International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2020): 7.803

News Articles Tag Categorization using Neural Networks and Natural Language Processing

Ganesh Manohar Bhat

MSc. Data Science and Computational Intelligence Coventry University, London, England
ganeshmbhat96[at]gmail.com

Abstract: ***This research paper and its subsequent implementation is aimed at employing a combination of neural networks and natural language processing methodologies to successfully identify and categorize tags found in unmarked news articles and can be extended to other literary publications as well. These methodologies will then be applied onto the specific dataset containing a collection of news headlines associated with a brief description of the article along with other associated information. The result of this implementation will allow us to identify such tags in the future for unmarked articles. Unmarked news articles are becoming increasingly common these days and thus, make it even more difficult for the everyday user to identify and read forth articles that are of importance and of interest to him. In the present day, people are attracted to news articles based on their headlines primarily, and unmarked articles are a means of representation for unambiguous statements found not only in the headlines but also in the description of the article as well. Thus, the goal of this implementation is to successfully classify the tags found in both the headlines and description and to produce a clear representation regarding the category the article belongs under. Once the models have been successfully built, we will then evaluate their accuracy and hence determine their effectiveness for the use of similar datasets in the future. An extension of this paper can be used to identify the type of language being used in the articles in order to identify the different writing styles present. After the models have been successfully implemented their accuracy will be used to determine their effectiveness in predicting such similar datasets in the future. In this paper, the methods being used are Text based Convolutional Neural networks, Bidirectional Gated Research Unit (GRU) and a Long Short-Term Memory (LSTM) with Attention. The performance is then the topic of discussion based on how effectively the testing data is handled as visualized by the confusion matrices of the respective models.***

Keywords: Neural Networks, Gated Recurrent Units, Natural Language Processing, Long Short-Term Memory (LSTM), Text CNN, Tensorflow, Keras

1. Introduction

Most news publications have been moving towards the online market increasingly and have been a major source of internet traffic. These days, people don't have the time to visit shops and newsstands and browse through several newspaper publications just to find a paper that slightly interests them. Thus, the shift to online has increased sales for the companies by providing a simplified, easy to view and interpret reading experience with bold headlines to attract the user. From the consumer's perspective based on the majority's fast paced lives, they would first look at the headlines and then decide whether or not to view the article. But a majority of the headlines are literally ambiguous to the reader. The interpretation of the headlines themselves are varied to each user and are affected by factors such as the previously mentioned interpretability of the user, knowledge of the user on the particular subject etc. From the writer's perspective not only must they have a knack for writing brilliant headlines that drive the user's interest, but they must also employ the art of compression and allusion to make immediate sense to attract the reader to tell the news. From the paper by Mazhar Iqbal Rana et. al [3] we were able to gain the current state of working on the topic of text classification for news headlines and also how the efficiency and accuracy of a model would end up when analyzing over large quantities of text. In the paper by Petr Kroha et. al [2] we are evaluating the similarity of good news and bad news and their approach using classification methods and retrieving market news and as a result we were able to gain insight into the effect of considering the document frequency and similarity of those documents before the classification process. In the paper by Geert Brone and Seana Coulson [1], they have highlighted their experiment on the ambiguity found in the creation of headlines not only was it a complex process but also found that subjects were aware of the ambiguity in the headlines and lost interest when they encountered it and even the subjects who tried interpreting it failed multiple times. The example presented by them is as follows-take for example the headline "Russia takes the froth of Carlsberg results", this can be interpreted as either a major effect on Russia or it refers to the beer company Carlsberg. For our approach, we are not focusing on only the headlines as that would result in a simple text classification problem but to add complexity, we have extended onto the description that follows the headline in order to obtain the context of the article and to identify the style and category it belongs to. News Classification is not a new subject or topic of interest that has suddenly emerged to the forefront but has been there for many years and has seen many variations. But in this paper, we aim to use specific neural network models that should provide us a decent accuracy and performance that enables us to use the said models on similar datasets in the future. Most of these said papers and implementations aim to use machine learning models such as Naïve-Bayes Classifier, Logistic Regression etc. and have achieved an accuracy mostly around 75%. There are also papers done with neural network implementations like the one done by Svitlana Volkova et. al [9]. where they have used CNN's and LSTM to classify suspicious and trusted news posts from twitter. Their paper provided valuable insight into how the separation of the categories was to be made and how the embedding framework was supposed to work with GloVe embeddings [11]. The paper by Peng Zhou et. al [6] was referenced in order to gain knowledge on how bidirectional neural nets [5] take place

and how the addition of Attention to the network makes a difference when coupled with a regular LSTM [7]. Finally, the paper by Bin He et. al was referenced in order to gain knowledge into the working and known usage of the BiGRU layer and how it would affect a model [4].

The dataset that has been used in this implementation is a relatively new dataset and hasn't been cited in any papers as of yet at the time of this paper. The goal of this paper is to develop a model that provides significantly high accuracy as there hasn't been any attempt as of yet and so that we can create a framework and also to understand which approach is most suitable for this type of dataset and such similar datasets. In section II we discuss the dataset attributes and description in detail. In section III we discuss the data preprocessing steps that were applied before building the model while in section IV we explain a detailed picture of the models themselves and their working and in section V we put forth the implementation details and model building and how the model was trained and tested and their results followed by a discussion and possible interpretation of the said results is performed in section VI. Finally, in the conclusion of section VII we present on how likely and what model would be the most appropriate for applying the model onto similar datasets in the future and also the goals of this paper in the future and its benefits and applications. The appendix contains code that was developed using Spyder IDE which can be executed onto any similar python IDE for corroboration of the said results.

2. The Data Set

The dataset used for this paper was obtained from the HuffPost an online newspaper which produces articles regularly, and the data was obtained by means of a crawler and uploaded to Kaggle from where it was downloaded for this implementation [10]. The dataset is a Json format file containing over 200, 000 records of news articles, all of which are categorical values and each instance of the record has six features. These entries depict relative info about each article such as the headlines, links, descriptions etc. The articles themselves encompass a wide variety of topics ranging from sports to lifestyle to cooking to technology to education etc. The timeline for the data of the news articles is from the years 2012 to 2018. This research is instrumental in explaining the importance of tags to a news article and the reach it can get and the type of language that it can utilize. This paper also explains how to apply the below mentioned features of the dataset in order to identify the optimal model with its accuracy when applied to such types of datasets. The features of the dataset are as follows-

- Category: This attribute specifies the type of category the respective article belongs to. For example-Politics, entertainment, travel etc. are some of the categories. Overall there are 31 such categories.
- Headline: This attribute represents the headline of the specific news article.
- Authors: This attribute depicts the authors who have written the specific article.
- Link: This attribute provides a link to the respective article.
- Short_description: This attribute provides a brief summary of the respective article.
- Date: This attribute provides the date the article was published on.

Figure 1 depicts a snapshot of the dataset and its associated attributes along with their respective values. As mentioned previously all the data in the dataset are present as categorical values and none are recurring. When proceeding further in the following sections we will be employing a way for our models to interpret the data but not in the categorical format as it is difficult for a neural network model to be able to map the individual words to their corresponding meaning. Thus, we will be representing each word by a vector space and will be explained further in detail in the following sections. When applying this dataset to our models we wish to correctly map the words to their respective tags present in the articles so as to not develop a model which misinterprets tags found in an unmarked article.

	authors	category	date	headline	link	short_description
0	Melissa Jeltsen	CRIME	2018-05-26	There Were 2 Mass Shootings In Texas Last Week...	https://www.huffingtonpost.com/entry/texas-ama...	She left her husband. He killed their children...
1	Andy McDonald	ENTERTAINMENT	2018-05-26	Will Smith Joins Diplo And Nicky Jam For The 2...	https://www.huffingtonpost.com/entry/will-smit...	Of course it has a song.
2	Ron Dicker	ENTERTAINMENT	2018-05-26	Hugh Grant Marries For The First Time At Age 57	https://www.huffingtonpost.com/entry/hugh-gran...	The actor and his longtime girlfriend Anna Ebe...
3	Ron Dicker	ENTERTAINMENT	2018-05-26	Jim Carrey Blasts 'Castrato' Adam Schiff And D...	https://www.huffingtonpost.com/entry/jim-carre...	The actor gives Dems an ass-kicking for not fi...
4	Ron Dicker	ENTERTAINMENT	2018-05-26	Julianna Margulies Uses Donald Trump Poop Bags...	https://www.huffingtonpost.com/entry/julianna-...	The "Dietland" actress said using the bags is ...

Figure 1: Dataset Description

3. Data Preprocessing Stage

As mentioned earlier, the dataset is completely categorical in nature and as such if we want our model to produce an efficient implementation then conversion of the categorical data into numeric data is highly necessary and so even before the training and test split followed by some data preprocessing. In this data preprocessing we have used primarily Keras version 2.2.4 with Tensorflow 1.5 as the backend and all their associated libraries. Initially, for the purpose of data visualization we extracted all the unique categories present in the dataset. Figure 2 illustrates the result after printing the categories and the number of articles associated with each category.

```
total categories: 31
category
ARTS                    1509
ARTS & CULTURE          1339
BLACK VOICES            3858
BUSINESS                4254
COLLEGE                 1144
COMEDY                  3971
CRIME                   2893
EDUCATION               1004
ENTERTAINMENT          14257
FIFTY                   1401
GOOD NEWS               1398
GREEN                   2622
HEALTHY LIVING          6694
IMPACT                  2602
LATINO VOICES           1129
MEDIA                   2815
PARENTS                 3955
POLITICS               32739
QUEER VOICES            4995
RELIGION                2556
SCIENCE                 1381
SPORTS                  4167
STYLE                   2254
TASTE                   2096
TECH                    1231
THE WORLDPOST           3664
TRAVEL                  2145
WEIRD NEWS              2670
WOMEN                   3490
WORLD NEWS              2177
WORLDPOST               2579
dtype: int64
```

Figure 2: Categories and number of articles

From Figure 2 it is apparent that the categories "The WorldPost" and "WorldPost" are the same category. Thus, the next transformation made to the dataset is the combination of the mentioned two categories into a single category labeled as "WorldPost". Once accomplished we now have to move onto the text processing part. The data variable text was created as an amalgamation of the headline and associated description of each respective article. Thus, now each text variable represents a typical bag of words type model. Now in order to convert the categorical variables we employ the method of word embedding. Word embedding is a class of approaches for the representation of words and documents as a vector space method. Thus, here the text variables are represented as dense vectors where the vector represents the projection of the continuous vector space. This position of the word with respect to the vector space is called an embedding [16]. We have chosen to go with the Keras embedding scheme with the Tokenizer API to perform the data preparation for converting the text data into their individual words and then into sequences. The Tokenizer API then passes it to the Keras embedding layer which initializes the weights randomly. As for the learning of the embeddings themselves, we have chosen to go with the Global Vectors for Word Representation (GloVe). This is actually a package of pretrained words converted into a vector space in order to accelerate the learning of the words. In the sense of a definition it can be expressed as an unsupervised learning algorithm for obtaining vector representation of words. Basically, instead of us converting each word into a vector representation we use the GloVe package to correlate the words in our document and retrieve the vector representation of a word and thus reducing the time complexity of the entire program. Also, we use a padding length of 50 to fill the empty vector values. Once this is done, we then call the Keras embedding layer passing the index of the word vectors. The vector word index then becomes our independent variable labelled as 'X'. With regards to our dependent variable we have converted our categories into an integer representation for the purpose of our validation. Once the word index has been created along with the embedding vector's we now have to determine the size of each of them and thus, we have obtained 86627 unique tokens and 400, 000 word vectors. Once the embedding layer is completed, we will begin the split of the dataset into training and test set. Choosing an optimal split of paramount importance and thus we have chosen to go with the test size as being 20% causing the training size to be 80%.

4. Neural Network Techniques

1) Text CNN

Convolution Neural networks are a class of supervised deep learning neural networks that are usually used for analyzing visual imagery. They combine three architectural methods for ensuring that some degree of shift and distortion invariance, local receptive fields, shared weights and spatial and temporal subsampling take place efficiently [12]. The CNN basically consists of input, output and several hidden layers. These hidden layers typically consist of convolutional layers, an activation layer i. e. RELU, pooling layers, fully connected layers and normalization layers. This convolutional layer emulates the response of an individual neuron to stimuli, which in the case of this paper would be the word vectors themselves. The easiest way to interpret it would be to consider a convolution as a sliding window function that slides over the embedding matrix of vectors and simplifies into a convolved feature. This layer learns the features and identifies the parameters and hyperparameters that can be used to classify the data. The pooling layers are used to reduce the dimensionality of the data. It does this by combining the outputs of the convolution layer into a single neuron for the next layer and it combines these values either using the max or average of each convolution cluster of neurons. The convolution layers can be added at multiple times among the model building layers to improve the learning rate of the features and to generate a fully-connected output with the proper training weights. The RELU layer, an abbreviation for Rectified linear unit, is an activation function that can be used when the generated convolution matrix is predominant with negative values which can then be converted to zero values by the RELU function. In this paper we have elected to go with the RELU function as it trains the model faster than other activation functions like hyperbolic and sigmoid. After several iterations of the convolution and pooling layers we have to initiate the full connection of all the layers as this is how all the outputs of the previous layers will be transferred to the corresponding layer.

2) Bidirectional GRU with convolution

Recurrent Neural networks (RNN) are a class of supervised neural network models which behave similarly to feedforward networks but the difference between the conventional feedforward network is that the connections between the nodes form a directed graph amongst particularly a sequence i. e. these are networks with a loop feeding the output back to the input [20]. Figure 3 illustrates the typical structure of an RNN unit node.

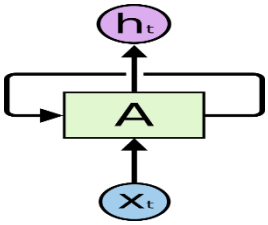

Figure 3: RNN unit node [17]

A Gated Recurrent Unit (GRU) is a variation of the RNN model and was proposed by Cho et al. [2014]. In this variation we make each recurrent unit to adaptively capture dependencies of different time scales [15]. The benefit of the GRU scheme is its ability to handle the vanishing gradient problem often faced by the standard RNN and also the difficulty in memorizing words from far away in the sequence [14]. Similar to the RNN LSTM model, a GRU also has several gating units that modulate and control the flow of information inside the unit but without having separate units handling this information. The main difference in these gating units is the presence of the update and reset gate. Figure 4 illustrates the typical unit of an GRU node.

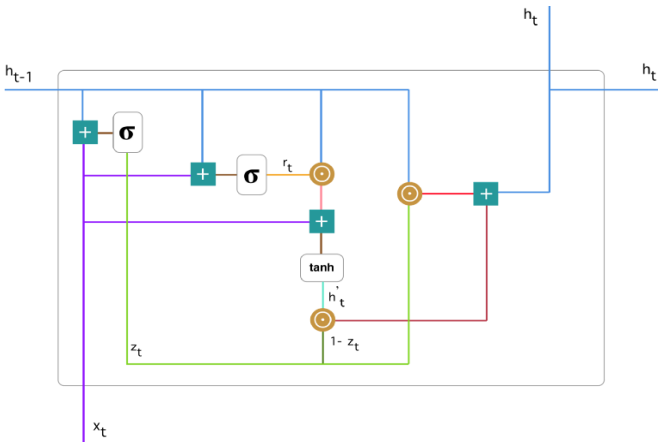

Figure 4: Gated Recurrent Unit [13]

With respect to the Update gate, this gate helps the model identify how much of the past information needs to be passed along to the future. This is an important contribution as it helps address the issue of the vanishing gradient by holding the current memory content and the memory content from the previous steps and deciding which subset of the information needs to be saved for future determination. The below mathematical representation depicts the update gates operation in processing the information.

$$z_t = \sigma\left(W^{(z)} x_t + U^{(z)} h_{t-1}\right)$$

Where x_t is plugged into the network unit multiplied by its own weight and the same goes for $h_{(t-1)}$. For the reset gate it is used to determine how much of the past information to forget and works hand in hand with the update gate to determine how much information to send to the next layer of nodes. The mathematical representation of the reset gate is as below.

$$r_t = \sigma\left(W^{(r)} x_t + U^{(r)} h_{t-1}\right)$$

The above representation is similar to the update gate's as the weight and gate usage is similar to it, but the difference is in their working. Now that we have a thorough and in-depth understanding of the basic principle of how the GRU works so we can now look at how bidirectional GRU affects a model along with convolution. Bidirectional GRU is realistically putting two independent GRU units together, but the working is such that the input sequence is fed in the normal i. e. positive time order sequence for one of the units and in the reverse time order for the other unit. The outputs of these two nodes are concatenated at each time order. The benefit of this structure allows the network to have both backward and forward information about each sequence at each time step. Thus, for the use of our paper, we have opted for bidirectional GRU as a hypothetical test to see if the addition of the reversed copy of the input sequence of word index will help the learning rate coupled with the time it might take to finish the computation. The idea for the implementation is to pass each line of the word index into the two parts of the BiGRU nodes providing each of them the word index and the reversed copy of the same part. As explained earlier traditional convolutional layers are used to test the features and their respective role in the dataset and by applying non-linearity and invariance to a regular CNN, but as in the paper by Gil Keren et. al, they proposed a model that enhances this feature extraction process for the case of sequential data by feeding patches of data into a regular RNN and using the outputs or the hidden layers of the recurrent units to compute the extracted features [8]. Thus, in this paper, we propose to use the same technique but with some modification as here we will use the convolution layer to slide through part of the word index (after the BiGRU layer has been fitted to the embedding layer) and build the convolutional layer followed by the pooling layer.

3) LSTM with Attention

The final model in our implementation is again a variation of Recurrent Neural networks called Long Short-Term Memory (LSTM) [19]. This special type of RNN is a network model capable of learning long-term dependencies and it does so due to the fact that they contain four neural network layers in each unit and are the most suitable for transferring entire vectors amongst the models themselves. Figure 5 contains the diagrammatic representation of the typical LSTM network along with a cross-sectional view of the underlying networks present in it.

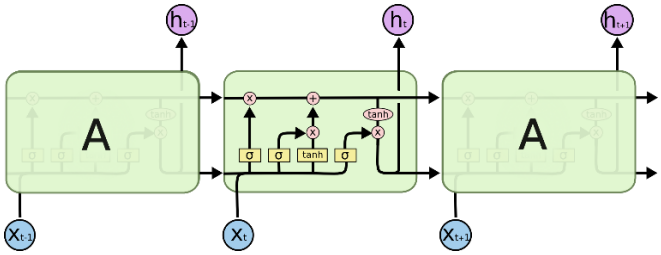

Figure 5: LSTM Neural Network [17]

The benefit of LSTM for our implementation is its ability to memorize large sequences of data thus allowing it to

store our large word vectors. For its working, it too contains gates that control the movement and modulation of the data that is flowing through it. The gates in this model are the input, output and forget gate. Each cell is capable of and responsible for maintaining the dependencies between the elements in the input sequence. The input gate controls the extent to which a new value is allowed to flow into the cell, the forget gate controls the time period the new value is allowed to remain in the cell and the output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit. The activation function used in our implementation is the softmax function. The addition of the attention mechanism is done to provide an extra sense of interoperability with the LSTM network. Attention is essentially a vector which is a byproduct of using the softmax activation function. Without the addition of Attention, the LSTM network would have to go through all the word vectors and compute the dependencies among the vectors. This is especially difficult if the sentence that we are processing is a large one with several hundreds of words. Thus, this would lead to a large amount of information loss, inadequate translation etc. The addition of attention does not combat all these problems or make a huge impact on the performance of a regular network, but still it has a way of correlating the dependencies found among words. The cost of the addition of attention is the bottleneck caused by the compulsory computation of the attention weight values at each cell of the LSTM i. e. we need to calculate the attention weight value for each combination of the input and output word. For example, if we had a 50-word input sentence and generated a 50-word output sentence then we would have 2500 attention values. But when it proceeds into hundreds of words then it would get computationally intensive. The other drawback being that if the attention vector is focused on a particular correlation and omits another correlation that might have the same weight as the other then we would have some significant loss in our output.

5. Experimental Setup

This dataset was obtained in the json format and the pandas library was used to obtain access to its contents (which are depicted in Figure 1). The major libraries being used for the implementation are the Keras version 2.2.4 with the tensorflow backend, Scikit-learn, seaborn, matplotlib and Numpy. The code was programmed in Spyder IDE. The next step that was performed was the data preprocessing phase as explained in section III. This phase was performed to bring the categorical data into a vectorized matrix of embeddings for the words present in the dataset. With this done we proceeded to split the dataset with the sklearn. model_selection library for the training and testing parts opting to go for the testing size as 20% of the dataset. Once this milestone was completed, we now proceeded to build each of our models as described in section IV of their working and merits. The goal of building the three models is to compare their respective accuracy and determine how good they are and also as to our approach about how the article information was processed.

The first model that was developed was the Text based CNN model. We have passed the embedding layer input to the input variable for the CNN model thus storing the word vectors in a list for processing by the convolutional layer. Thus, the next step is to call the convolutional layer. But before that we have set a varying kernel size integer values, those being 2, 3 and 4 just to find the best possible value for the convolution window. We then call the convolution layer with 64 filters and padding being ‘same’ meaning that the output of the convolution step will be padded to the same length as the input (i. e.50 as set previously for the vectors). For the activation function we have chosen to go with ‘RELU’ as it would help adjust the negative values present amongst our vectors to come forth to zero values. We are applying the convolution layer to the embedding matrix. This convolution layer is then passed to the pooling layer, which has a pool size of 3 indicating that the ‘max’ pooling scheme that is being applied will have a window size of 3X3. Before proceeding we have to first flatten the output so that we are left with a one-dimensional view of the pooled layers. The regularization scheme being used is Dropout Regularization with a value set to 0.1 indicating that 10% of our nodes are to be released as they may cause overfitting and unnecessary adaptations amongst our vectors. The next step involves creating the Dense layer with a softmax activation layer. While the compilation of our model is set to happen the parameters chosen for the factors such as accuracy, loss and optimizer are respectively set to categorical cross entropy (for loss) and adam (for optimizer). Figure 6 illustrates the summary of the Text CNN model before the fitting process has been applied.

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 50)	0	
embedding_2 (Embedding)	(None, 50, 100)	8662800	input_2[0][0]
conv1d_1 (Conv1D)	(None, 50, 64)	12864	embedding_2[0][0]
conv1d_2 (Conv1D)	(None, 50, 64)	19264	embedding_2[0][0]
conv1d_3 (Conv1D)	(None, 50, 64)	25664	embedding_2[0][0]
max_pooling1d_1 (MaxPooling1D)	(None, 16, 64)	0	conv1d_1[0][0]
max_pooling1d_2 (MaxPooling1D)	(None, 16, 64)	0	conv1d_2[0][0]
max_pooling1d_3 (MaxPooling1D)	(None, 16, 64)	0	conv1d_3[0][0]
dropout_3 (Dropout)	(None, 16, 64)	0	max_pooling1d_1[0][0]
dropout_4 (Dropout)	(None, 16, 64)	0	max_pooling1d_2[0][0]
dropout_5 (Dropout)	(None, 16, 64)	0	max_pooling1d_3[0][0]
concatenate_1 (Concatenate)	(None, 16, 192)	0	dropout_3[0][0] dropout_4[0][0] dropout_5[0][0]
flatten_1 (Flatten)	(None, 3072)	0	concatenate_1[0][0]
dropout_6 (Dropout)	(None, 3072)	0	flatten_1[0][0]
dense_3 (Dense)	(None, 30)	92190	dropout_6[0][0]

Total params: 8,812,782
Trainable params: 149,982
Non-trainable params: 8,662,800

Figure 6: Text CNN Summary

The final step is to fit the model to our training set variables X_train and Y_train. Regarding the parameters we have set the number of epochs initially to 20 and the batch size being 120. The batch size indicates the number of samples to be considered at each gradient update. Thus, for the first run we have placed it at 120 and will progress to 150 for the next run to see if there is any significant change to the accuracy and loss. Similarly, we shall observe if there is any change in the accuracy by the

increase in the number of epochs when set at 50. For the validation data we have set it to X_test and Y_test in order to test the variation of the loss at the end of each epoch. We have also made use of the 'Timeit' library to identify the total run time in nanoseconds for all the models. The results and discussions for all the models will be presented in the next section.

The next model to be implemented was the Bidirectional Gated Recurrent Units (BiGRU) with convolution. Initially, we call the SpatialDropout function to drop the excess number of feature maps present in the embedding matrix with the value set to 20%. Then we call the BiGRU function of keras which has a recurrent dropout of 10% which tells it to drop a fraction of the units for each recurrent unit and set the dimensionality of the output space as 128 units. We have also set the last output in the output sequence to be returned to the next unit as memory for the next unit. The bidirectional aspect comes from the reversal of the sequence being passed as the input to the current GRU layer. Then once the BiGRU layer has been built we then build the convolution layer with window size as half the size passed to the BiGRU layer as we will be processing only the changes from the previous i. e. BiGRU layer. And we have set the kernel window size as a fixed value of 3. Finally, we construct the Dense layer with the 'softmax' activation function. For the compilation of the model we have used the same parameters as we did in the previous model i. e. loss as categorical cross entropy and the optimizer as 'adam'. Figure 7 illustrates the summary of the BiGRU with a convolution model before fitting it to the dataset.

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 50)	0	
embedding_2 (Embedding)	(None, 50, 100)	8662800	input_3[0][0]
spatial_dropout1d_1 (SpatialDro	(None, 50, 100)	0	embedding_2[1][0]
bidirectional_1 (Bidirectional)	(None, 50, 256)	175872	spatial_dropout1d_1[0][0]
conv1d_4 (Conv1D)	(None, 48, 64)	49216	bidirectional_1[0][0]
global_average_pooling1d_1 (Glo	(None, 64)	0	conv1d_4[0][0]
global_max_pooling1d_1 (GlobalM	(None, 64)	0	conv1d_4[0][0]
concatenate_2 (Concatenate)	(None, 128)	0	global_average_pooling1d_1[0][0] global_max_pooling1d_1[0][0]
dense_4 (Dense)	(None, 30)	3870	concatenate_2[0][0]

Total params: 8,891,758
Trainable params: 228,958
Non-trainable params: 8,662,800

Figure 7: BiGRU with convolution Summary

The next step involves fitting the model to our training set and this involves the same condition as the previous model wherein we rotate the values for the batch size and the number of epochs and see the effect they have on the accuracy and loss of the model.

The final model that was implemented was the Long Short-term Memory (LSTM) with Attention. Before building the model and its associated layers we first have to build the Attention class so that we will be able to call it when our model is being built. For the Attention module we first build the weight and bias regularizer along with its associated constraints and the fact that it can acquire the feature maps i.e. the word vectors for our implementation when we pass the LSTM layer to it. We then build the masking layer and the build that performs the correlations and the handling of the additional dependencies present on the input dimension. Then we finally proceed to the LSTM layer. The parameters we are passing to the LSTM layer are more or less the same as the GRU layer in the previous model. This is no surprise as the LSTM and GRU are variations of the RNN model. For the LSTM layer we have specified a dimension size for output space as 300 due to the additional weights that have to be stored due to the addition of the attention module. The recurrent dropout is set to 25% so as to drop any units for the linear transformation of the input space and the return sequence type is also used to return the last sequence of the output sequence to the attention module so that it stores the correlation value vector. We then apply the LSTM layer to the embedding matrix and after applying the Attention module we use dropout to remove the problem of overfitting if it ever comes forth. We also apply BatchNormalization for maintaining the mean close to zero at each batch layer. We then build the Dense layer with the 'softmax' activation function. We then compile the model with the same parameters as the previous models. Figure 8 illustrates the summary of the LSTM model before it is fit to the training set.

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	(None, 50)	0
embedding_2 (Embedding)	(None, 50, 100)	8662800
lstm_2 (LSTM)	(None, 50, 300)	481200
dropout_7 (Dropout)	(None, 50, 300)	0
attention_2 (Attention)	(None, 300)	350
dense_5 (Dense)	(None, 256)	77056
dropout_8 (Dropout)	(None, 256)	0
batch_normalization_2 (Batch	(None, 256)	1024
dense_6 (Dense)	(None, 30)	7710

Total params: 9,230,140
Trainable params: 566,828
Non-trainable params: 8,663,312

Figure 8: LSTM with Attention Summary

For the fitting of the model we have applied the same parameters as in the previous models so as to maintain the consistency across the models so that their evaluation and comparison is thorough and accurate. For each model we have also created plots for the training and validation accuracy along with the plots for the training and validation loss. And we have derived the confusion matrices for each of the models and will be illustrated in the next section along with the discussion for them as well.

6. Experimental Results

The models that were implemented in the previous section were used for the training and testing of the news article dataset and have successfully classified the over 200, 000 instances with 6 features each present in the dataset. For

the purpose of corroboration of results, we have successfully plotted their confusion matrix for each model along with their respective accuracy and loss plots and finally computed their overall accuracy for each model along with their runtime for each of them. The entire program was run on a Windows 64-bit environment with the CPU as Intel i7-5500u with clock speed as 2.40 GHz and RAM at 8 GB and a Nvidia 850m GPU. For the Text CNN model, Figure 9 illustrates the confusion matrix that was obtained after fitting the model to the dataset.

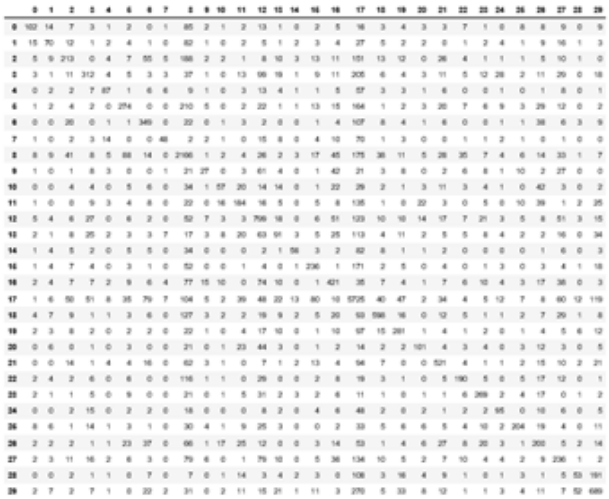

Figure 9: Confusion Matrix for Text CNN model

From the above figure it is readily observable that the classifier has predicted the majority of the samples of the test set into the right class, but still a large number of samples were misclassified especially for the eighth and seventeenth class. The optimal confusion matrix for this type of problem would be if the matrix would be a sparse matrix where the diagonal holds the majority of the samples. Figure 10 and 11 depicts the plot of the accuracy and loss for this model.

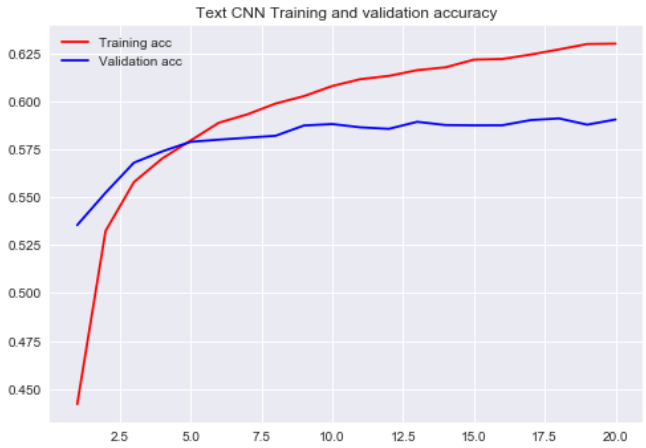

Figure 10: Text CNN accuracy

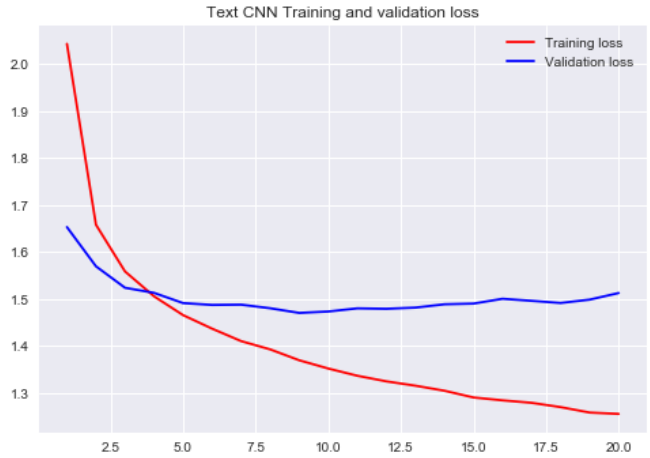

Figure 11: Text CNN loss

The plots for the accuracy and loss were obtained when the number of epochs was set to 20. But with regards to the batch size being either 120 or 150 it generated the similar plots for both criteria. From the plot of the accuracy we can observe that for the training accuracy we are seeing a steep increase in the accuracy as the number of epochs progress forth, but while training the accuracy starts out at a much higher point due to the fact that for training we have tried at three different kernel sizes and identified the optimal one when passing it to the convolution layer. Thus, the testing stage has learnt the right kernel size and put it forth to the layer when predicting the output. Table 1 contains the accuracy that was obtained for this model. With regards to the loss of this model in direct proportion to the accuracy we are seeing a sharp decline in the loss as the number of epochs progress for the training set and almost similarly for the test set as well. Figure 12 and 13 indicates the plots of having the number of epochs set to 50 and how it affects the learning rate. From the plots below we can see that the training accuracy starts at a higher point and we can safely assume that the error rates have been minimized and hence the model is starting at a higher level as it knows the optimal starting point. But the problem arises for the testing accuracy wherein we are getting no increase in the accuracy of the model and also the problem of overfitting has raised, thus the only way to counter this is to increase the dropout value and. The total runtime of the whole model is depicted in table 2.

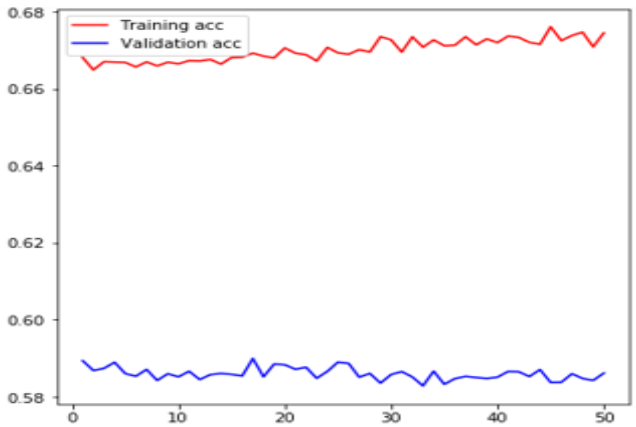

Figure 12: Text CNN accuracy with epochs as 50

Figure 13: Text CNN loss with epochs as 50

For the case of the BiGRU model with convolution, Figure 14 illustrates the confusion matrix that was extrapolated from the program. The confusion matrix when compared to the confusion matrix of the Text CNN model is slightly better when it comes to the number of samples that have been accurately classified.

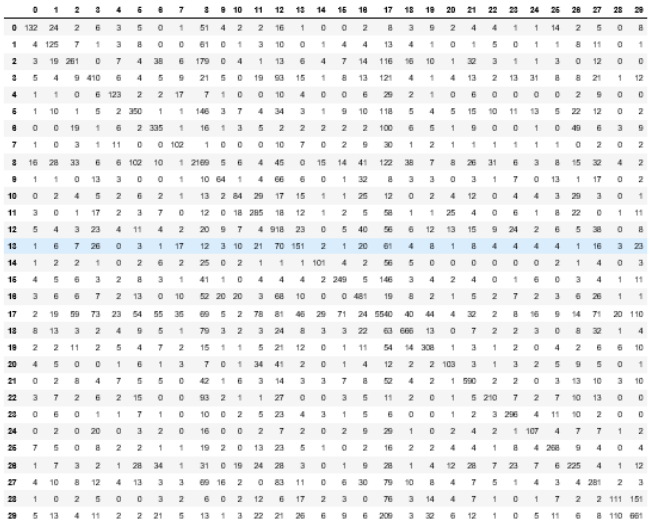

Figure 14: Confusion Matrix of BiGRU model with convolution

Just by observation and intuition we can assume that the results of the BiGRU model will be slightly better than the Text CNN model. Figure 15 and 16 illustrates the plots for the accuracy and loss for the BiGRU model with convolution when the number of epochs is 20 and the batch size is 150.

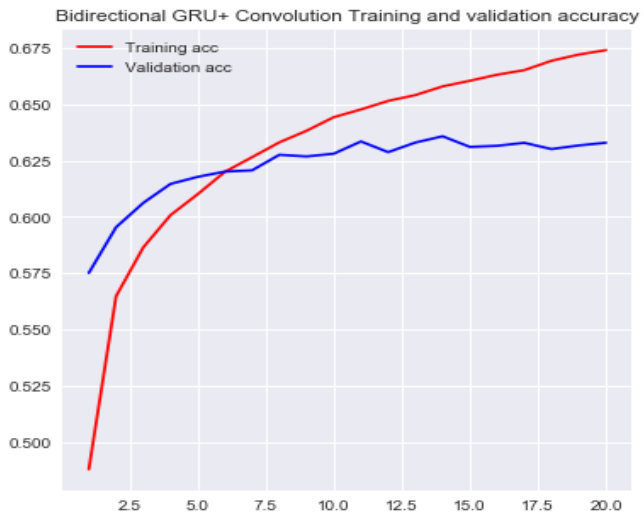

Figure 15: BiGRU with convolution accuracy

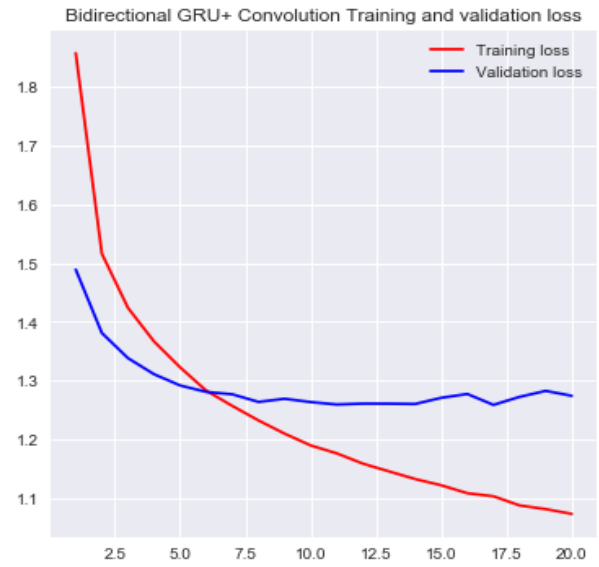

Figure 16: BiGRU with convolution loss

The plot for the accuracy and loss of the model shows the learning rate as being similar to the previous model. For the accuracy again we see a sharp increase in the accuracy from below 50% all the way to over 65% close to 67%. The testing accuracy is starting higher and this is due to the addition of the spatial dropout before calling the convolution layer. The benefit of the spatial dropout is that even though it is deleting the feature maps i. e. the word vectors in our case even before the BiGRU layer can build the correlation map for these word vectors, we are training the model initially such that we prevent co-adaptation of the vectors amongst its neighbors such that it learns as if there are no other feature maps existing currently. Thus, we are making the nodes learn and store memory initially and for a significant amount of time. The other reason that this model could present higher accuracy than the previous layer is the fact that we are using the keras GlobalMaxpooling and GlobalAveragepooling as the pooling layers consecutively. The benefit of these two methods is that they take the max and average vector over the whole steps dimension i. e. in one go rather than by reducing them at each iteration until it finally comes to the final reduction step. The accuracy and total time run will

be illustrated in tables 1 and 2. From the total time taken to run the whole model it is evident that the model which would have 50 epochs would be computationally intensive and could not be run on the current hardware platform but would run successfully on high-performance systems.

The final model that was implemented is the LSTM model with attention. Figure 17 illustrates the confusion matrix of the model.

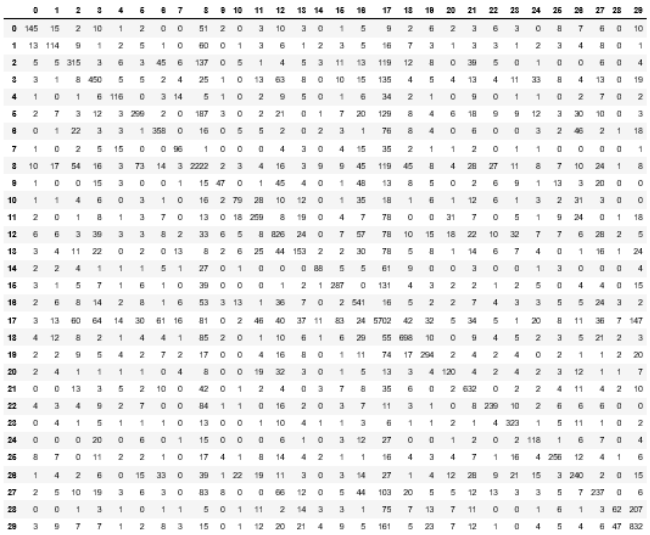

Figure 17: Confusion Matrix of LSTM with attention.

The confusion matrix for this model when compared to the confusion matrix of the other models is by far the best model when a high percentage of samples being classified accurately. The addition of the attention module has significantly improved the classification process and identified the optimal word vectors and their associated correlations that can be used to bring forth the right words and their associations so as to identify the important and correct words that could be used to predict the headline of the given description. Figure 18 and 19 illustrates the accuracy and loss of the model.

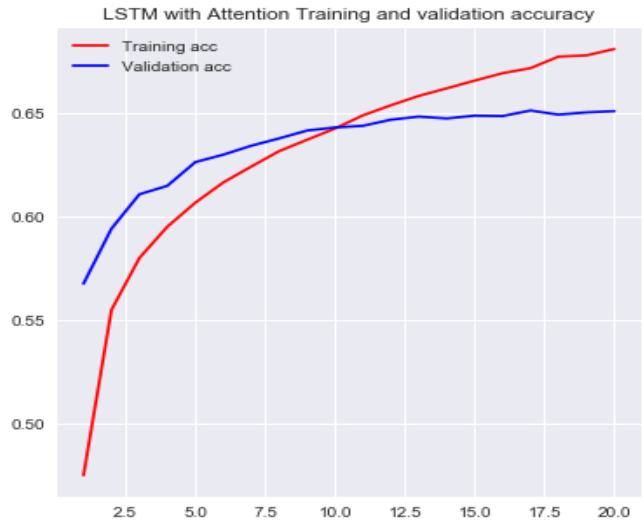

Figure 18: LSTM with attention accuracy

The plot of the accuracy above shows that there is no problem of overfitting that may occur as there isn't much overlap of the training and testing parts. Also, the we can derive from the graph that the accuracy is the highest for this model than all of them and is reaching approximately 65%. The attention module isn't the only exquisite thing about this model but also the LSTM itself, this LSTM [18] is holding the information until the next output sequence and then passing the specific part of the sequence as determined by the Attention module as the relevant information and this again is held by the LSTM in its memory and passed to the next node as the input thus having multiple layers of processing of each instance of the testing dataset. The loss plot as seen below in Figure 19 corroborates the same information.

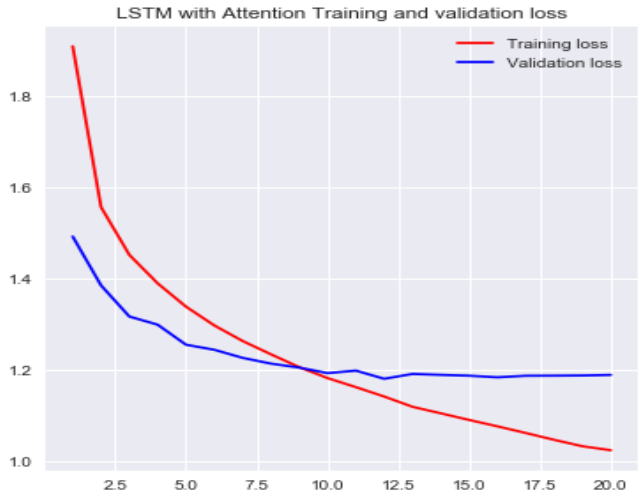

Figure 19: LSTM with attention loss

The loss coming forth in this model is again the least thus depicting that the error rate and number of misclassifications will be the least and this is due to above stated answer of why this is a good model. predict the headline of the given description. Figure 18 and 19 illustrates the accuracy and loss of the model.

Table 1: Models and their Accuracy Score

Training Model	Model Accuracy (in %)
Text CNN	59.055
BiGRU with convolution	63.294
LSTM with Attention	65.076

Table 2: Time for fitting the model

Training Model	Model Fitting Time (in nanoseconds)
Text CNN	186072.84
BiGRU with convolution	765789.99
LSTM with Attention	1217550.72

The above tables illustrate the implemented models and their respective accuracies and also the total run time for the fitting process. The reason to have gone with the fitting time rather than the testing time is because when running the model, the observation made was that the testing and training time was miniscule even on the current hardware and even if we were to increase the number of computations to be run then hypothetically it wouldn't change much. But the fitting as observed from running the program is that the fitting time takes an extreme amount of time and computation power. Thus, in

the future for increasing the scale of this problem and the number of computations to be run then having this information will be useful and we will be able to plot a time-series extrapolation as to how much time it might take based on the current hardware and what would happen if we were to switch to a high-performance system. That being said, from Table 2 we can observe that the LSTM model with attention is the most time-consuming and this is a result of the continuous backpropagation of the vectors to the input sequence. Unlike the other models in the LSTM model we are not applying dropout regularization multiple times and as a result it takes longer for the computations to run for each epoch. With regards to Table 1, as we noticed from the confusion matrices and the plots and as we predicted the LSTM model with Attention is the best performing model comparatively and as such can be used in the future for such similar datasets. The loss also for that model indicates that it can be modified further, and the error rate is amenable. In the next section we will discuss how these results could be used for further and future ideas and implementations.

7. Discussions and Conclusions

Upon completion of all the phases namely the preprocessing, model building, evaluation and assessment of the dataset and the results, we can now begin to discuss how exactly this implementation can be put forth for further review and how it will benefit the usage of this implementation for the real world. The significance of this paper was done to help the daily reader of online content and as an extrapolation to the physical reader of the newspaper as well. It does this by making the headlines of an article as appealing to the user as possible so that it not only benefits the writers of the article by gaining a larger and appropriately targeted audience but also benefits the reader as well benefits by gaining essential information as and when required. Let's explain how this implementation will help the readers with an example. Consider a natural or manmade calamity that has affected an area that may or may not be in the immediate surroundings of the reader, if the reader casually browses through the newspaper and doesn't come across any article about the disaster that interests him then he may not choose to further pursue it and in doing so is losing out on crucial information that could potentially benefit him and all because even though there may have been articles relating to the disaster, but the headlines or the following context may not have been properly summarized due to improper usage of grammar and context thus, leading to a huge impact and loss for many people. Thus, with this implementation we are providing a way for online worded content to be more appealing to the reader by providing at least one model that accurately goes through the worded context present in the description of an article and puts forth the words that have the highest number of correlations such that they can then be used as the tags for the headline of the said article. With regards to the creation of this paper and what it contributes to the field of Natural Language Processing, it gives us a way to extract a word with significant meaning from any passage or block of text. A good extension of this implementation would be the shifting over from the news articles section to the extraction of tags that can be used while searching for a video in a video platform such as YouTube, vine, Periscope, Instagram etc. Basically, on platforms such as YouTube where it is possible to generate the subtitles of the entire video using Speech processing, we can implement this paper onto the subtitles which is essentially a block of text such that we can generate suitable hashtags and other such markers and place them in the database along with the respective video. Thus, when any user wishes to view a video with those specific hashtags then we can obtain faster lookup times due to the presence of syntactically and topologically correct wordings based on the context as spoken on the video. For the future advancement of this paper we have already discussed which model is the most apt that being the LSTM model with Attention. The only further goal would be to improve the accuracy of the model such that we have deeper and much more correlations amongst the words that are getting generated from the text. The future implementation would be to look at how and what would be the effect of the Bidirectional nature on the LSTM model and also look at implementing spatial dropout initially before the LSTM layer is created. The addition of a spatial dropout scheme as used in the BiGRU model was found to improve the accuracy as seen in Figure 15. Thus, we can test these further and try to obtain the accuracy. Thus, for now the optimal model to apply on similar such datasets is the LSTM with Attention. There were no papers that have been implemented before for such an application especially using the methods which have been put forth in this paper as per the research that has been conducted; we decided that this paper would be a new step in this category. But with any application there is a drawback and the major drawback for this implementation is in its applied usage and that being majorly advertisement specifically for the online usage, and that being if the publishers are making a good turnover in their audience then they would naturally attract advertisers and thus we could see an influx of ads being posted along with the articles and this could dip the interests of the readers.

Appendix

In this paper for the sake of corroboration all the code along with the dataset and their associated images have been placed in a Google drive folder and made available to everyone.

The link is – https://drive.google.com/drive/folders/1_xzWgWTyKtj3_DYUjTFMEwLPJ5YL1YWf.

References

[1] Geert Brône & Seana Coulson (2010): Processing Deliberate Ambiguity in Newspaper Headlines: Double Grounding, Discourse Processes, 47: 3, 212-236.
[2] Petr Kroha & Ricardo Baeza-Yates. "A Case Study: News Classification Based on Term Frequency" in International Workshop on Database and expert

Systems applications, Denmark, 2005, https://doi.org/10.1109/DEXA.2005.6.
[3] Mazhar Iqbal Rana, Shehzad Khalid & Muhammad Usman Akbar. "News Classification Based on their Headlines: A Review" in 17th IEEE International Multi-topic Conference 2014, Karachi, https://doi.org/10.1109/INMIC.2014.7097339.
[4] Bin He, Yi Guan & Rui Dai, "Convolutional Gated Recurrent Units for Medical Relation Classification" July 2018 from https://www.researchgate.net/publication/326696595_Convolutional_Gated_Recurrent_Units_for_Medical_Relation_Classification
[5] Miwa, Makoto & Bansal, Mohit. (2016). End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures.1105-1116.10.18653/v1/P16-1105.
[6] Zhou Peng, Qi Zhenyu, Suncong Zheng, Jiaming Xu, Hongyun Bao & Bo Xu, "Text Classification Improved by Integrating Bidirectional LSTM with Two-dimensional Max Pooling", November 2016 in https://ui.adsabs.harvard.edu/#abs/2016arXiv161106639Z/.
[7] Gao, Shang & T Young, Michael & X Qiu, John & Yoon, Hong-Jun & B Christian, James & A Fearn, Paul & Tourassi, Georgia & Ramanathan, Arvind. (2017). Hierarchical attention networks for information extraction from cancer pathology reports. Journal of the American Medical Informatics Association: JAMIA.25.10.1093/jamia/ocx131.
[8] Gil Keren & Bjorn Schuller, "Convolutional RNN: An Enhanced Model for Extracting Features from Sequential Data", February 2016, in https://arxiv.org/abs/1602.05875
[9] Volkova, Svitlana & Shaffer, Kyle & Yea Jang, Jin & Hodas, Nathan. (2017). Separating Facts from Fiction: Linguistic Models to Classify Suspicious and Trusted News Posts on Twitter.647-653.10.18653/v1/P17-2102.
[10] Kaggle. (2018, Jun 21), "News Categorization Dataset" [Online]. Available from https://www.kaggle.com/rmisra/news-category-dataset/metadata.
[11] Jefferey Pennington, Richard Socher, Christopher D. Manning, "GloVe: Global Vectors for Word Representation" [Online] available in https://nlp.stanford.edu/pubs/glove.pdf.
[12] Yann Lecun & Yoshua Bengio, "Convolutional Networks for Images, Speech and Time-series" [Online] available in http://yann.lecun.com/exdb/publis/pdf/lecun-bengio-95a.pdf.
[13] Simon Kostadinov, (2017, Dec 16), "Understanding GRU networks" [Online] available at https://towardsdatascience. com/understanding-gru-networks-2ef37df6c9be
[14] Jungoung Chung, Caglar Gulcehre, KyungHyun Cho & Yoshua Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling".
[15] Simon Kyunghyun Cho, Bart Van Merrienboer, Caglar Gulchre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Youshua Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for statistical Machine Translation", June 2014, https://ui.adsabs.harvard.edu/#abs/2014arXiv1406.1078C/.
[16] Simon Jason Brownlee (2017, Oct 4), "How to Use Word Embedding Layers for Deep Learning with Keras" [Online] available from https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/.
[17] Colah's Blog (2015, Aug 27), "Understanding LSTM networks" [Online] available from http://colah.github.io/posts/2015-08-Understanding-LSTMs/.
[18] Shubham Khandelwal, Benjamin Lecouteux & Laurent Besacier, "Comparing GRU and LSTM for Automatic Speech Recognition", [Research Report] LIG.2016. <hal-01633254>.
[19] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, Bo Xu, "Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification", ACL 2016, https://doi.org/10.18653/v1%2FP16-2034.
[20] George E. Dahl, Tara N Sainath, Geoffrey Hinton, "Improving Deep Neural Networks for LVCSR using rectified linear units and dropout", IEEE International Conference on Acoustics, Speech and Signal Processing, May 2013, in https://doi.org/10.1109/ICASSP.2013.6639346