COMPARATIVE ANALYSIS OF VARIOUS ALGORITHMS FOR FAKE NEWS DETECTION

Thesis submitted in partial fulfilment
of the requirements for the degree of

Bachelor of Technology
In
Computer Science and Engineering

by

Prithwiraj Samanta

Under the guidance of

Dr. Rashmi Panda

भारतीय सूचना प्रौद्योगिकी संस्थान राँची
INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, RANCHI
(An Institution of National importance under act of Parliament)
(Ranchi - 834010), Jharkhand
Department of Computer Science and Engineering

Indian Institute of Information Technology Ranchi

Certificate

This is to certify that the thesis entitled "**Comparative Analysis of various Algorithms for Fake New Detection**" is a Bonafede record of work carried out by **Prithwiraj Samanta,** under my supervision and guidance, for the partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology (Honours)** in Computer Science and Engineering at the Indian Institute of Information Technology, Ranchi. The thesis has fulfilled all the requirements as per the regulations of the institute and in my opinion reached the standard for submission.

Dr. Rashmi Panda
Assistant Professor
Indian Institute of Information Technology

Date: December 2022
Place: IIIT Ranchi

Acknowledgement

I am greatly indebted to my project supervisor Prof. Rashmi Panda for her invaluable guidance and encouragement throughout the course of this project. I would like to take this opportunity to express my sincere and profound gratitude to her.

I would also like to gratefully acknowledge the suggestions and discussions received from my teammate Satya.

My sincere thanks also go to all the faculty of CSE for their kind co-operation in all spheres during my project work. I would also like to thank all others whose direct or indirect help has benefited me during my stay at IIIT Ranchi.

Furthermore, I would like to thank all other friends and fellow students of IIIT Ranchi and particularly my CSE mates for sparing great time throughout my B.Tech and very good support I received from them during my stay.

Last but not the least, I thank my parents and all my family members for their support and motivation they have provided throughout my project.

Prithwiraj Samanta

Abstract

The advent of the World Wide Web and the rapid adoption of social media platforms (such as Facebook and Twitter) paved the way for information dissemination that has never been witnessed in the human history before. With the current usage of social media platforms, consumers are creating and sharing more information than ever before, some of which are misleading with no relevance to reality. Automated classification of a text article as misinformation or disinformation is a challenging task. Even an expert in a particular domain has to explore multiple aspects before giving a verdict on the truthfulness of an article. In this work, we have created an analysis report of various algorithms (particularly LSTM + CNN + Attention + Transformer Models) for automated classification of news articles. Our study explores different textual properties that can be used to distinguish fake contents from real. By using those properties, we train a combination of different machine learning algorithms using various methods and evaluate their performance on real world datasets.

Content

List of Figures

List of Tables

1

Introduction

1.1 Background

With the advancement of technology, digital news is more widely exposed to users globally and contributes to the increment of spreading hoaxes and disinformation online. Fake news can be found through popular platforms such as social media and the Internet. There have been multiple solutions and efforts in the detection of fake news where it even works with artificial intelligence tools. However, fake news intends to convince the reader to believe false information which deems these articles difficult to perceive. The rate of producing digital news is large and quick, running daily at every second, thus it is challenging for machine learning to effectively detect fake news.

In the discourse of not being able to detect fake news, the world would no longer hold value in truth. Fake news paves the way for deceiving others and promoting ideologies. These people who produce the wrong information benefit by earning money with the number of interactions on their publications. Spreading disinformation holds various intentions, in particular, to gain favour in political elections, for business and products, done out of spite or revenge. Humans can be gullible and fake news is challenging to differentiate from the normal news. Most are easily influenced especially by the sharing of friends and family due to relations and trust. We tend to base our emotions from the news, which makes accepting not difficult when it is relevant and stance from our own beliefs. Therefore, we become satisfied with what we want to hear and fall into these traps.

1.2 LSTM Model

In theory, classic RNNs can keep track of arbitrary long-term dependencies in the input sequences. The problem with vanilla RNNs is computational (or practical) in nature: when training a vanilla RNN using back-propagation, the long-term gradients which are back-propagated can "vanish" (that is, they can tend to zero) or "explode" (that is, they can tend to infinity), because of the computations involved in the process, which use finite-precision numbers. RNNs using LSTM units partially solve the vanishing gradient problem, because LSTM units allow gradients to also flow unchanged. However, LSTM networks can still suffer from the exploding gradient problem.

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series.

1.3 CNN Model

CNNs are regularized versions of multilayer perceptron. Multilayer perceptron usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "full connectivity" of these networks makes them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.) CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme.

A convolutional neural network consists of an input layer, hidden layers and an output layer. In any feed-forward neural network, any middle layers are called hidden because their inputs and outputs are masked by the activation function and final convolution. In a convolutional neural network, the hidden layers include layers that perform convolutions. Typically, this includes a layer that performs a dot product of the convolution kernel with the layer's input matrix. This product is usually the Frobenius inner product, and its activation function is commonly ReLU. As the convolution kernel slides along the input matrix for the layer, the convolution operation generates a feature map, which in turn contributes to the input of the next layer. This is followed by other layers such as pooling layers, fully connected layers, and normalization layers.

1.4 LSTM + CNN Models

This involves using Convolutional Neural Network (CNN) layers for feature extraction on input data combined with LSTMs to support sequence prediction. CNN LSTMs were developed for visual time series prediction problems and the application of generating textual descriptions from sequences of images or sequence of text document.

This architecture is appropriate for problems that:

- Have spatial structure in their input such as the 2D structure or pixels in an image or the 1D structure of words in a sentence, paragraph, or document.
- Have a temporal structure in their input such as the order of images in a video or words in text, or require the generation of output with temporal structure such as words in a textual description.

1.5 RNN + Attention Models

Improved RNN models such as Long Short-Term Memory networks (LSTMs) enable training on long sequences overcoming problems like vanishing gradients. However, even the more advanced models have their limitations and researchers had a hard time developing high-quality models when working with long data sequences. In machine translation, for example, the RNN has to find connections between long input and output sentences composed of dozens of words. It seemed that the existing RNN architectures needed to be changed and adapted to better deal with such tasks.

Attention is a mechanism combined in the RNN allowing it to focus on certain parts of the input sequence when predicting a certain part of the output sequence, enabling easier learning and of higher quality. Combination of attention mechanisms enabled improved performance in many tasks making it an integral part of modern RNN networks.

1.6 Transformer Models

Attention mechanisms let a model draw from the state at any preceding point along the sequence. The attention layer can access all previous states and weights them according to a learned measure of relevancy, providing relevant information about far-away tokens.

A clear example of the value of attention is in language translation, where context is essential to assigning the meaning of a word in a sentence. In an English-to-French translation system, the first word of the French output most probably depends heavily on the first few words of the English input. However, in a classic LSTM model, in order to produce the first word of the French output, the model is given only the state vector of the last English word. Theoretically, this vector can encode information about the whole English sentence, giving the model all necessary knowledge. In practice this information is often poorly preserved by the LSTM. An attention mechanism can be added to address this problem: the decoder is given access to the state vectors of every English input word, not just the last, and can learn attention weights that dictate how much to attend to each English input state vector.

When added to RNNs, attention mechanisms increase performance. The development of the Transformer architecture revealed that attention mechanisms were powerful in themselves, and that sequential recurrent processing of data was not necessary to achieve the performance gains of RNNs with attention. Transformers use an attention mechanism without an RNN, processing all tokens at the same time and calculating attention weights between them in successive layers.

2

Review of Literature

The radical change in availability of technology has brought has been a great contribution in every aspect of life. Contrary to that, as fresh news content is rapidly being generated it is as important to test the truthfulness of the content and credibility of the source. Fake news Detection has been around actively for a decade now but the rate of producing digital news is large and quick, running daily at every second, thus it is challenging for machine learning to effectively detect fake news and it is very important to keep developing the techniques in order to keep up with pace.

[1] This paper is a perfect way to dive into the vast spectrum of Fake News Detection. Starting from the introduction and definition of fake news detection, it takes a very good look to various methods implemented to identify fake news and prevent it from releasing publicly. It talks about different machine learning traditional models used and ongoing deep learning models being used to improve the accuracy of identifying fake news.

[5] This paper tells us about accuracy of different models i.e., Naive Bayes, Decision trees, SVM, Neural Networks, Random Forest, XG Boost on different datasets. [Survey on Fake News Detection using Machine learning Algorithms] published at (IJERT) in 2021 shows that Random Forest yielded 65.6 accuracy on liar dataset [liar dataset] preceding naive bayes, svm, logistic regression and decision tree with 63.7, 63, 62.5, 60 accuracies.

[6] by Xuzhou in 2015 proposed a rumour detector which identifies trending rumours on twitter. The detector, which searches for rare but informative phrases, combined with clustering and a classifier on the clusters, yields surprisingly good performance. According to this detector, on twitter out of 50 candidate statements, about one third of them are real rumours.

[7] published by Ahmed in International Journal of Computer and Information Engineering in 2020 gives us all the relevant information regarding implementation of machine learning on fake news. It gives us an overview of Methodology, pre-processing and implementation tasks of a model. The passive aggressive classifier gives 0.93 accuracy on fake news dataset which is the maximum of all the classifiers used.

[8] published in 2020 at IEEE proposed a hybrid deep learning model (a combination of CNN and LSTM) which experiments with dimension reduction techniques and pre-processing. The dimensionality reduction methods it uses are principal component analysis [2018 on using Principal Component Analysis] and Chi-square [chi square

reduction] on fake news challenge dataset. The best accuracy is yielded by CNN-LSTM with PCA, that is 97.8%.
[9] this study shows an implementation of deep two-path semi-supervised learning model "dstl" on PHEME dataset. To train and test the model both labelled and unlabelled dataset is used. The model contains three CNNs. The performance of dstl is inspected with different ratios of labelled data. The proposed model surpasses the F-score of bidirectional recurrent neural (BRNN), 35.85%, by yielding F-Score of 57.98% with 30% labelled data.

[10] published in 2020 proposes a tree structure model having two branches with the idea of using CNN and LSTM parallel. As we know CNN is good at extracting spatial features and LSTM is good at finding long-term dependencies in data. Further it suggests to concatenate both output vector and implement SoftMax layer. The input data of the branches differ due to pre-processing methods and corpus representation. CNN yields best accuracy when used with character level embedding where content like URL information, emoji, stop words are also taken into account. RNN variants like LSTM, Bi-LSTM, GRU input data is ready after pre-processing and applying word embedding methods like FastText, which can embed the words successfully which are not present in the corpus. It yields an F1-score of 0.89 on self-mined dataset which consists of 17,289 Turkish tweets.

[2] The paper uses Fake news challenge dataset which has 75000 instances is divided into train and validation data. The baseline models used are CNN models and BERT. The proposed model uses attention layer with CNN and RNN. To improve the accuracy dropout layer and max pooling are used. The best accuracy it achieved is 71.21% on competition test set.

[3] Using Transformer Network] The dataset used is a combination of three datasets which are WILD dataset, LIAR dataset and a dataset taken from one of the Kaggle competition. Models like fine-tuned BERT, CNN-LSTM, CNN are used with embedding layer. The best accuracy of 90% is achieved by BERT.

[4] Recurrent Neural Network is proficient at detecting pattern on Sequential data. This paper experiments with various models containing RNN layer. The dataset used consists of 5800 tweets used in the work of Zubiaga et al[]. The model with LSTM layer performs the best with the accuracy of 82.29%.

3

Implementation

3.1 Data Pre-processing

The text data which is available to us for using in the fake news detection is full of noisy information and present in format that can't be directly used by our algorithms. So, we have done the data pre-processing to transform the data into a useable form.

We started with removing the null values present in the dataset. We have dropped the entire row in which any column is null.

$$df\_1 = pd.read\_csv("../Dataset/csv/train.csv", header = 0, index\_col = 0)$$
$$df\_1 = df\_1.drop([^{\prime} title^{\prime}, ^{\prime} author^{\prime}], axis = 1$$
$$df\_1.dropna(inplace = True)$$
$$print(df\_1.isnull().sum(axis = 0))$$

We have expanded the contaction(short forms and shorthands) used in the english text using contractions library of python. We converted the text into lowercase and have split the text into word list.

$$df\_1[^{\prime}text^{\prime}] = df\_1[^{\prime}text^{\prime}].apply(lambda\ x: [contractions.fix(word, slang = False).lower()\ for\ word\ in\ x.split()])$$

We have removed all form of puctuation and stop words from the processed word list.

$$df\_1[^{\prime}text^{\prime}] = df\_1[^{\prime}text^{\prime}].apply(lambda\ x: [re.sub(r^{\prime}[^\backslash w\backslash s]^{\prime}, ^{\prime\prime}, word)\ \ for\ word\ in\ x])$$
$$stop\_words = set(stopwords.words(^{\prime}english^{\prime}))$$
$$df\_1[^{\prime}text^{\prime}] = df\_1[^{\prime}text^{\prime}].apply(lambda\ x: [word\ for\ word\ in\ x\ if\ word\ not\ in\ stop\_words])$$

We have removed all special charecters and numbers from the processed word list.

$$df\_1[^{\prime}text^{\prime}] = df\_1[^{\prime}text^{\prime}].apply(lambda\ x: [word\ for\ word\ in\ x\ if\ re.search("[@\_!\#\$\%\^\&*()<>?/|\}\{\sim:0-9]", word) == None])$$

We have removed all non-english words from the word list and then again convert it back to text.

$english\_words = set(nltk.corpus.words.words())$
$df\_1['text'] = df\_1['text'].apply(lambda\ x: [word\ for\ word\ in\ x\ if\ word\ in\ english\_words])$
$df\_1['text'] = df\_1['text'].apply(lambda\ x: ' '.join(x))$

Till now we have removed the noise and unwanted data. Now we will be formatting the text to be used by models. We have use tfidf vectorizer to extract the features from the corpus(processed text) and convert it to word vector.

$tfidfVectorizer = TfidfVectorizer(use\_idf = True, stop\_words = 'english')$
$tfidfVectorizer.fit(df\_1['text'])$
$print(tfidfVectorizer.get\_feature\_names())$
$trainTfidfVector = tfidfVectorizer.transform(df\_1['text'])$

We have also created a separate form of dataset using tokenizer instead of vectorizer.

$tokenizer = Tokenizer()$
$tokenizer.fit\_on\_texts(df\_1['text'])$
$trainSequence =$
$np.array([np.array(seq)\ for\ seq\ in\ pad\_sequences(tokenizer.texts\_to\_sequences(df\_1['text']))])$

At last, we save both the format using pickle library of python. We have generated both the format used by common ML algorithms

$pickle.dump(tfidfVectorizer, open("../Dataset/tfidf\_vectorizer - 1.pickle", "wb"))$
$pickle.dump(tokenizer, open("../Dataset/tokenizer - 1.pickle", "wb"))$
$pickle.dump(trainTfidfVector, open("../Dataset/tfidf\_train - 1.pickle", "wb"))$
$pickle.dump(trainSequence, open("../Dataset/sequence\_train - 1.pickle", "wb"))$
$pickle.dump(df\_1['label'], open("../Dataset/label\_train - 1.pickle", "wb"))$

We tested the performance of our datasets using Multinomial Naïve Bayes Classifiers.

tfidfVectorizer = pickle.load(open("../Dataset/tfidf_vectorizer-2.pickle", "rb"))

trainTfidfVector = pickle.load(open("../Dataset/tfidf_train-2.pickle", "rb"))
df_train = pd.DataFrame(data = trainTfidfVector.toarray(),columns = tfidfVectorizer.get_feature_names())

testTfidfVector = pickle.load(open("../Dataset/tfidf_test-2.pickle", "rb"))
df_test = pd.DataFrame(data = testTfidfVector.toarray(),columns = tfidfVectorizer.get_feature_names())

trainLabels = pickle.load(open("../Dataset/label_train-2.pickle", "rb"))
testLabels = pickle.load(open("../Dataset/label_test-2.pickle", "rb"))

clf = MultinomialNB()
clf.fit(trainTfidfVector, trainLabels)
print(clf.score(trainTfidfVector, trainLabels))
print(clf.score(testTfidfVector, testLabels))

We selected one of the datasets among 3 based on the performance. Now we are ready to proceed with preparing the selected dataset for Deep Learning.

For preparing the data for Deep Learning, we will be using stemming instead of vectorization or tokenization. We will be using a python library, Port Stemmer.

```
ps = PorterStemmer()
corpus = []
for i in range(0, len(messages)):
  print("Status: %s / %s" %(i, len(messages)), end="\r")
  review = re.sub('[^a-zA-Z]', ' ',messages['text'][i])
  review = review.lower()
  review = review.split()
  review = [ps.stem(word) for word in review if not word in stopwords.words('english')]
  review = ' '.join(review)
  corpus.append(review)
```

We used one-hot encoder to encode the words and then generate the embedding matrix.

$$onehot\_rep = [one\_hot(words, vo\_size)\ for\ words\ in\ corpus]$$
$$onehot\_rep\_test = [one\_hot(words, vo\_size)\ for\ words\ in\ corpus\_test]$$

```
sent_length = 1000
embedded_doc=pad_sequences(onehot_rep, padding='pre', maxlen=sent_length)
embedded_doc_test=pad_sequences(onehot_rep_test, padding='pre', maxlen=sent_length)

pickle.dump(embedded_doc, open("../Dataset/embedded_doc.pickle", "wb"))
pickle.dump(embedded_doc_test, open("../Dataset/embedded_doc_test.pickle", "wb"))
```

We convert the matrix into NumPy array before feeding it to network.

```
X_final=np.array(embedded_doc)
y_final=np.array(y_train)
X_final_test=np.array(embedded_doc_test)
y_final_test=np.array(y_test)
```

3.2 LSTM + CNN Models

We will be analysing performance of different architecture of combination of CNN + LSTM networks.

Model-1

```
embedding_vector_feature = 100
model=Sequential()
model.add(Embedding(vo_size,embedding_vector_feature,input_length=sent_length))
model.add(LSTM(10))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

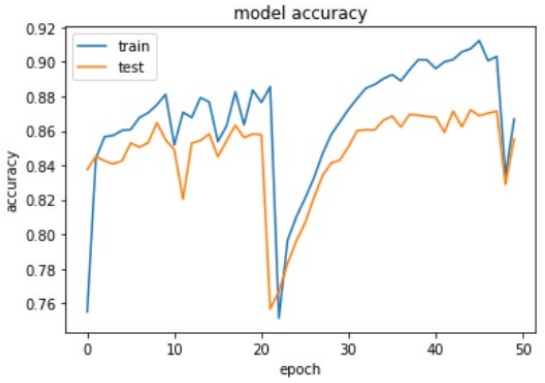

Figure 3.1

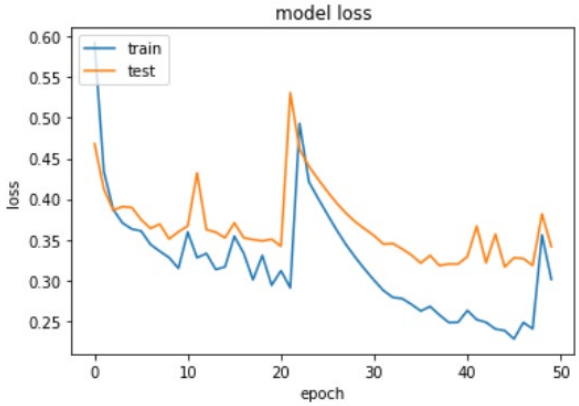

Figure 3.2

It is a simple LSTM model with 100 features. We observe that its validation accuracy is 86% though we have the training accuracy 2-3% higher. There is a good amount of oscillation in the accuracy during training and there is steep change in curve at few points.

Model-2

```
embedding_vector_feature = 300
model=Sequential()
model.add(Embedding(vo_size,embedding_vector_feature,input_length=sent_length))
model.add(LSTM(50))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

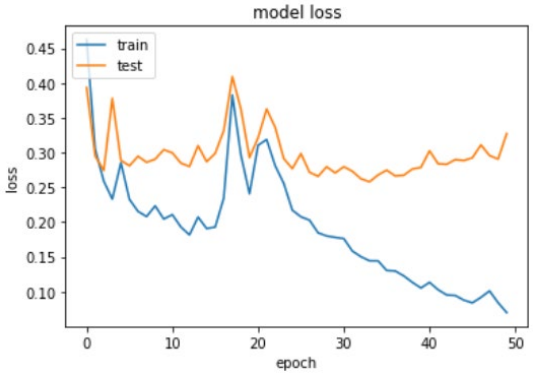

Figure 3.3

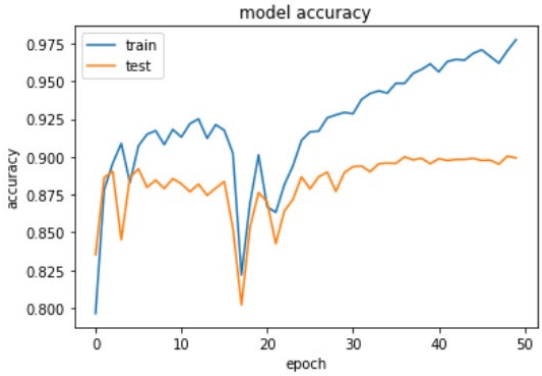

Figure 3.4

This is an upgrade of previous model with a greater number of features and neurons. It has a significant increase in the accuracy but takes a large amount of computation power due to large network.

Model-3

```
embedding_vector_feature = 300
model=Sequential()
model.add(Embedding(vo_size,embedding_vector_feature,input_length=sent_length))
model.add(LSTM(50))
model.add(BatchNormalization())
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

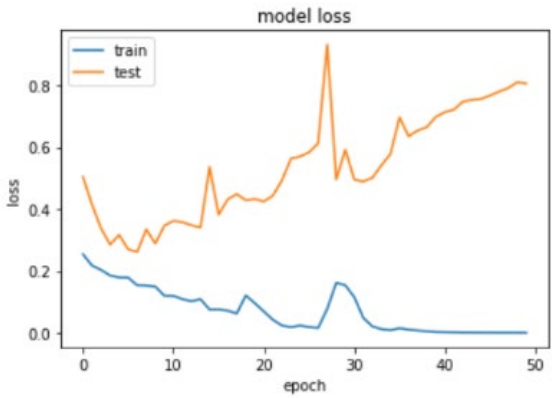

Figure 3.5

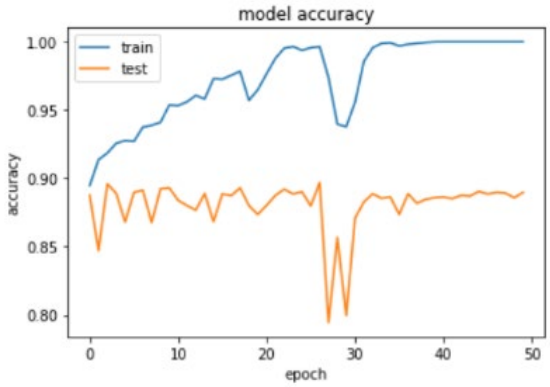

Figure 3.6

It is an upgrade of previous model with additional batch normalization layer. On adding batch normalization, it has boosted the overall accuracy of the previous model.

Model-4

```
embedding_vector_feature = 300
model=Sequential()
model.add(Embedding(vo_size,embedding_vector_feature,input_length=sent_length))
model.add(Conv1D(filters=25, kernel_size=5, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(50))
model.add(BatchNormalization())
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

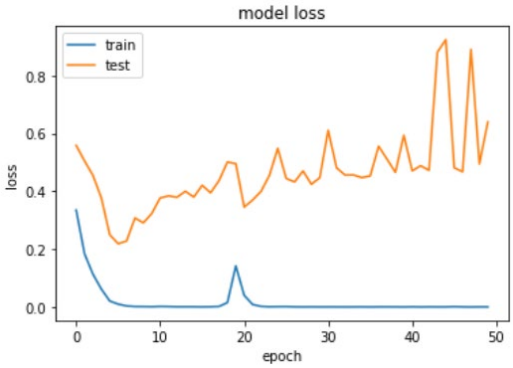

Figure 3.7

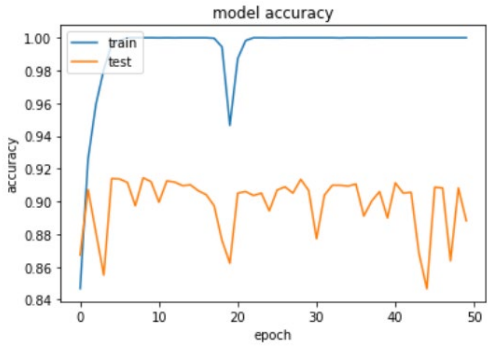

Figure 3.8

It is an upgrade of previous model with additional CNN network. CNN input the is used for detecting the features and then more refined details is passed to LSTM boosting its accuracy to 91%. Though we have used batch normalization it is not required that will be clear from our final model.

Model-5

```
embedding_vector_feature = 100
model=Sequential()
model.add(Embedding(vo_size,embedding_vector_feature,input_length=sent_length))
model.add(Conv1D(filters=25, kernel_size=5, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=5))
model.add(LSTM(25))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

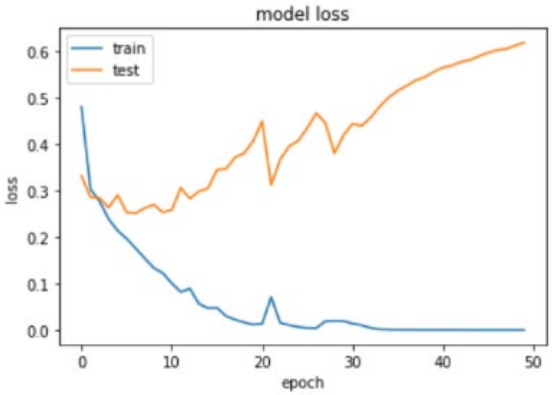

Figure 3.9

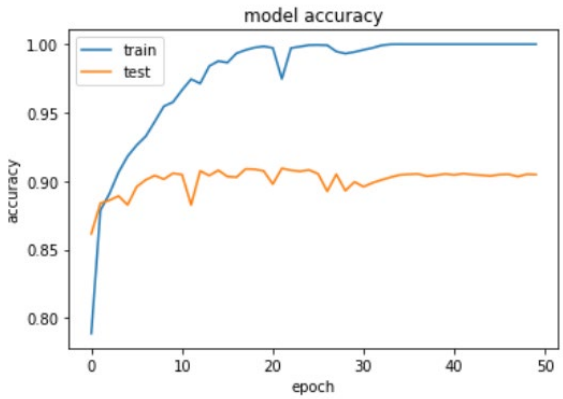

Figure 3.10

It is a downgrade of previous model with decrease in number of neurons and features and removal of batch normalization. We can observe that our final model uses much less computing power and features for same accuracy(as of only LSTM Models). Batch normalization is not required as the LSTM layer is already stabilized by the CNN layer.

3.3 RNN + Attention Models

Further, to improve the accuracy we will be analysing performance of different architecture of combination of LSTM + Attention layer networks.

Model-6

$$sequence\_input = Input(shape = (maxlen,), dtype = "int32")$$
$$embedding = Embedding(vocab\_size, output\_dim = EMBEDDING\_DIM, weights = [embedding\_vectors], input\_length = maxlen, trainable = \mathbf{False})(sequence\_input)$$
$$dropout = Dropout(0.2)(embedding)$$
$$conv1 = Conv1D(filters = 64, kernel\_size = 3, padding = 'same', activation = 'relu')(dropout)$$
$$maxp = MaxPooling1D(pool\_size = 2)(conv1)$$
$$\#(lstm, state\_h, state\_c) = LSTM(units = 128, return\_sequences = True, dropout = 0.2, return\_state = True)(maxp)$$
$$\#bn1 = BatchNormalization()((lstm, state\_h, state\_c))$$
$$(lstm, state\_h, state\_c) = LSTM(units = 128, dropout = 0.2, return\_sequences = \mathbf{True}, return\_state = \mathbf{True})(maxp)$$
$$context\_vector, attention\_weights = Attention(10)(lstm, state\_h)$$
$$densee = Dense(20, activation = 'relu')(context\_vector)$$
$$\#bn = BatchNormalization()(densee)$$
$$dropout2 = Dropout(0.2)(densee)$$
$$densef = Dense(1, activation = 'sigmoid')(dropout2)$$
$$model = tensorflow.keras.Model(inputs = sequence\_input, outputs = densef)$$
$$model.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['acc'])$$
$$display(model.summary())$$
$$model.fit(X\_train, y\_train, validation\_data = (X\_test, y\_test), epochs = 50)$$

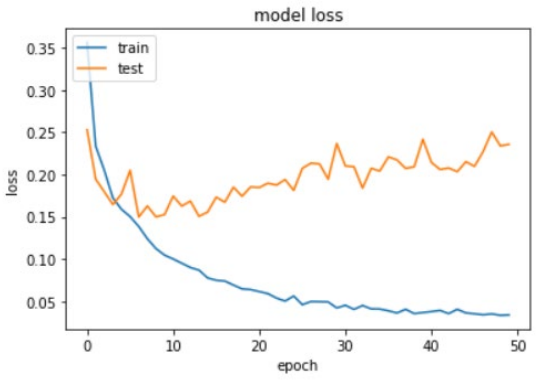

Figure 3.11

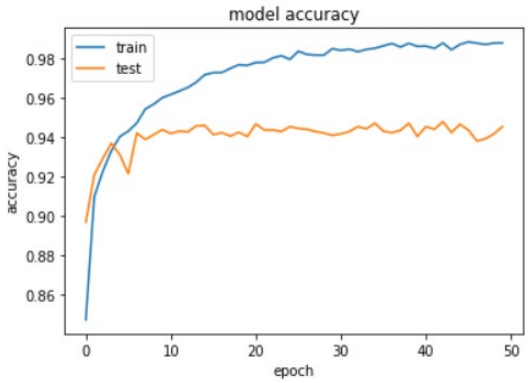

Figure 3.12

The model with attention layer gets a significant improvement on accuracy. The attention layer decides efficiently about the data model needs to focus more on. The model is more complex as it uses a greater number of layers than previous ones, but the computational cost for attention layer is less than recurrent layers.

3.4 Transformer Models

Model-7

```
def build_model(transformer, maxlen = 512):
    input_word_ids = Input(shape=(maxlen,), dtype=tf.int32, name="input_word_ids")
    sequence_output = transformer(input_word_ids)[0]
    cls_token = sequence_output[: 0, :]
    out = Dense(1, activation='sigmoid')(cls_token)
   model = Model(inputs=input_word_ids, outputs=out)
    model.compile(Adam(lr=1e-5), loss='binary_crossentropy', metrics=['accuracy'])
return model

# Configuration
  EPOCHS = 2
  BATCH_SIZE = 16 * strategy.num_replicas_in_sync
  MAX_LEN = 152
  MODEL = 'jplu/tf-xlm-roberta-large'
  tokenizer = AutoTokenizer.from_pretrained(MODEL)
  with strategy.scope():
  transformer_layer = TFAutoModel.from_pretrained(MODEL)
  model = build_model(transformer_layer, maxlen=MAX_LEN)
  model.summary()
```

Use of XLM-RoBERTa has boosted the accuracy significantly. XLM models specialize in finding hidden sentiments in the texts which led to the best result. The model used is a transformer model and takes more RAM and computational cost.

4

Results and Discussion

4.1 Result

Model	Val. Acc.	Train Acc.	Val. Loss	Train Loss	Test Acc.
LSTM only(Model-1) (100 features)	86%	88%	0.30	0.35	-
LSTM only(Model-2) (300 features)	89%	94%	0.12	0.26	-
LSTM only(Model-3) (300 features)(Batch Normalization)	89%	98%	0.50	0.10	-
LSTM + CNN (Model-4) (300 features) (Batch Normalization)	91%	99%	0.50	0.05	-
LSTM + CNN (Model-5) (100 features)	90%	99%	0.45	0.05	-
LSTM + Attention (Model-6) (100 features)	94%	99%	0.20	0.04	0.94
XLM-RoBERTa (Model-7) (100 features)	-	99%	-	-	0.99

Table 4.1

4.2 Discussion

In the final set of experiments, the proposed Roberta transformer model is trained on 20, 387 samples and tested on 5,127 headlines and articles. The training is performed using freely available TPU on Kaggle. The training takes 30 minutes to run epochs on 'Fake News' dataset using pre-trained word embedding which is in accordance with the desired model and to show the classification results. Before moving to using transformer model, we have experimented with

various combination of recurrent, convolution and attention layer. By analysing all the results, one can conclude that using Transformer model is more effective as it significantly improved the accuracy. The presented model outperforms all other models by producing an accuracy of 99.5%. The detailed statistical results of our proposed model are shown in Table. The statistical significance ensures that one can easily classify any news as fake or legitimate using our proposed model. The train and test, accuracy and loss are shown in the Result table.

5

Conclusion and Future Work

5.1 Conclusion

In our study we started with simple LSTM models and tested their performance which was proportional to the number of features and the size of network. But still it produced maximum accuracy up to 89%. We observed that LSTM + CNN models are capable of producing accuracy up to 90% with much smaller network compared to simple LSTM model. We used RNN + Attention model which has accuracy about 95%. While studying attention model further we found that attention model is itself sufficient to give the result. Then we switched to transformer models which has accuracy up to 99%. In our study we have able to find out how gradually we progressed from LSTM models to transformer models. Our study will help the future researcher to understand the how these models are derived from its predecessor models and what improved its performance from its predecessor.

5.2 Future Work

We are planning to work on detecting fake news shown in form of videos. We will be using our knowledge of text-based fake news detection and speech to text conversion. We will convert the speech in video into text, and then try to predict whether the news is fake or not. We may develop algorithm to identify the fake speaker and then warn the users against him/her. We have not yet created a data pipeline for our models. Our next work will contain data pipeline to automate the entire process of fetching data and converting it to required form. We will need to create the dataset as we have not found a suitable dataset for detecting fake news shown in form of videos.

List of Abbreviations

NN	Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
Val.	Validation
Acc.	Accuracy

References

[1] Xinyi Zhou and Reza Zafarani. 2020. A Survey of Fake News: Fundamental Theories, Detection Methods, and Opportunities. ACM Comput. Surv. 53, 5, Article 109 (September 2020). https://doi.org/10.1145/3395046

[2] Ayat Abedalla, Aisha AI-Sadi, and Malak Abdullah. 2019. A Closer Look at Fake News Detection: A Deep Learning Perspective. ICAAI Proceedings of the 2019 3rd International Conference in Artificial Intelligence (October 2019). https://doi.org/10.1145/3369114.3369149

[3] Jibran Fawaid, Asiyah Awalina, and Rifky Yunus Krishnabayu . 2021. Indonesia's Fake News Detection using Transformer Network. SIET. 6th International Conference on Sustainable Information Engineering and Technology (September 2021) https://doi.org/10.1145/3479645.3479666

[4] Oluwaseun Ajao, Deepayan Bhowmik, and Shahrzad Zargari. 2018. Fake News Identification on Twitter with Hybrid CNN and RNN Models. SMSociety. 9th International Conference on social media and Society. (July 2018). https:// doi.org/10.1145/3217804.3217917

[5] S. I. Manzoor, J. Singla and Nikita, "Fake News Detection Using Machine Learning approaches: A systematic Review," 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), 2019, pp. 230-234, https://doi.org/10.1109/ICOEI.2019.8862770.

[6] Zhao, Zhe & Resnick, Paul & Mei, Qiaozhu. (2015). Enquiring Minds: Early Detection of Rumours in social media from Enquiry Posts. 1395-1405. https://doi.org/10.1145/2736277.2741637.

[7] Sa, Ahmed & hinkelmann, knut & Corradini, Flavio. (2020). Development of Fake News Model using Machine Learning through Natural Language Processing.

[8] M. Umer, Z. Imtiaz, S. Ullah, A. Mehmood, G. S. Choi and B. -W. On, "Fake News Stance Detection Using Deep Learning Architecture (CNN-LSTM)," in IEEE Access, vol. 8, pp. 156695-156706, 2020, https://doi.org/10.1109/ACCESS.2020.3019735.

[9] arXiv:1906.05659 **[cs.CL]**

[10] M. U. Salur and I. Aydin, "A Novel Hybrid Deep Learning Model for Sentiment Classification," in IEEE Access, vol. 8, pp. 58080-58093, 2020, https://doi.org/ 10.1109/ACCESS.2020.2982538.