on test set and validation sets (VS).



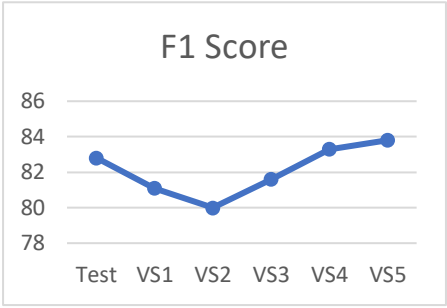
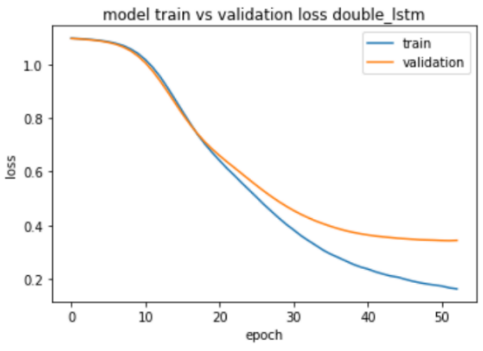


FIGURE 9 F1 SCORE SINGLE LSTM

FIGURE 10 TRAIN / VALIDATION LOSS – SINGLE LSTM

4.4.1.2 Double LSTM

The double LSTM model was configured with 32 nodes and an activation of tanh for the first layer and 32 nodes for the second layer. Epochs of 100 and batch of 100 was set for the model train. The model ran for 53 epochs before early stop with an average F1 score of 87.3 and execution time of 157 secs. It is observed from train/validation loss and train/validation accuracy graph (Figure 11) around epoch number 40 the model starts to overfit. Figure 12 outlines the F1 scores on test-set and validation sets (VS). It is observed that double layer LSTM gave substantially better results than single LSTM.



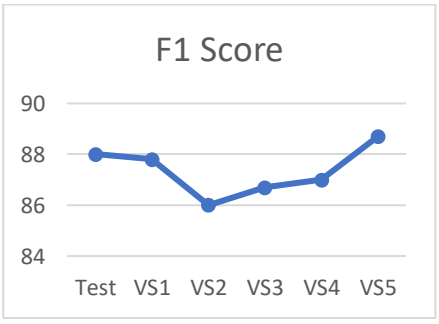


FIGURE 11 TRAIN / VALIDATION LOSS - DOUBLE LSTM

FIGURE 12 F1 SCORE DOUBLE LSTM

4.4.1.3 Bi-Directional LSTM

Bi-Directional LSTM model was configured with 32 nodes. Epochs of 100 and batch of 100 was set for the model train. The model ran for 73 epochs before early stop with an average F1 score of 86 and execution time of 200 secs. It is observed from train/validation loss and train/validation accuracy graph (Figure 14) around epoch number 40 the model starts to overfit. Figure (13) outlines the F1 scores on test-set and validation sets (VS). It is observed that Bi-directional LSTM has a better score than double LSTM.

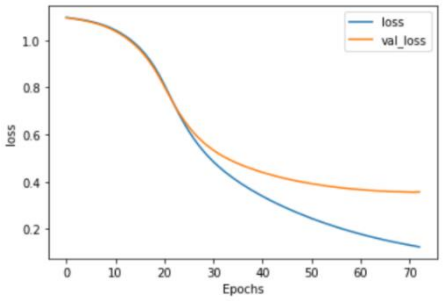


FIGURE 13 TRAIN / VALIDATION LOSS - BIDIRECTIONAL LSTM

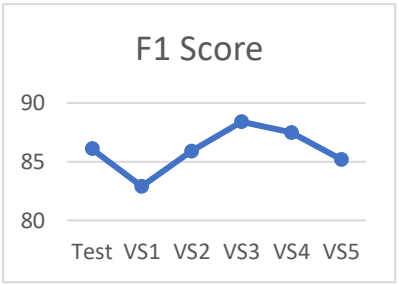


FIGURE 14 F1 SCORES BIDIRECTIONAL LSTM

4.4.1.4 Transformer

The model is configured with 2 attention heads and 100 epochs which runs for around 300secs with an F1 average of 86.6. It is observed from train/validation loss and train/validation accuracy graph (Figure 16) around epoch number 40 the model starts to overfit. Figure (15) outlines the F1 scores on test-set and validation sets (VS). It is observed that this model is like Bi-directional LSTM in terms of F1 score.

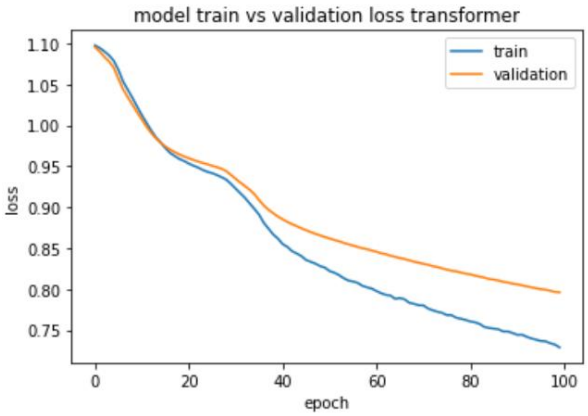


FIGURE 16 TRAIN / VALIDATION LOSS TRANSFORMER

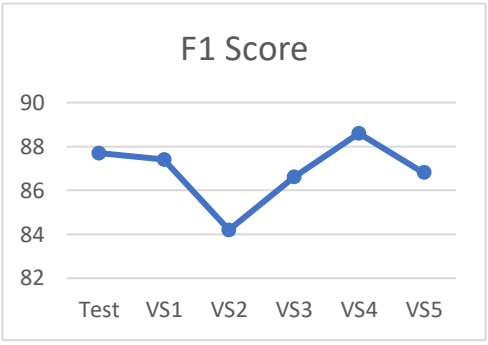


FIGURE 15 F1 SCORES TRANSFORMERS

4.4.2 With FastText Embedding

Below are the outline observations from models with FastText embedding. All the models embedding layer dimension was set to 300.

4.4.2.1 Single LSTM

The single LSTM model was configured with 64 nodes and an activation of tanh. Epochs of 100 and batch of 100 was set for the model train. The model ran for 100 with an average F1 score of 80.3 and execution time of 92 secs. It is observed from train/validation loss graph (Figure 18) the model does not overfit. There is minimal gap between both the lines training and validation loss lines. Figure (17) outlines the F1 scores on test set and validation sets (VS).

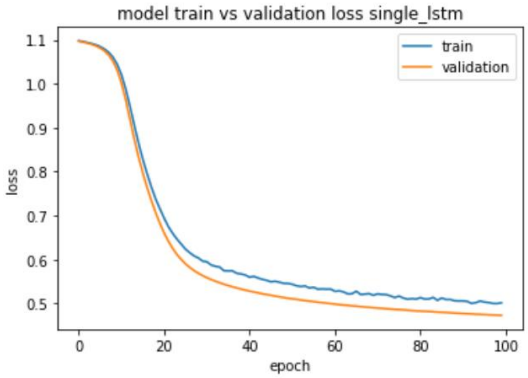


FIGURE 18 TRAIN / VALIDATION LOSS SINGLE LSTM - FT

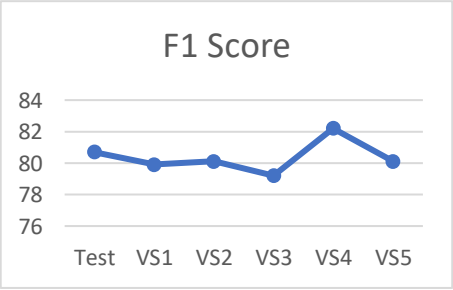


FIGURE 17 F1 SCORES SINGLE LSTM - FT

4.4.2.2 Double LSTM

The double LSTM model was configured with 32 nodes and an activation of tanh for the first layer and 32 nodes for the second layer. Epochs of 200 and batch of 100 was set for the model train. The model ran for 100 with an average F1 score of 81.5 and execution time of 245 secs. It is observed from train/validation loss graph (Figure 19) the model does not overfit. There is minimal gap between both the lines training and validation loss lines. Figure (20) outlines the F1 scores on test set and validation sets (VS).

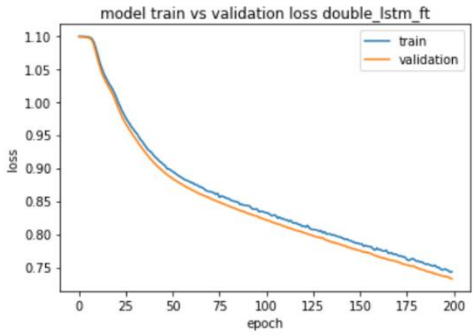


FIGURE 19 TRAIN / VALIDATION LOSS DOUBLE LSTM -FT

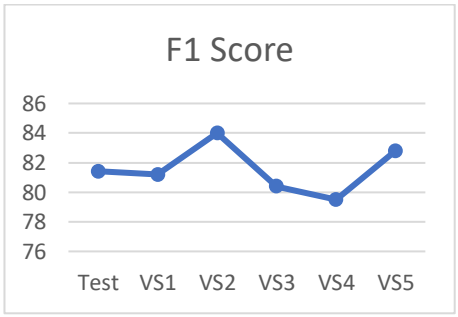


FIGURE 20 F1 SCORES DOUBLE LSTM -FT

4.4.2.3 Bi-Directional LSTM

Bi-Directional LSTM model was configured with 32 nodes. Epochs of 200 and batch of 100 was set for the model train. The model ran for 100 with an average F1 score of 81.5 and execution time of 163 secs. It is observed from train/validation loss graph (Figure 22) the model does not overfit. There is minimal gap between both the lines training and validation loss lines. Figure 21 outlines the F1 scores on test set and validation sets (VS).

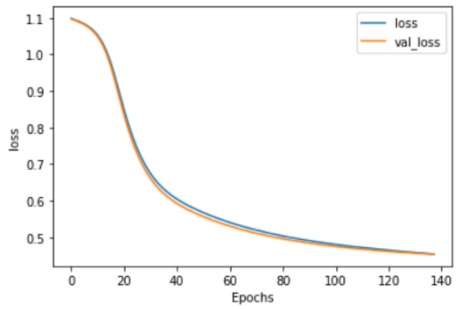


FIGURE 22 TRIAN / VALIDATION LOSS BI-DIRECTIONAL LSTM -FT

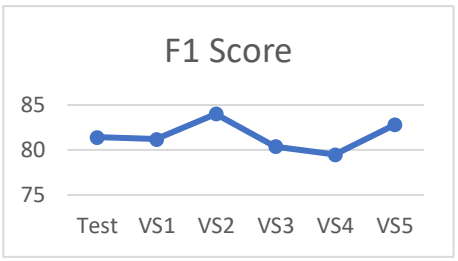


FIGURE 21 F1 SCORES BIDIRECTIONAL LSTM -FT

4.4.2.4 Transformer

The model is configured with 2 attention heads and 200 epochs which runs for around 300 secs with an F1 average of 82.3. It is observed from train/validation loss graph (Figure 24) the model does not overfit. There is minimal gap between both the lines training and validation loss lines. Figure 23 outlines the F1 scores on test set and validation sets (VS).

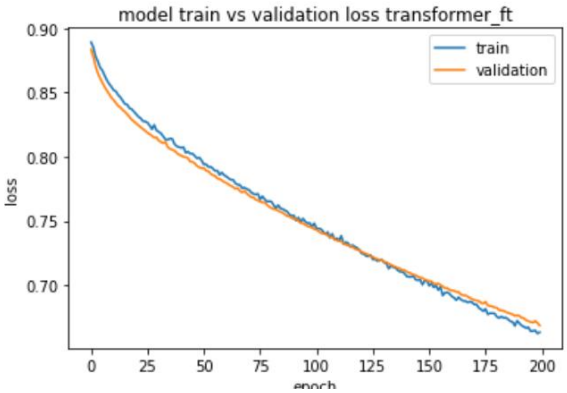


FIGURE 24 TRAIN/VALIDATION LOSS TRANSFORMER -FT

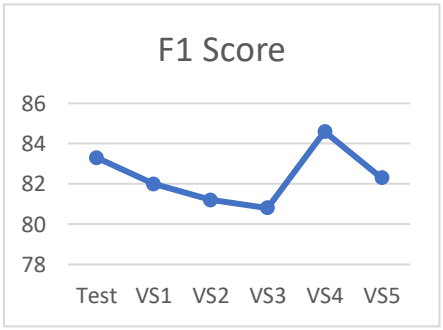


FIGURE 23 F1 SCORES FOR TRANSFORMERS -FT

It is observed that, the models with FastText word embedding had a better training to validation loss ratio than keras embedding. The graph plots are with minimum distance and better aligned. Although the F1 scores are like Keras embedding, that might be a result of pre-trained fasttext vector. The next section outlines the statistical testing on the similarities of the models based on observed F1 scores.

4.5 Statistical Test

To evaluate the models based on F1 scores and to test null hypothesis One-way ANNOVA test was conducted. Figure below represents the F1 scores across the models with Keras embedding.

lstm_single_f1	lstm_double_f1	lstm_bi_f1	transformer_f1
0.8285	0.8801	0.8840	0.87743
0.8118	0.8787	0.8611	0.87425
0.8004	0.8606	0.8293	0.84231
0.8163	0.8676	0.8590	0.86626
0.8336	0.8709	0.8844	0.88622
0.8389	0.8873	0.8756	0.86826

FIGURE 25 DATAFRAME OF F1 SCORES

Based on the test conducted, the p-value is calculated to be **0.0002761**. As it much lower than 0.05 the null hypothesis can be rejected. That is, the models have significant difference.

From ANNOVA analysis it is concluded that the models are significantly different, however which two pairs are different is not clear. Hence Tukey HSD test is conducted for the pairwise comparison.

Table 6 showcases the Tukey HSD results.

TABLE 6 TUKEY TEST

A	B	mean(A)	mean(B)	diff	se	T	p-tukey	hedges
lstm_bi_f1	lstm_double_f1	0.865567	0.8742	-0.008633	0.008962	-0.963279	0.742635	-0.513369
lstm_bi_f1	lstm_single_f1	0.865567	0.821583	0.043983	0.008962	4.907518	0.001	2.615406
lstm_bi_f1	transformer_f1	0.865567	0.869122	-0.003555	0.008962	-0.396655	0.9	-0.211393
lstm_double_f1	lstm_single_f1	0.8742	0.821583	0.052617	0.008962	5.870797	0.001	3.128775
lstm_double_f1	transformer_f1	0.8742	0.869122	0.005078	0.008962	0.566624	0.9	0.301976

lstm_single_f1	transformer_f1	0.821583	0.869122	-0.047538	0.008962	-5.304173	0.001	-2.826799

The p-tukey value observations outline that Single LSTM model is significantly different that Double, Bi-Directional LSTM and Custom Transformer model.

Similar test is conducted for models with FastText embeddings. Based on ANNOVA, p-Value for FastText embedding models is 0.0442. As it is below 0.05 the null hypothesis of all the models on FastText embedding can be rejected. The models have significant difference in F1 score. Upon Tukey-HSD test, it is also observed that Double layer LSTM and Transformer have a significant difference.

There is also notable difference between models of Keras embedding and FastText embedding with variety of tests. The experiments validated with F1 scores for transformer Keras embedding and Transformer FastText embedding resulting in p-value of 0.00027 showcasing difference. Similar test has been conducted for LSTM models as well, all concluding to significant difference between the models of keras embedding and FastText embedding.

Hence, the null hypothesis – as all the models are same for Odia text classification can be rejected. The above statistical tests conclude there is significant difference between the model – Single LSTM and Transformer in case of Keras embedding, Double LSTM and Transformer in case of FastText embedding and similar models across different embeddings.

5 Conclusions

With Odia language spoken and written by substantial number of people and things moving online, the paper outlined few necessities of language model for Odia. This paper discussed deep learning models based on LSTM and Transformer architecture, which were configured and tested with two types of embeddings (Keras and FastText). All the 8 models discussed produced F1 score in a range of 80-85.

5.1 Revisiting Research Questions

- **How well does LSTM & Transformer models perform on Odia language with Keras embedding?**

  With Keras embedding it is observed that the models were able to produce an average F1 score in the range of 82-86.

- **Do models perform significantly better on Odia language with FastText embeddings?**

  Models do not out-perform the F1 scores of Keras embedding. However, training / validation loss graph for models with FastText embedding outlines much better validation loss to train loss ratio with minimal gap and much cleaner graph (discussed in section 4.4.2). With FastText model trained on a better corpus the models should perform much better.

- **Do the models provide a substantial result on Odia text classification?**

  Both model sets do provide results of F1 score in range of 80-85. The table below summarizes the experiments with average F1 scores across the models.

TABLE 7 RESULTS

Keras	F1
Single LSTM	82.1
Double LSTM	87.3
Bi-Directional LSTM	86
Transformer	86.6

FastText	F1
Single LSTM	86.8
Double LSTM	80.3
Bi-Directional LSTM	81.5
Transformer	82.3

5.2 Limitations

- These are the base models, which has room for additional improvement.
- Although, the models are evaluated based on validation sets which are unseen, there is a room for addition of K-fold cross-validation to the models.
- Although FastText word embedding has been used in embeddings, there is a scope to train the FastText model with better corpus.
- The models are trained on around **14K** news headlines, there is a scope for training the models with larger datasets.
- Pre-training of language model such as BERT is not done due to dataset and time limitations.

5.3 Future Work

The future work in language model for Odia can involve training these models with more data. These models do have a scope of extension with additional layers and more experiments. With limitations of data or compute in current experiments, there is a scope of training these models with more data. These models can also be extended with custom embeddings by training FastText on better corpus as well which will in-turn give better results.

It is also recommended to pre-train BERT on large Odia corpus and build models like French language model CamemBERT (Martin et al., 2020) or ParsBERT for Parsi (Farahani et al., 2020) which can achieve state-of-the-art results for NLP in Odia.

6 References

1. Ashish Vaswani, Noam Shazeer, Parmar, N., Jakob Uszkoreit, Llion Jones, Gomez, A.N., Łukasz Kaiser and Illia Polosukhin (2017). Attention is All you Need. *Nips.cc*, [online] pp.5998–6008. Available at: http://papers.nips.cc/paper/7181-attention-is-all-you-need [Accessed 19 Jun. 2020].
2. Ba, J.L., Kiros, J.R. and Hinton, G.E. (2016). Layer Normalization. *arXiv:1607.06450 [cs, stat]*. [online] Available at: https://arxiv.org/abs/1607.06450 [Accessed 19 Jul. 2020].
3. Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q.V. and Salakhutdinov, R. (2019). *Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context*. [online] arXiv.org. Available at: https://arxiv.org/abs/1901.02860 [Accessed 19 Jun. 2020].
4. Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. [online] arXiv.org. Available at: https://arxiv.org/abs/1810.04805 [Accessed 19 Jun. 2020].
5. Driscoll, W.C. (1996). Robustness of the ANOVA and Tukey-Kramer statistical tests. *Computers & Industrial Engineering*, 31(1–2), pp.265–268.
6. Farahani, M., Gharachorloo, M., Farahani, M. and Manthouri, M. (2020). ParsBERT: Transformer-based Model for Persian Language Understanding. *arXiv:2005.12515 [cs]*. [online] Available at: https://arxiv.org/abs/2005.12515 [Accessed 11 Jul. 2020].
7. Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), pp.1735–1780 [Accessed 19 Jun. 2020].
8. Kim, Y. (2014). *Convolutional Neural Networks for Sentence Classification*. [online] arXiv.org. Available at: https://arxiv.org/abs/1408.5882 [Accessed 17 Jun. 2020].
9. Kingma, D.P. and Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. [online] arXiv.org. Available at: https://arxiv.org/abs/1412.6980 [Accessed 12 Jul. 2020].
10. Letarte, G., Paradis, F., Giguère, P. and Laviolette, F. (2018). *Importance of Self-Attention for Sentiment Analysis*. [online] pp.267–275. Available at: https://www.aclweb.org/anthology/W18-5429.pdf [Accessed 17 Jun. 2020].
11. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L. and Stoyanov, V. (2019). *RoBERTa: A Robustly Optimized BERT Pretraining*

Approach. [online] arXiv.org. Available at: https://arxiv.org/abs/1907.11692 [Accessed 17 Jul. 2020].

12. Ma, X. and Hovy, E. (2016). End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. *arXiv:1603.01354 [cs, stat]*. [online] Available at: https://arxiv.org/abs/1603.01354.
13. Martin, L., Muller, B., Suárez, P.J.O., Dupont, Y., Romary, L., de la Clergerie, É.V., Seddah, D. and Sagot, B. (2020). CamemBERT: a Tasty French Language Model. *arXiv:1911.03894 [cs]*. [online] Available at: https://arxiv.org/abs/1911.03894 [Accessed 11 Jul. 2020].
14. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S. and Dean, J. (2013). *Distributed Representations of Words and Phrases and their Compositionality*. [online] Neural Information Processing Systems. Available at: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and.
15. Mohanty, G., Kannan, A. and Mamidi, R. (2017). *Building a SentiWordNet For Odia*. [online] Association for Computational Linguistics, pp.143–148. Available at: https://www.aclweb.org/anthology/W17-5219.pdf [Accessed 23 Jun. 2020].
16. Nanda, S., Mishra, S. and Mohanty, S. (2011). *Oriya Language Text Mining Using C5.0 Algorithm*. [online] Available at: https://pdfs.semanticscholar.org/14c4/85822bf8bf44bed50c8d76a48e6cc6c13d62.pdf [Accessed 23 Jun. 2020].
17. Sahu, S.K., Behera, P., Mohapatra, D.P. and Balabantaray, R.C. (2016). Sentiment analysis for Odia language using supervised classifier: an information retrieval in Indian language initiative. *CSI Transactions on ICT*, 4(2–4), pp.111–115.
18. Schneider, C. (2016). *The biggest data challenges that you might not even know you have*. [online] Watson Blog. Available at: https://www.ibm.com/blogs/watson/2016/05/biggest-data-challenges-might-not-even-know/ [Accessed 16 Jun. 2020].
19. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, [online] 15(56), pp.1929–1958. Available at: https://jmlr.org/papers/v15/srivastava14a.html [Accessed 20 Jul. 2020].
20. de Vries, W., van Cranenburgh, A., Bisazza, A., Caselli, T., van Noord, G. and Nissim,

M. (2019). BERTje: A Dutch BERT Model. *arXiv:1912.09582 [cs]*. [online] Available at: https://arxiv.org/abs/1912.09582 [Accessed 11 Jul. 2020].

21. Wirth, R. and Hipp, J. (2000). *CRISP-DM: Towards a Standard Process Model for Data Mining*. [online] Available at: http://www.cs.unibo.it/~danilo.montesi/CBD/Beatriz/10.1.1.198.5133.pdf.
22. Zhou, Q., Zhang, Z. and Wu, H. (2018). NLP at IEST 2018: BiLSTM-Attention and LSTM-Attention via Soft Voting in Emotion Classification. [online] Available at: https://aclweb.org/anthology/W18-6226 [Accessed 3 Jul. 2020].