NATURAL LANGUAGE PROCESSING

CSE4022

PROJECT REPORT
WINTER SEMESTER 2021-22

PROJECT TITLE-

Using Transformers and Recurrent Neural Network (Bi- LSTM and Bi-GRU) to Identify Clickbaits.

Under guidance of
Prof. Rajeshkannan R

TEAM MEMBERS- (Group 13)

1. Soumyaraj Roy , 19BCE0200
2. Kaustubh Dwivedi, 19BCE0249
3. Rebello Nishma Avalon, 19BCE0253
4. Debangsu Sarkar, 19BCE0902

1. ABSTRACT-

Clickbait is a widespread problem that troubles online readers and misleads the readers to an irrelevant site. Currently, detection of clickbait on tweets remains a challenging task.

Clickbait is a rampant problem haunting online readers. Nowadays, clickbait detection in tweets remains an elusive challenge. In this paper, we propose Clickbait Detector with Bidirectional Encoder Representations from Transformers (Click-BERT), which could effectively identify clickbaits utilizing recent advances in pre-training methods (BERT and Longformer) and recurrent neural networks (Bi LSTM and Bi-GRU) with a parallel model structure. Our model supports end-to-end training without involving any hand-crafted features and achieved state-of-the-art results on the Webis 17 dataset.

In this project, we propose to build a Clickbait Detector using Bidirectional Encoder Representations from Transformers (BERT), which can effectively identify click-baits using the latest developments in advanced training methods like BERT and Longformer and using Recurrent Neural Networks (Bi. - LSTM and Bi-GRU) with a parallel model structure.

Our model will support end-to-end training without incorporating any manual features and achieve efficient results. We will approach this task as a regression problem in our two parallel baseline models for benchmarking with previous models. The model will take the post title and the linked content as input and will output a clickbait score in the range of [0, 1] with 0 indicating non-clickbait and 1 indicating clickbait.

By training on a large twitter posts corpus with annotations of their ‘click-baitness’ on a scale of [0, 1], we expect our model to be capable of capturing clickbait patterns in the headline and the content.

2. INTRODUCTION-

Clickbait refers to a certain kind of headline that attracts people to click but gives something uncorrelated in that link. The motivation behind clickbaiting is to boost site traffic (and therefore, advertisement revenue) by exploiting the curious nature of human readers. The clickbait technique works by dangling a hyperlink with enticing headlines to lure people into clicking; and then redirect them to the publishers' own websites which are uncorrelated to the headline.

The discrepancy between the headline and the destination content wastes online readers significant amount of time on contents of which they have no interest. To address this problem, we propose Click-BERT: (Clickbait Detector with Bidirectional Encoder Representations from Transformers), which could effectively identify clickbaits utilizing state-of-the-arrt pre-training methods and self-attentive network. We approach this task as a regression problem in our two parallel baseline models for benchmarking with previous models. The model takes the post title and the linked content as input and will output a clickbait score in the range of [0, 1] with 0 indicating non-clickbait and 1 indicating clickbait. By training on a large twitter posts corpus with annotations of their 'clickbaitness' on a scale of [0, 1], we expect our model to be capable of capturing clickbait patterns in the headline and the content.

Main challenges of the clickbait detection problems lies-

in how well our model capture the meaning and correlation of the input headline/content (which differs greatly in length) and how properly the followed analysis gets performed. And our main contributions include:

1. We applies advanced pre-trained models BERT and Longformer to extract sequence (headline/content) embedding to form a better understanding of the headline and the content.

2. We proposes a parallel model structure to integrate both prediction of whether the headline is luring people to click and prediction of whether the headline is related with the content into final judgement.

3. LITERATURE REVIEW

S. No.	Paper Title & Details	Method/Algorithm	Challenges	Observations
1.	Clickbait Detection in YouTube Videos. Authors- Ruchira Gothankar, Fabio Di Troia, Mark Stamp. Year- 2021	The authors performed clickbait detection experiments are based on a set of labeled videos. The problem is formulated as a binary classification problem where for each video a machine learning algorithm classifies it is clickbait or non-clickbait. The information from multiple sources (e.g., title, description, comments) are combined and fed to the classification model. The performance is evaluated and analyzed by multiple measures, specifically, precision, recall and the F-score. BERT, Word2Vec, and DistilBERT were used for word embeddings	They confirmed that the accuracy of the models could be increased by adding more features. For future work, more features have to be included and also DocToVec embeddings could be considered.	Multiple classification techniques were considered, including logistic regression, random forest, and MLP, and we employed Word2Vec, BERT, and DistilBERT as language models. The best accuracy was achieved using an MLP classifier based on BERT embeddings which is 94.5 %, but a the more lightweight DistilBERT performed almost same.
2.	exBAKE: Automatic Fake News Detection Model Based on Bidirectional Encoder Representations from Transformers (BERT). Authors- Heejung Jwa , Dongsuk Oh, Kinam Park, Jang Mook Kang and Heuiseok Lim.	In this paper, the authors focus on data-driven automatic fake news detection methods. First they apply the Bidirectional Encoder Representations from Transformers model (BERT) model to detect fake news by analyzing the relationship between the headline and the body text of news. To further improve performance, additional	They experiment with various cases of fake news detection tasks using the pre-trained BERT model proposed in this study. They only analyzed the relationship between the headline and the body text of an article. But, Further experimentation is needed to apply data from other fake news detection tasks to	They determine that the deep-contextualizing nature of BERT is best suited for this task and improves the 0.14 F-score over older state-of-the-art models

		news data are gathered and used to pre-train this model.	BERT model, which will use additional news data in the pre-training phase.	
3.	Clickbait Headline Detection in Indonesian News Sites using Multilingual Bidirectional Encoder Representations from Transformers (M-BERT). Authors-Muhammad Noor Fakhruzzaman , Sa'idah Zahrotul Jannah, Ratih Ardiati Ningrum, Indah Fahmiyah. Year 2021.	This study contributes to show that Multilingual BERT, a state-of-the-art model is able to classify Indonesian clickbait headlines. By using BERT, the whole model looks simplified, using only a BERT layer and a hidden standard dense layer, finally topped with a sigmoid activated neuron, the classifier worked remarkably well with an average accuracy of 92%.	A further study is needed to evaluate the model versatility. Moreover, training a Neural Network with M-BERT took a lot of computing resource.	If efficiency is the priority, XGBoost can perform moderately well (80% avg.). The additional evaluation shows average accuracy of 0.83, precision of 0.82, recall of 0.83, and f1-score of 0.83
4.	Stop clickbait: Detecting and preventing clickbaits in online news media. Chakraborty, A., Paranjape, B., Kakarla, S. and Ganguly, N. IEEE	1. Authors did a definite phonetic investigation on the 15, 000 features both in the misleading content and non-misleading content, utilizing the Stanford CoreNLP tool. They examined how semantic and syntactic subtleties which are explicit to misleading content sources like Sentence Structure, Stop words, Determiners, Word N Grams, POS Tags. At last they characterized utilizing Feature selections like Word patterns, clickbait content language, N Gram features. At last they implement the classifier through a Browser extension.	1. Manually identify the clickbait articles from Clickbait-y sites, and to avoid false negatives we need multiple opinions as an article is a clickbait or not is a subjective opinion - We need to take majority vote. 2. Need to manually compiled a list of most commonly used bait phrases. 3. One issue about earlier works is that they either work on a single domain, or the fixed ruleset does not capture the nuances employed across different websites	1. Conventional non-clickbait headlines contain much larger proportion of proper nouns. 2. Clickbait headlines contain more adverbs and determiners There's a lot of extreme positive or negative words in clickbait sites, called Hyperboles. 3. Informal Punctuations.

5.	Bert: Pre-training of deep bidirectional transformers for language understanding, J., Chang, M.W., Lee, K. and Toutanova, K., 2018	There are two steps in the BERT framework: pre-training and fine-tuning. During pre-training, the model is trained on unlabeled data over different pre-training tasks. For finetuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks. Each downstream task has separate fine-tuned models, even though they are initialized with the same pre-trained parameters	1. Deployment of BERT models in dynamic commercial environments often yields poor results. This is because commercial environments are usually dynamic, and contain continuous domain shifts (e.g. new themes, new vocabulary or new writing styles) between inference and training data, thus the challenge of dealing with dynamic cross-domain setups in which there is no labeled target-domain data, still remains. 2. BERT can be used only for answering questions from very short paragraphs and a lot of key issues need to be addressed. NLP as a general task is way too complex and has many more meanings and subtleties. BERT solves only a part of it but is certainly going to change entity Recognition models soon.	1. A distinctive feature of BERT is its unified architecture across different tasks. There is minimal difference between the pre-trained architecture and the final downstream architecture. 2. Recent empirical improvements due to transfer learning with language models have demonstrated that rich, unsupervised pre-training is an integral part of many language understanding systems. In particular, these results enable even low-resource tasks to benefit from deep unidirectional architectures. 3. To improve the training procedure, RoBERTa removes the Next Sentence Prediction (NSP) task from BERT's pre-training and introduces dynamic masking so that the masked token changes during the training epochs. It was also trained on an order of magnitude more data than BERT, for a longer amount of time.

6.	Detecting and Categorization of Click Baits, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) NTASU – 2020 (Volume 09 – Issue 03). Sainath Patil, Mayur Koul, Harikrishan Chauhan, Prachi Patil, 2021.	The authors propose a completely unique approach considering all information found during a social media post. We train a bidirectional Long Short Term Memory(LSTM) with an attention mechanism to learn the extent to which a word contributes to the posts clickbait score in a differential manner. Sequence Followed: Data Collection, Word Embedding, Developing the Deep Learning Models.	1. The aforementioned Attention mechanism wasn't implemented into the paper, leading to us thinking that it might be hard to do so. 2. However, work done specifically for Twitter had to be expanded since clickbait was available throughout the Internet, and not just social networks. 3. Again, the definition of what a clickbait is and what isn't is vague and is an issue that needs to be discussed before approaching the required problem.	1. The primary instance of detecting clickbait across social media can be traced , hand-crafting linguistic features, including a reference dictionary of clickbait phrases, over a data set of crowdsourced tweets. 2. The features need to be more nuanced to avoid flagging non-clickbait articles.
7.	Unified Medical Language System resources improve sieve-based generation and BERT–based ranking for concept normalization 2020 Dongfang Xu ,1 Manoj Gopale,2 Jiacheng Zhang ,3 Kris Brown,4 Edmon Begoli,4 and Steven Bethard	Authors designed a sieve-based system over the training data, Unified Medical Language System (UMLS) preferred terms, and UMLS synonyms to generate a list of possible concepts for each mention.They then design a list-wise classifier based on the BERT neural network to rank the candidate concepts, integrating UMLS semantic types through a regularizer.	A major challenge is the unseen mentions and concepts: 50.76% (29.85%) of test mentions (concepts) were not seen in the training data. Systems that memorize the training data or rely on it to determine the space of output concepts will thus perform poorly. Also Lexical and grammatical variations are pervasive in such text, posing key challenges for data interoperability and the development of natural	Analysis of the model shows that prioritizing UMLS preferred terms yields better performance, that the UMLS semantic type regularize results in qualitatively better concept predictions, and that the model performs well even on concepts not seen during training.

			language processing (NLP) techniques.	
8.	A transformer based approach for fighting COVID-19 fake news 2021 S.M. Sadiq-Ur-Rahman Shifath1, Mohammad Faiyaz Khan2, and Md. Saiful Islam3	Authors performed experiments primarily on traditional language models such as Bidirectional LSTM(Bi-LSTM) with attention, 1 dimensional CNN(1D-CNN), Hierarchical Attention Networks(HAN), Recurrent convolutional Neural Networks(RCNN), and Multichannel CNN with Attention(AMCNN) on the competition dataset. We also experiment with transformer-based pre-trained models like BERT and RoBERTa.	Authors tested different hyper-parameters like the number of layers, number of units in a layer, learning rate, weight decay, dropouts, normalization, etc. within a feasible range which was a very difficult job, also they faced resource limitation for experimenting with larger models.	Authors have presented our overall workflow for the fake news detection task. They have conducted a number of experiments and provided a comprehensive solution based on modified transformers with additional layers and An ensemble classifier.
9.	A Comparative Analysis Of Classifiers Used For Detection of Clickbait In News Headlines. Aaryaman Bajaj , Himanshi Nimesh , Raghav Sareen , Dinesh Kumar Vishwakarma. Proceedings of the Fifth International Conference on Intelligent Computing and Control Systems (ICICCS 2021).	The authors compare the performance of different classifiers in detecting the clickbait headlines of news articles by performing the extraction of new features from a multi-source dataset. Random Forest classifier yields a better accuracy than Naïve Bayes and Logistic Regression models in identifying headlines disseminating misleading information.	New clickbait formats are added each year, and many new methods can be incorporated into the model, to further improve accuracy. There is a high degree of similarity between the evaluation performance of the proposed model and other existing models.	They obtained scores on applying the various methods. We got the best results from Random Forests. Random Forest accuracy 0.891.

10.	BERT, XLNet or RoBERTa: The Best Transfer Learning Model to Detect Clickbaits 2021 Authors : PRABODA RAJAPAKSHA , (Student Member, IEEE), REZA FARAHBAKHSH , (Member, IEEE), AND NOEL CRESPI, (Member, IEEE)	:Based on the author’s knowledge, this is the first attempt to adapt Transfer Learning to classify Clickbaits in social media. In this work they have fine-tuned BERT, XLNet and RoBERTa models by integrating novel configuration changes into their default architectures such as model expansion, pruning and data augmentation strategies. Authors have used three fine-tuning approaches, namely; model generalization, expansion and pruning. The analysis has shown that pruning performed better than model expansion. In the expansion, the best result is achieved when we generated the output from hidden states without directly using pooled output (the default model output).	There is no significant performance improvement when each model expanded by adding an extra RNN layer(s). Apart from that, we experimented with another labelled clickbait dataset (Kaggle clickbait challenge) to explore the performance of our fine-tuned models under different scenarios.	The results shown that, RoBERTa outperformed the BERT and XLNet in many experiments mainly when we fine-tuned the model using hidden outputs to generate the output vector without using the pooled output and adding a non-linear layer at the end. This model architecture is considered to be the best performed model in our experiments.

4. PROBLEM STATEMENT-

Clickbait refers to a certain kind of headline that attracts people to click but gives something uncorrelated in that Link. The motivation behind click-baiting is to boost site traffic (and therefore, advertisement revenue) by exploiting the curious nature of human readers.

This technique works by dangling a hyperlink with enticing headlines to lure people into clicking; and then redirect them to the publishers' own websites which are uncorrelated to the headline. The discrepancy between the headline and the destination content wastes online readers significant amount of time on contents of which they have no interest. To address this problem, we propose a Clickbait Detector with Bidirectional Encoder Representations from transformers which could effectively identify clickbait.

The relation between the both things headline and the destination content wastes online readers significant amount of time on contents of which they have no interest. To address this problem, we propose Click-BERT: (Clickbait Detector with Bidirectional Encoder Representations from Transformers), which could effectively identify clickbaits utilizing state-of-the-arrt pre-training methods and self-attentive network. We approach this task as a regression problem in our two parallel baseline models for benchmarking with previous models. The model takes the post title and the linked content as input and will output a clickbait score in the range of [0, 1] with 0 indicating non-clickbait and 1 indicating clickbait. By training on a large twitter posts corpus with annotations of their 'clickbaitness' on a scale of [0, 1], we expect our model to be capable of capturing clickbait patterns in the headline and the content.

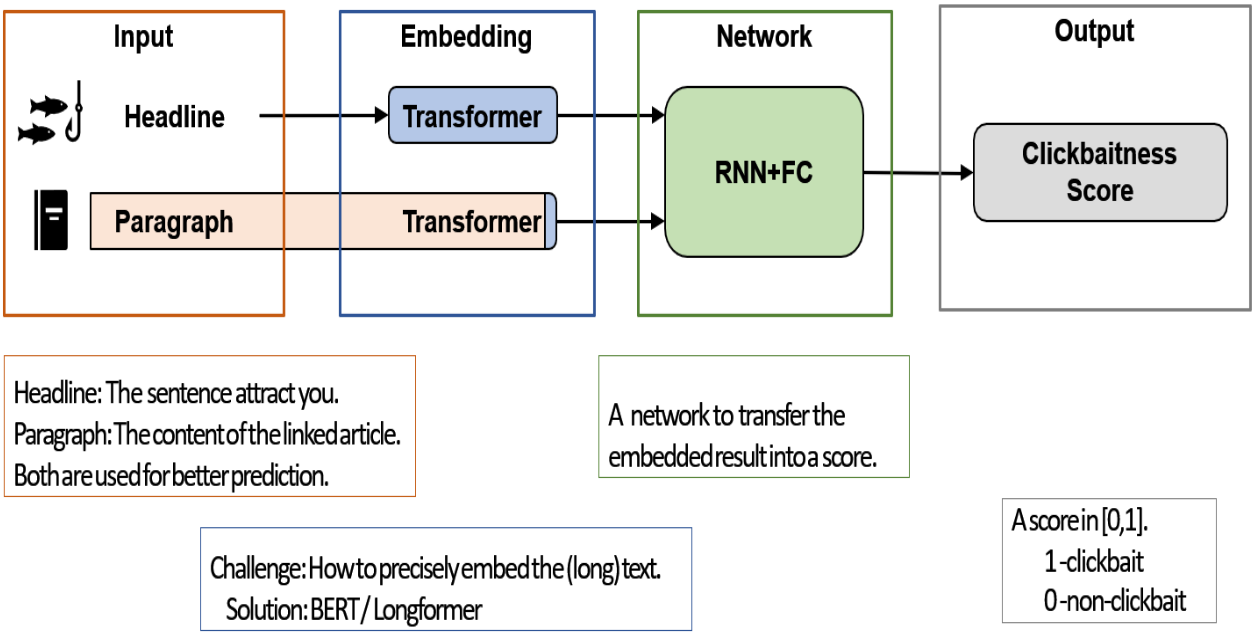
Main challenges of the clickbait detection problems lies-

in how well our model capture the meaning and correlation of the input headline/content (which differs greatly in length) and how properly the followed analysis gets performed. And our main contributions include:

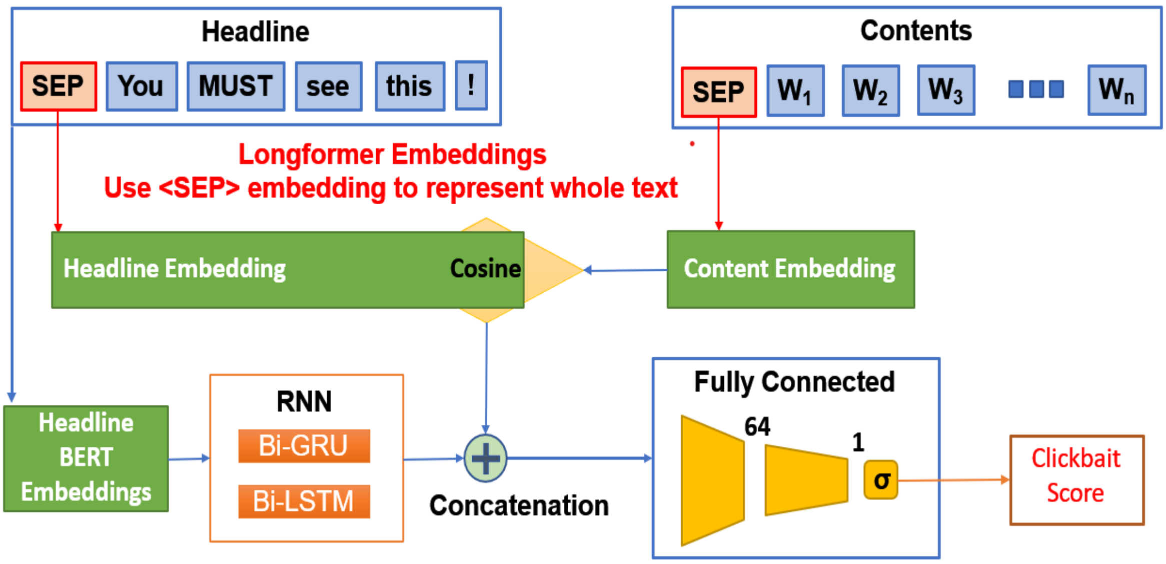
1. We applies advanced pre-trained models BERT and Longformer to extract sequence (headline/content) embedding to form a better understanding of the headline and the content.

2. We proposes a parallel model structure to integrate both prediction of whether the headline is luring people to click and prediction of whether the headline is related with the content into final judgement.

4.1 ARCHITECTURE DIAGRAM



4.2 FLOW DIAGRAM



4.3 PSEUDO CODE

Step 1 : Read the data

- train_file = instances.jsonl
- test_file = truth.jsonl
- df_train, df_test = read train, read_test
- size = train.shape[0]

Step 2 : Create the dictionary

- truth_id, truth_mean = list test(id), list test(mean)
- truth_dict = truth_id[i]:truth_mean[i] for all I
- train_id, train_post, train_text = list(id, heading, content)
- creating corpus = join(id, post , text) with truth_dict

Step 3 : Cleaning of data (discard tweets with 0.3 < score < 0.7)

- initial_length = size = 19538
- cleaned_web17 = new List [ ]
- iterating from i = 0 to size and if condition match append in cleaned_web17
- condition = 0.3 < mean score < 0.7
- new_final_length = 12963

Step 4 : Bert Embedding

Download BERT

bert_tokenizer = BertTokenizer.from_pretrained("bert-base-uncased") bert_model = BertModel.from_pretrained("bert-base-uncased")

save BertTokenizer, BertModel.

now encode text into sequence of IDs.

encode1 = torch.tensor(bert_tokenizer.encode(web17.corpus[0][0]))

print(encode1.shape)

Step 5 : Data Profiling

- extract data; title_all,content_all,score_all = [data in web17.corpus]

-title_all_token = bert_tokenizer()

- Print the mean no. Of token, ID etc.

-Average # of tokens = 17.628058143105743

- content_all_token = bert_tokenizer()

- Print the mean no. Of token, ID etc.

-Average # of tokens = 791.2599037772546

Step 6 : Extract embeddings & divide train/val/test set

-title_all_tokenized = bert_tokenizer()

-Save it as a Py Torch file

-Give train_size and val_size

-shape gives batch size of 800

Step 7 : Process by patches and combine

-import torch and gc

-extract_size=800 // One batch

-for loop form I =0 to num_data//800

-outputs = bert_model()

-check the shape

-save the title_all_embed as per the shape

-save the last 501 content and title separately (from 19200 to 19538)

Step 8 : Loading Data that we saved as pythorch tensor files

- importing TensorDataset and DataLoader
- Xt_all = torch.load ('titles_all.pt')
- yt_all = torch.load ('scores.pt')
- diving training, validation size = 10000, 2000
- test size = total - training - val = 963
- batch_size = 64
- using TensorDataset on train, val and test
- using DataLoader on train, val and test

Step 9 : Defining LSTM Model Architecture

- importing torch.nn
- class LSTM inherit base class nn.Module
- _init_ constructor : batch size , num_tokens, embed_dim, hidden_dim, n_layers, dropout
- self.LSTM = embed_dim, hidden_dim, n_layers, batch_first=True, dropout=dropout, bidirectional=True
- defining two fully-connected layers :

  self.fc1=nn.Linear(2*hidden_dim, 64)

  self.fc2=nn.Linear(64, 1)

- defining forward function LSTM
- lstm_out, hidden = self.lstm(x.unsqueeze(1), hidden)

  flat = lstm_out.squeeze()

  out1 = self.fc1(flat)

  out2 = self.fc2(torch.relu(out1))

  out = torch.sigmoid(out2)
- iterating over the parameters
- defining init_hidden(batch_size) :

  hidden = (weight.new(self.n_layers*2, batch_size, self.hidden_dim).zero_()

  weight.new(self.n_layers*2, batch_size, self.hidden_dim).zero_()
- initializing the weights using xavier uniform (normal)
- torch.nn.init.xavier_uniform_(m.weight)

  m.bias.data.fill_(0.0)

Step 10 : Hyper-parameters Initialization

- hidden_dim = 10
- dropout = 0.2
- optimizer = Adam Optimizer
- learning rate = 3e-4
- n_layers = 2
- importing learning rate scheduler
- hyper-parameters of lr_scheduler :

  optimizer, 'min', factor=0.25, patience=0, threshold=0.05,min_lr=3e-5, verbose=True

Step 11 : Training and Testing

- define training function with parameters as :

train_dataloader, y_truth, model, loss_fn, optimizer, mute = False

- y_pred_train = []
- enumerate over train_dataloader
- for batch, (X, y) in enumerate(train_dataloader):
- Compute prediction error

pred, hidden = model(X, hidden)

y_pred_train.extend(pred.squeeze().cpu())

loss = loss_fn(pred.squeeze(), y)

- Backpropagation

loss.backward( )

optimizer.step( )

- define testing function with parameters same as training function and mode
- mode = 0: validation when training (lr_scheduler)

mode = 1: validation

mode = 2: test

- evaluating the model using four metrices

Loss, Accuracy, F1Score, Pearson Coefficient

Step 12 : Running the model for 5 epochs

- epochs = 5
- model.train()
- best_val_performance = 1.0
- train(train_dataloader, yt_all[:train_size], model, loss_fn, optimizer)
- val_performance = test(val_dataloader, yt_all[train_size:train_size+val_size], model, loss_fn, lr_scheduler)

Step 13 : Loss Function Optimizer and Accuracy

- hidden_dim = 10 # num of tokens is typically 20
- _ , num_tokens, embed_dim = Xt_all.shape
- dropout = 0.2
- Using MSELoss as a loss function

  loss_fn = nn.MSELoss()
- Using Adam Optimizer with learning rate 3e-4

  optimizer = torch.optim.Adam (model.parameters(), lr=3e-4)
- using learning rate scheduler

  lr_scheduler = ReduceLROnPlateau(optimizer, 'min', factor=0.25, patience=0, threshold=0.05, min_lr=3e-5, verbose=True)

Step 14 : Testing on validation and test data

- _ = test(val_dataloader, yt_all[train_size: train_size+val_size], model, loss_fn, lr_scheduler, mode = 1)
- _ = test(test_dataloader, yt_all[train_size+val_size:], model, loss_fn, lr_scheduler, mode = 2)

5. EXPERIMENTS AND RESULTS

5.1 DATASET

Webis-Clickbait-17 Dataset (19538 Tweets)

Link-

https://zenodo.org/record/5530410#.YjIb5XpBxhF

On the Webis 17 dataset, we conduct experiments. contains a total of 38,517 tweets from 27 major US news organisations. They'd been categorised according to how clickbaity they were. The title and content of the article were included in these tweets, as well as supplemental information like the target description, target keywords, and related photos. The data has previously been divided into two sets: a train set (19,538 posts, 4761 of which are clickbaits and 14,777 non-clickbaits) and a test set (19,538 posts, 4761 of which are non-clickbaits) (18,979 posts).

Five Amazon Mechanical Turk annotators rated each post on a 4-point scale [not click baiting (0.0), mildly click baiting (0.33), significantly click baiting (0.66), and heavily click baiting (1.0). Most annotators estimate a total of 9,276 postings to be clickbait.

5.1.1 METHODOLOGY

BERT and Longformer with Parallel Structure-

On top of the two baseline models, we construct our final model. The first baseline model depicts the relationship between the headline and the text. The second baseline model focuses on the headline interpretation. Both factors should be considered when determining if a tweet is a clickbait or not, according to the clickbait definition stated in our introduction section. As a result, we set up the two models in a parallel structure and combine their outputs. We effectively create an ensemble of the two baseline models in this way. Finally, to calculate the clickbait score, the outputs are mapped via a fully linked layer that is activated by the Sigmoid function. To get around the BERT model's input length limitation, we modified the encoding layer in the second baseline model to Longformer, which works well with long texts. Because the performance difference between Bi-LSTM and Bi-GRU is minor, we only tried both Bi-LSTM and Bi-GRU in the recurrent neural network block.

5.1.2 OUTPUT

Setting up our project : installing libraries and defining directory

0 - Setup

In [1]:
```
import pandas as pd
import numpy as np
import os

import warnings
warnings.filterwarnings('ignore')

dir = 'C:\\Users\\kaust\\Desktop\\Click-BERT-main\\clickbait17-validation-170630\\'
```

In [2]:
```
dir
```

Out[2]:
```
'C:\\Users\\kaust\\Desktop\\Click-BERT-main\\clickbait17-validation-170630\\'
```

In [3]:
```
# !pip install transformers
# !pip install tensorflow
# !pip install torch
```

In [4]:
```
from transformers import pipeline;
print(pipeline('sentiment-analysis')('we love NLP'))
```

```
No model was supplied, defaulted to distilbert-base-uncased-finetuned-sst-2-english (https://huggingface.co/distilbert-base-unc
ased-finetuned-sst-2-english)
```

```
[{'label': 'POSITIVE', 'score': 0.9997754693031311}]
```

Loading the data and Defining the data corpus Web17 class. The we created a dictionary of id and mean of the content. Then we defined the corpus having post content and dictionary.

1 - Data Corpus

- data = 19538

In [17]:
```
class Webis17:
    '''
    self.corpus: (post, text, truthMean)
    '''
    def __init__(self, path):
        self.train_file = path + 'instances.jsonl'
        self.truth_file = path + 'truth.jsonl'
        df_train = pd.read_json(self.train_file, lines=True)
        df_truth = pd.read_json(self.truth_file, lines=True)
        self.size = df_train.shape[0]

        truth_id, truth_mean = list(df_truth['id']), list(df_truth['truthMean'])
        truth_dict = {truth_id[i]:truth_mean[i] for i in range(self.size)}
        train_id, train_post, train_text = list(df_train['id']), list(df_train['postText']), list(df_train['targetParagraphs'])
        self.corpus = [(train_post[i][0], ' '.join(para for para in train_text[i]), truth_dict[train_id[i]]) for i in range(self.

#       print(self.corpus[:10])
```

In [18]:
```
# web17 = Webis17('./data/clickbait17/')
dir = 'C:\\Users\\kaust\\Desktop\\Click-BERT-main\\clickbait17-validation-170630\\'
web17 = Webis17(dir)
num_data = len(web17.corpus)
print(num_data)
```

```
19538
```

In [19]:
```
print(web17.corpus[0])
```

```
('UK’s response to modern slavery leaving victims destitute while abusers go free', 'Thousands of modern slavery victims have\x
a0not come forward, while others who have chosen to report their enslavers have ended up destitute as a result of insufficient
support, say\xa0MPs “Inexcusable” failures in the UK’s system for dealing with modern slavery are\xa0leaving victims reduced to
```

Next step is to clean the data. The raw clickbait scores ('truth Mean' label) are in a range of [0, 1], as an average of 5 scores coming from 5 annotators, and we decided to exclude tweets with a mean clickbait score that deviate no more than 0.2 from 0.5, or in the range of [0.3, 0.7]. These annotations shows little confidence to be distinguished as either clickbait or not, and might confuse our model.

2 - Dataset Preprocessing

Data Cleaning : Data Cleaning: discard tweets with 0.3 < score < 0.7

In [7]:
```
# for i in range(100):
#     print(web17.corpus[i][2])

cleaned_web17 = []
for i in range(0, 19538):
    if web17.corpus[i][2] < 0.3 or web17.corpus[i][2] > 0.7:
        cleaned_web17.append(web17.corpus[i])

print(len(cleaned_web17))
```

```
12963
```

In [8]:
```
print(cleaned_web17[1])
```

```
('this is good', "President Donald Trump has appointed the\xa0pro-life advocate and former president of Americans United for Li
fe (AUL), Dr. Charmaine Yoest, to be assistant secretary of public affairs for the Department of Health and Human Services (HH
S). National pro-life leader and Susan B. Anthony List President Marjorie Dannenfelser responded to the announcement of Yoest's
appointment with the following statement: Charmaine Yoest is one of the pro-life movement's most articulate and powerful commun
icators. As the former president and CEO of Americans United for Life, she led groundbreaking efforts to advance pro-life, pro-
```

Downloading and loading bert. BERT is a bidirectional transformer pre-trained using a combination of masked language modeling and next sentence prediction. The core part of BERT is the stacked bidirectional encoders from the transformer model, but during pre-training, a masked language modeling and next sentence prediction head are added onto BERT. The BERT Tokenizer is a tokenizer that works with BERT. It has many functionalities for any type of tokenization tasks.

2 - Dataset Preprocessing - BERT Embedding

Download BERT

In [8]:
```
from torch.utils.data import TensorDataset, DataLoader
from transformers import BertTokenizer, BertModel

bert_tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
bert_model = BertModel.from_pretrained("bert-base-uncased")

bert_tokenizer.save_pretrained(dir+'bert-base-uncased')
bert_model.save_pretrained(dir+'bert-base-uncased')

# it turns out that bert has limited token length of 512
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertModel: ['cls.predictions.bias',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight', 'cls.seq_relationship.bias', 'cls.predictions.transfo
rm.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.transform.dense.weight', 'cls.seq_relationsh
ip.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task or with another arc
hitecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
```

Load BERT

In [21]:
```
## load from files & tokenizer analysis

from torch.utils.data import TensorDataset, DataLoader
from transformers import BertTokenizer, BertModel

bert_tokenizer = BertTokenizer.from_pretrained(dir+'bert-base-uncased')
bert_model = BertModel.from_pretrained(dir+'bert-base-uncased')
```

Data Profiling is done. Data profiling is the process of reviewing the source data, understanding structure, content and interrelationships, and identifying potential for data projects. Title and content and score are extracted to get proper insights of data like average no of tokens, maximum no of tokens, id with maximum title, etc.

data profiling

In [22]:
```
## extract data
import torch
title_all = [data[0] for data in web17.corpus]
content_all = [data[1] for data in web17.corpus]
score_all = torch.tensor([data[2] for data in web17.corpus], requires_grad=True)
```

In [23]:
```
score_all
torch.save(score_all, dir+'/scores.pt')
```

In [24]:
```
# title profiling

title_all_tokenized_raw = bert_tokenizer(title_all,return_token_type_ids=False, return_attention_mask=False)['input_ids']
print(max([len(lst) for lst in title_all_tokenized_raw ]))
print(f"Average # of tokens = {np.mean([len(lst) for lst in title_all_tokenized_raw])}")
print(f"max # of tokens = {max([len(lst) for lst in title_all_tokenized_raw])}")
print(f"ID of title with max # of tokens = {np.argmax([len(lst) for lst in title_all_tokenized_raw ])}")
print("---the title---")
print(title_all[np.argmax([len(lst) for lst in title_all_tokenized_raw ])])
print("---the title---")
```

```
104
Average # of tokens = 17.628058143105743
max # of tokens = 104
ID of title with max # of tokens = 16508
---the title---
................
................
................
................
................
................
Okay, then...
---the title---
```

In [25]:
```
# content profiling
content_all_tokenized_raw = bert_tokenizer(content_all,return_token_type_ids=False, return_attention_mask=False)['input_ids']
print(f"Average # of tokens = {np.mean([len(lst) for lst in content_all_tokenized_raw])}")
```

```
Token indices sequence length is longer than the specified maximum sequence length for this model (1467 > 512). Running this se
quence through the model will result in indexing errors
```

```
Average # of tokens = 791.2599037772546
```

In [26]:
```
print(f"max # of tokens = {max([len(lst) for lst in content_all_tokenized_raw])}")
```

```
max # of tokens = 43357
```

All the title all then tokenized using bert tokenizer function with various attributes like padding truncation maxlength etc. We will divide our data in batches of 800 out of which 700 we will use for training and 100 for validation.

extract embeddings & divide train/val/test set

Raw

In [27]:
```
# All embeddings
title_all_tokenized = bert_tokenizer(title_all, padding=True,truncation=True,max_length=20, return_token_type_ids=False, return_a
print(title_all_tokenized.shape)
print(title_all_tokenized)
torch.save(title_all_tokenized, dir+'titles_tokens.pt')
```

```
torch.Size([19538, 20])
tensor([[  101,  2866,  1521,  ...,  2489,   102,     0],
        [  101,  2023,  2003,  ...,     0,     0,     0],
        [  101,  1996,  1000,  ...,  1996,  2047,   102],
        ...,
        [  101,  2413,  2015,  ...,  2112,  1997,   102],
        [  101,  2821,  5076,  ...,     0,     0,     0],
        [  101,  2957, 11011,  ...,     0,     0,     0]])
```

In [28]:
```
train_size = 700
val_size = 100
outputs = bert_model(title_all_tokenized[:(train_size+val_size), :])
title_all_embed = outputs[0]  # The last hidden-state is the first element of the output tuple
print(title_all_embed.shape) # batchsize x # tokens of sent x embed_dim
```

```
torch.Size([800, 20, 768])
```

Then we will process all the data by taking size of 800 each time and saving it as as a torch tensor file in our directory.

Process by patches

```
In [31]: import torch
         title_all_tokenized = torch.load(dir+'titles_tokens.pt')
         print(title_all_tokenized.shape)

         torch.Size([19538, 20])
```

```
In [32]: import gc

         num_data = 19538
         extract_size = 800
         for i in range(num_data//800):
             outputs = bert_model(title_all_tokenized[(extract_size*i):(extract_size*(i+1)), :])
             title_all_embed = outputs[0]  # The last hidden-state is the first element of the output tuple
             print(title_all_embed.shape) # batchsize x # tokens of sent x embed_dim
             print(f"From size {str(extract_size*i)} to {str(extract_size*(i+1))}")
             # save Data
             torch.save(title_all_embed, dir+'/titles_'+str(extract_size*i)+'_'+str(extract_size*(i+1)))
             del outputs
             del title_all_embed
             gc.collect()

         torch.Size([800, 20, 768])
         From size 0 to 800
         torch.Size([800, 20, 768])
         From size 800 to 1600
         torch.Size([800, 20, 768])
         From size 1600 to 2400
         torch.Size([800, 20, 768])
         From size 2400 to 3200
         torch.Size([800, 20, 768])
         From size 3200 to 4000
         torch.Size([800, 20, 768])
         From size 4000 to 4800
```

Extracting the last portion of the data and combining all in Xt

```
In [21]: # last portion

         num_patchs = num_data//extract_size
         outputs = bert_model(title_all_tokenized[(extract_size*num_patchs):, :])
         title_all_embed = outputs[0]  # The last hidden-state is the first element of the output tuple
         print(title_all_embed.shape) # batchsize x # tokens of sent x embed_dim
         print(f"From size {str(extract_size*num_patchs)} to {str(num_data)}")
         # save Data
         torch.save(title_all_embed, dir+'/titles_'+str(extract_size*num_patchs)+'_'+str(num_data))

         del outputs
         del title_all_embed
         gc.collect()

         torch.Size([163, 20, 768])
         From size 12800 to 12963

Out[21]: 0
```

Combine

```
In [22]: Xt = torch.zeros(num_data, 20, 768)
         for i in range(num_data//800):
             # curr_Xt = torch.load(dir+'/titles_'+str(extract_size*i)+'_'+str(extract_size*(i+1)))
             Xt[extract_size*i:extract_size*(i+1), :,: ] = torch.load(dir+'/titles_'+str(extract_size*i)+'_'+str(extract_size*(i+1)))
         Xt[extract_size*num_patchs:,:,:] = torch.load(dir+'/titles_'+str(extract_size*num_patchs)+'_'+str(num_data))

         print(Xt.shape)
         # print(Xt[-10:,:,:])

         torch.Size([12963, 20, 768])

In [23]: torch.save(Xt, dir+'/titles_all.pt')
```

Importing TensorDataset and Dataloader : The TensorDataset is an abstraction to be able to load and process each sample of your dataset lazily, while the DataLoader takes care of shuffling/sampling/weigthed sampling, batching, using multiprocessing to load the data, use pinned memory etc.

All (20 tokens)

In [24]:
```
# load data
import torch
from torch.utils.data import TensorDataset, DataLoader

dir = 'C:\\Users\\kaust\\Desktop\\NLP_Project\\Data\\'
Xt_all = torch.load(dir+'titles_all.pt')
yt_all = torch.load(dir+'scores.pt')
print(Xt_all.shape)
print(yt_all.shape)

num_data = Xt_all.shape[0]
train_size = 10000
val_size = 2000
test_size = num_data - train_size - val_size
batch_size = 64
train_set = TensorDataset(Xt_all[:train_size,:,:], yt_all[:train_size])
val_set = TensorDataset(Xt_all[train_size:train_size+val_size,:,:], yt_all[train_size:train_size+val_size])
test_set = TensorDataset(Xt_all[train_size+val_size:,:,:], yt_all[train_size+val_size:])

train_dataloader = DataLoader(train_set, batch_size=batch_size)
val_dataloader = DataLoader(val_set, batch_size=batch_size)
test_dataloader = DataLoader(test_set, batch_size=batch_size)
```

```
torch.Size([12963, 20, 768])
torch.Size([12963])
```

Only [CLS]

In [25]:
```
# load data
from torch.utils.data import TensorDataset, DataLoader

dir = 'C:\\Users\\kaust\\Desktop\\NLP_Project\\Data\\'
Xt_all = torch.load(dir+'/titles_all.pt')
yt_all = torch.load(dir+'/scores.pt')
print(Xt_all.shape)
print(yt_all.shape)

num_data = Xt_all.shape[0]
train_size = 10000
val_size = 2000
test_size = num_data - train_size - val_size
batch_size = 64
train_set = TensorDataset(Xt_all[:train_size,0,:], yt_all[:train_size])
val_set = TensorDataset(Xt_all[train_size:train_size+val_size,0,:], yt_all[train_size:train_size+val_size])
test_set = TensorDataset(Xt_all[train_size+val_size:,0,:], yt_all[train_size+val_size:])

train_dataloader = DataLoader(train_set, batch_size=batch_size)
val_dataloader = DataLoader(val_set, batch_size=batch_size)
test_dataloader = DataLoader(test_set, batch_size=batch_size)
```

```
torch.Size([12963, 20, 768])
torch.Size([12963])
```

Defining our LSTM model the outputs are again mapped by a fully connected layer, activated by the Sigmoid function to get the clickbait score. To overcome the input length restriction of the BERT model, we changed the encoding layer in the second baseline model into Longformer, which performs well on long texts. Here, we only tried both Bi-LSTM in the recurrent neural network block since the previous test result shows the performance difference between Bi-LSTM and Bi-GRU is trivial.

self.parameters() is a generator method that iterates over the parameters of the model.So weight variable simply holds a parameter of the model.Then weight.new() creates a tensor that has the same data type, same device as the produced parameter. Next retrieve the next item from the iterator by calling its next() method.here, it returns the first parameter from the class.

The Xavier initialization is exactly like uniform except Xavier computes the two range endpoints automatically based on the number of input nodes ("fan-in") and output nodes ("fan-out") to the layer.

4 - Model 1 - Simple LSTM

Model Architecture

In [26]:
```
import torch
import torch.nn as nn
import numpy as np

class LSTM(nn.Module):
    def __init__(self, batch_size, num_tokens, embed_dim, hidden_dim,  n_layers = 1, dropout = 0.0):
        super(LSTM, self).__init__()
        self.hidden_dim = hidden_dim
        self.n_layers = n_layers
        self.lstm=nn.LSTM(embed_dim, hidden_dim, n_layers, batch_first=True, dropout=dropout, bidirectional=True)
        self.flatten = nn.Flatten(1)
        # self.fc1=nn.Linear(num_tokens*hidden_dim, 64)
        # self.fc1=nn.Linear(num_tokens*hidden_dim, 1)
        # take CLS token, birection
        self.fc1=nn.Linear(2*hidden_dim, 64)

        self.fc2=nn.Linear(64, 1)

    def forward(self, x, hidden):
        '''
            x: batch_size x num_tokens x embed_dim
        '''
        # take CLS token
        # print(x[:,0,:].unsqueeze(1).shape)
        lstm_out, hidden = self.lstm(x.unsqueeze(1), hidden) # batch_size x 1 x (2*hidden_dim)

        # flat = self.flatten(lstm_out)
        flat = lstm_out.squeeze() # batch_size x hidden_dim

        out1 = self.fc1(flat) # batch_size x 64
        out2 = self.fc2(torch.relu(out1)) # batch_size x 1
        out = torch.sigmoid(out2)

        # # single layer
        # out = torch.sigmoid(out1)
        return out, hidden

    def init_hidden(self, batch_size):
        weight = next(self.parameters()).data
        # birections -> *2
        hidden = (weight.new(self.n_layers*2, batch_size, self.hidden_dim).zero_().to(device),
                  weight.new(self.n_layers*2, batch_size, self.hidden_dim).zero_().to(device))
        return hidden

def init_weights(m):
    '''
    Initialize weights
    '''
    if isinstance(m, nn.Linear):
        torch.nn.init.xavier_uniform_(m.weight)
        m.bias.data.fill_(0.0)
```

Hyperparameters of our model

	Hyperparameters
Bi-LSTM & Bi-GRU	Hidden dimension: 50 Layers: 2 Dropout: 0.2
FC Layer	Hidden dimension: 64 Dropout: 0.2
Loss Function	Mean Squared Error (MSE)
Weights Intialization	Xavier
Adam Optimizer	Learning rate : 10^{-4} Weight Decay: 10^{-3}
Learning rate scheduler (`ReduceLROnPlateu`)	Patience : 2 Weight Decay Factor: 0.25
Mini-batch size	8
Epochs	20

Hyperparamters

In [28]:

```
hidden_dim = 10 # num of tokens is typically 20
_ , num_tokens, embed_dim = Xt_all.shape
# dropout = 0.0
dropout = 0.2

model = LSTM(batch_size, num_tokens, embed_dim, hidden_dim, n_layers=2, dropout = dropout).to(device)
loss_fn = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=3e-4)

from torch.optim.lr_scheduler import ReduceLROnPlateau # learning rate scheduler
lr_scheduler = ReduceLROnPlateau(optimizer, 'min', factor=0.25, patience=0, threshold=0.05,min_lr=3e-5, verbose=True)

model.apply(init_weights)
```

Out[28]:

```
LSTM(
  (lstm): LSTM(768, 10, num_layers=2, batch_first=True, dropout=0.2, bidirectional=True)
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (fc1): Linear(in_features=20, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=1, bias=True)
)
```

During training, we used Mean Squared Error(MSE) as the loss function and Adam Optimizer with a starting learning rate of 10–4 , notice we applied an adaptive learning rate with decay by factor of 0.25 and patience of 2. We use Xavier weights initialization.

Defining our Training function with train dataloader nad loss function and the model and optimizer., We will iterate through our tensor dataloader dataset and compute prediction error and do backpropogation.

The gradients are "stored" by the tensors themselves (they have a grad and a requires_grad attributes) once you call backward() on the loss. After computing the gradients for all tensors in the model, calling optimizer.step() makes the optimizer iterate over all parameters (tensors) it is supposed to update and use their internally stored grad to update their values.

Training

.

In [29]:
```
from sklearn.metrics import f1_score
from scipy.stats import pearsonr

### Training ###
def train(train_dataloader, y_truth, model, loss_fn, optimizer, mute = False):
    model.train()

    size = len(train_dataloader.dataset)

    y_pred_train = []
    for batch, (X, y) in enumerate(train_dataloader):
        hidden = model.init_hidden(X.shape[0])
        X, y = X.to(device), y.to(device)

        optimizer.zero_grad()

        # Compute prediction error
        pred, hidden = model(X, hidden)
        y_pred_train.extend(pred.squeeze().cpu())
        loss = loss_fn(pred.squeeze(), y)
        # Backpropagation

        loss.backward()
        optimizer.step()

        if batch % 20 == 0:
            loss, current = loss.item(), batch * len(X)
            if not mute:
                print(f"Training loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")

    y_pred_train = torch.tensor(y_pred_train, dtype=float)
    performance = loss_fn(y_pred_train, y_truth)
    clf_performance = ((y_pred_train>0.5)==(y_truth>0.5)).float().mean()

    if not mute:
        print(f"Training Loss: {performance}")
        print(f"Training Classifier Accuracy: {clf_performance}")
    return y_pred_train
```

We are evaluating the performance of our model on 2 tasks: binary classfificiation task and regression task, the evaluation metrics include:

• For the binary classifification task, we used accuracy and F1 score as our evaluation metric, which is consistent with prior works ([16, 12, 15]) on this dataset for comparison.

• For the regression model, we used Mean Squared Error (MSE) as our metric, which is consistent with prior works ([16, 12, 15]) on this dataset for comparison.

Testing

```python
### Testing ###
def test(val_dataloader, y_truth, model, loss_fn, lr_scheduler, mute = False, mode = 0):
    '''
    mode = 0: validation when training (lr_scheduler)
    mode = 1: validation
    mode = 2: test
    '''
    hidden_val = model.init_hidden(batch_size)
    model.eval()

    y_pred_val = []
    for batch, (X, y) in enumerate(val_dataloader):
        hidden_val = model.init_hidden(X.shape[0])
        X, y = X.to(device), y.to(device)

        pred, hidden_val = model(X, hidden_val)
        y_pred_val.extend(pred.squeeze().cpu())

    y_pred_val = torch.tensor(y_pred_val, dtype=float)
    performance = loss_fn(y_pred_val, y_truth)
    if mode == 0:
        lr_scheduler.step(performance)
    clf_performance = ((y_pred_val>0.5)==(y_truth>0.5)).float().mean()

    f1_performance = f1_score((y_pred_val>0.5).float().numpy(), (y_truth>0.5).float().numpy()
    p_performance = pearsonr(y_pred_val.detach().numpy(), y_truth.detach().numpy())[0]
    if not mute:
        if mode == 2:
            print(f"Test Loss: {performance}")
            print(f"Test Accuracy: {clf_performance}")
            print(f"Test F1 Score: {f1_performance}")
            print(f"Test Pearson Coefficient: {p_performance}")
        else:
            print(f"Validation Loss: {performance}")
            print(f"Validation Accuracy: {clf_performance}")
            print(f"Validation F1 Score: {f1_performance}")
            print(f"Test Pearson Coefficient: {p_performance}")

    return performance
```

We have trained our model for 5 epochs but we can do it for 20 epochs, It is done for 5 to save time while adjusting with the accuracy.

Ongoing

```
In [30]: ## Training & validation

### ESTIMATED TIME: 2
# num * 20 * 768 -> 1 min per batch -> 2 hr per epoch
# CLS -> num * 1 * 768, hidden = 10, bidirectional -> 8 min per epoch
###


epochs = 5
model.train()

best_val_performance = 1.0 # any number works
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(train_dataloader, yt_all[:train_size], model, loss_fn, optimizer)
    val_performance = test(val_dataloader, yt_all[train_size:train_size+val_size], model, loss_fn, lr_scheduler)

    if val_performance < best_val_performance:
        best_val_performance = val_performance
        print(f'NEW BEST MODEL! Performance: {best_val_performance}')
        torch.save(model, dir+'/best_model')
print("Done!")
```

```
Epoch 1
-------------------------------
Training loss: 0.130494  [    0/10000]
Training loss: 0.120505  [ 1280/10000]
Training loss: 0.098896  [ 2560/10000]
Training loss: 0.072008  [ 3840/10000]
Training loss: 0.075095  [ 5120/10000]
Training loss: 0.059128  [ 6400/10000]
Training loss: 0.065695  [ 7680/10000]
Training loss: 0.046549  [ 8960/10000]
Training Loss: 0.07642852594365278
Training Classifier Accuracy: 0.8353999853134155
Validation Loss: 0.0462958023445965
Validation Accuracy: 0.9020000100135803
Validation F1 Score: 0.6
Test Pearson Coefficient: 0.6695715744944148
NEW BEST MODEL! Performance: 0.0462958023445965
Epoch 2
-------------------------------
Training loss: 0.044389  [    0/10000]
Training loss: 0.041818  [ 1280/10000]
Training loss: 0.047358  [ 2560/10000]
Training loss: 0.041306  [ 3840/10000]
Training loss: 0.033113  [ 5120/10000]
Training loss: 0.039051  [ 6400/10000]
Training loss: 0.032798  [ 7680/10000]
Training loss: 0.023674  [ 8960/10000]
Training Loss: 0.03807899840006584
Training Classifier Accuracy: 0.9101999998092651
Validation Loss: 0.03325891615124802
Validation Accuracy: 0.9175000190734863
Validation F1 Score: 0.728171334431631
Test Pearson Coefficient: 0.7501662041118375
NEW BEST MODEL! Performance: 0.03325891615124802
```

Reduce learning rate when a metric has stopped improving. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This scheduler reads a metrics quantity and if no improvement is seen for a ‘patience’ number of epochs, the learning rate is reduced.

We have uses Adam Optimizer for our model because we know Adam optimizer is used as a replacement optimizer for gradient descent and is it is very efficient with large problems which consist of a large number of data.Adam optimizer does not need large space it requires less memory space which is very efficient.

```
In [31]: #CLS, num * 1 * 768, hidden = 10, bidirectional -> 8 min per epoch
         torch.save(model, dir+'model_CLS_10_bi')
```

```
In [32]: import torch

         dir = "C:\\Users\\kaust\\Desktop\\NLP_Project\\Data\\"

         hidden_dim = 10 # num of tokens is typically 20
         _ , num_tokens, embed_dim = Xt_all.shape
         # dropout = 0.0
         dropout = 0.2

         model = LSTM(batch_size, num_tokens, embed_dim, hidden_dim, n_layers=2, dropout = dropout).to(device)
         model = torch.load(dir+'/best_model')

         loss_fn = nn.MSELoss()
         optimizer = torch.optim.Adam(model.parameters(), lr=3e-4)

         from torch.optim.lr_scheduler import ReduceLROnPlateau # learning rate scheduler
         lr_scheduler = ReduceLROnPlateau(optimizer, 'min', factor=0.25, patience=0, threshold=0.05,min_lr=3e-5, verbose=True)
```

```
In [33]: _ = test(val_dataloader, yt_all[train_size:train_size+val_size], model, loss_fn, lr_scheduler, mode = 1)
         _ = test(test_dataloader, yt_all[train_size+val_size:], model, loss_fn, lr_scheduler, mode = 2)

         Validation Loss: 0.03325891615124802
         Validation Accuracy: 0.9175000190734863
         Validation F1 Score: 0.728171334431631
         Test Pearson Coefficient: 0.7501662041118375
         Test Loss: 0.03108036533118153
         Test Accuracy: 0.9262720942497253
         Test F1 Score: 0.7526132404181185
         Test Pearson Coefficient: 0.7712170394031064
```

6. CONCLUSION

In this project, we proposed to build a Clickbait Detector with Bidirectional Encoder Representations from Transformers. It could be trained from end-to-end without involving any manual feature engineering. It will effectively identify clickbaits and non-clickbaits with high accuracy.

Our second baseline model beats the previous S.O.T.A on both accuracy and MSE [16] on Webis 17 dataset with a simple Bi-LSTM structure even without finetuning on the raw BERT architecture, and this demonstrates that:

1. The BERT embedding is significantly powerful in the problem setting and provides salient representation of headlines.

2. The RNN(Bi-LSTM) architecture fits this downstream task well, and we substitute LSTM with GRU for ablation study and get similar performance (~ 1% worse accuracy).

7. FUTURE WORKS

While our proposed model achieve S.O.T.A on accuracy and MSE, there are still rooms for improvement, possible directions for future work are listed below.

1. When using BERT and Longformer techniques, we are directly using the pre-trained models to do the task. However, fine-tuning these models with data before the usage can improve their ability for embedding the headline and contents. We didn’t carry out this procedure due to the limitations of compute resources.

2. Improve language understanding abilities for media idioms.

3. Incorporate feature engineering. We have built a classifier with features like readability score and the amount of special characters. However, our result shows that this method’s performance drops sharply when the dataset size increase and get surpassed by our baseline 2 models. However, we expect that integrate the extracted features with our current model could produce a better performance.

4. the dataset is imbalanced — there are approximately 5.5 times non-clickbaits training data than clickbait ones and that imbalancy do leads to some performance difference on classifying clickbait vs. non-clickbait examples. We may seek to sampling approaches that help with imbalanced regression like Synthetic Minority over-Sampling Technique for Regression with Gausian Noise (SMOGN [5]) and compare with the baseline

8. REFERENCES

[1] Clickbait Detection in YouTube Videos.

Authors- Ruchira Gothankar, Fabio Di Troia, Mark Stamp.

[2] exBAKE: Automatic Fake News Detection Model Based on Bidirectional Encoder Representations from Transformers (BERT). Authors- Heejung Jwa , Dongsuk Oh, Kinam Park, Jang Mook Kang and Heuiseok Lim.

[3] Clickbait Headline Detection in Indonesian News Sites using Multilingual Bidirectional Encoder Representations from Transformers (M-BERT). Authors-Muhammad Noor Fakhruzzaman , Sa’idah Zahrotul Jannah, Ratih Ardiati Ningrum, Indah Fahmiyah. 2021.

[4] Stop clickbait: Detecting and preventing clickbaits in online news media. Chakraborty, A., Paranjape, B., Kakarla, S. and Ganguly, N. IEEE.

[5] Bert: Pre-training of deep bidirectional transformers for language understanding, J., Chang, M.W., Lee, K. and Toutanova, K., 2018

[6] Detecting and Categorization of Click Baits, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) NTASU – 2020 (Volume 09 – Issue 03). Sainath Patil, Mayur Koul, Harikrishan Chauhan, Prachi Patil, 2021.

[7] Unified Medical Language System resources improve sieve-based generation and BERT–based ranking for concept normalization, 2020, Dongfang Xu ,1 Manoj Gopale,2 Jiacheng Zhang ,3 Kris Brown,4 Edmon Begoli,4 and Steven Bethard.

[8] A transformer based approach for fighting COVID-19 fake news, 2021 , S.M. Sadiq-Ur-Rahman Shifath1, Mohammad Faiyaz Khan2, and Md. Saiful Islam3.

[9] A Comparative Analysis Of Classifiers Used For Detection of Clickbait In News Headlines. Aaryaman Bajaj , Himanshi Nimesh , Raghav Sareen , Dinesh Kumar Vishwakarma.

[10] BERT, XLNet or RoBERTa: The Best Transfer Learning Model to Detect Clickbaits 2021. Authors : PRABODA RAJAPAKSHA , (Student Member, IEEE), REZA FARAHBAKHSH , (Member, IEEE), AND NOEL CRESPI, (Member, IEEE) .

[11] Paula Branco, Lu´ıs Torgo, and Rita P. Ribeiro. SMOGN: a pre-processing approach for imbalanced regression. In First International Workshop on Learning with Imbalanced Domains: Theory and Applications, LIDTA@PKDD/ECML 2017, 22 September 2017, Skopje, Macedonia, volume 74 of Proceedings of Machine Learning Research, pages 36–50. PMLR, 2017.

[12] Abhijnan Chakraborty, Bhargavi Paranjape, Sourya Kakarla, and Niloy Ganguly. Stop clickbait: Detecting and preventing clickbaits in online news media. CoRR, abs/1610.09786, 2016.

[13] Martin Potthast, Tim Gollub, Matthias Hagen, and Benno Stein. The clickbait challenge 2017: Towards a regression model for clickbait strength.

[14] Philippe Thomas. Clickbait identification using neural networks. CoRR, abs/1710.08721, 2017.

[15] A Comparative Analysis Of Classifiers Used For Detection of Clickbait In News Headlines. Aaryaman Bajaj, Himanshi Nimesh; Raghav Sareen, Dinesh Kumar Vishwakarma.