Amazon fake review detection using NLP

K.Srilatha[1] , K.Sai Chandu[2] , G.Rama Krishna[3]

[1]Department of Computer Science and Engineering, SR University, Warangal, Telangana, India.

[2]Department of Computer Science and Engineering, SR University, Warangal, Telangana, India.

[3]Department of Electrical & Electronics Engineering, SR University, Warangal, Telangana, India.

ABSTRACT

Reviews are becoming a more important source of information for consumers.Reviews are prevalent everywhere—on Amazon, Facebook, Google, Yelp, and more—and for good reason. According to research, 91% of consumers read online reviews either occasionally or regularly, and 84% of consumers trust online reviews as much as a personal suggestion. Because they build trust and increase transparency in the shopping experience, consumers are more likely to make purchases. 2022 shoppers rely on Amazon reviews from their 200 million most trusted peers rather than only word-of-mouth recommendations.Reviews are useful for everyone, not just customers looking for the perfect product. They are one of the best ways for a company to increase the conversion, credibility, and overall e-commerce presence of company's brand.Companies can use reviews to select how to create their goods or services, and consumers can use it to choose whether to purchase or use a product. A company is less likely to convince customers that their product is superior to the competition if it has a small number of reviews or if the reviews that a company have , are unfavourable .Unfortunately, certain people have attempted to create fake reviews in an effort to either boost the popularity of the product or to discredit it, misusing the significance of the review.However, fake reviews that paint a false impression of a product's quality limit the value of online reviews. Therefore, it's important to find fake reviews.Natural Language Processing (NLP) is a field of computer science and linguistics concerned with the interactions between computers and human (natural) languages ,we use Neural networks algorithm to detect fake reviews.

1 INTRODUCTION

In this era of the internet, customers can post their reviews or opinions on several websites. These reviews are helpful for the organizations and for future consumers, who get an idea about products or services before making a selection. In recent years, it has been observed that the number of customer reviews has increased significantly. Customer reviews affect the decision of potential buyers. In other words, when customers see reviews on social media, they determine whether to buy the product or reverse their purchasing decisions. Therefore, consumer reviews offer an invaluable service for individuals.

Positive reviews bring big financial gains, while negative reviews often exert a negative financial effect. Consequently, with customers becoming increasingly influential to the marketplace, there is a growing trend towards relying on customers' opinions to reshape businesses by enhancing products, services, and marketing. For example, when several customers who purchased a specific model of Acer laptop posted reviews complaining about the low display quality, the manufacturer was inspired to produce a higher-resolution version of the laptop.

2 DATASET

- The dataset is based on Amazon reviews from Kaggle with an already provided column "LABEL" which states if the a current review is fake or not.
- Dataset consists of 21000 rows and 9 columns which include DOC_ID,LABEL,RATING,VERIFIED_PURCHASE,PRODUCT_CATEGORY,PRODUCT_ID ,PRODUCT_TITLE,REVIEW_TITLE,REVIEW_TEXT.
- **DOC_ID :** It represents the document number.
- **LABEL :** This is the parameter which indicated if the relation between the "RATING" and the "REVIEW_TEXT" has a mismatch, giving the value 1 if that happen, meaning that it is a fake one, or not giving the value 0.
- **RATING :** The actual rating integer number from 1 to 5 with 1 be the lowest and 5 the highest.
- **PRODUCT_CATEGORY** : It represents the category of the product.
- **VERIFIED_PURCHASE:** A parameter which indicates of the user who comments is a verified on (with "Y") or is not (with "N"). Later on we are going to change those value with 1 and 0 respectively.
- REVIEW_TEXT: The comment description that is given as review for a product
- **PRODUCT_ID:**It is the unique id of the product.
- **PRODUCT_TITLE :** It is the name of the product with its characteristics.
- **REVIEW_TITLE** : It is the title of the review given by the customers about a product.

Sample dataset of Amazon product reviews

	DOC_ID	LABEL	RATING	VERIFIED_PURCHASE	PRODUCT_CATEGORY	PRODUCT_ID	PRODUCT_TITLE	REVIEW_TITLE	REVIEW_TEXT
0	1	_label1_	4	N	PC	B00008NG7N	Targus PAUK10U Ultra Mini USB Keypad, Black	useful	When least you think so, this product will save the day. Just keep it around just in case you ne...
1	2	_label1_	4	Y	Wireless	B00LH0Y3NM	Note 3 Battery : Stalion Strength Replacement 3200mAh Li-Ion Battery for Samsung Galaxy Note 3 [...	New era for batteries	Lithium batteries are something new introduced in the market there average developing cost is re...
2	3	_label1_	3	N	Baby	B000I5UZ1Q	Fisher-Price Papasan Cradle Swing, Starlight	doesn't swing very well.	I purchased this swing for my baby. She is 6 months now and has pretty much out grown it. It is ...
3	4	_label1_	4	N	Office Products	B003822IRA	Casio MS-80B Standard Function Desktop Calculator	Great computing!	I was looking for an inexpensive desk calcolatur and here it is. It works and does everything I ...
4	5	_label1_	4	N	Beauty	B00PWSAXAM	Shine Whitening - Zero Peroxide Teeth Whitening System - No Sensitivity	Only use twice a week	I only use it twice a week and the results are great. I have used other teeth whitening solution...

Figure 1: Visualizing the attributes of the dataset

3 DATA PREPROCESSING:

In this step we pre-process our data in order to bring them in a format that is more suitable for manipulation and it would be easier by our model.

- Firstly, we change the our categorical feature "LABEL" from "label_1" and "label_0" to 1 and 0 respectively.
- Secondly, we change our categorical feature "VERIFIED_PURCHASE" from "Y" and "N" to "0" and "1". This helps our model to work in a more efficient way and to "understand" better the "meaning" of this feature.
- Thirdly, we pre-processed the "REVIEW_TEXT" column by removing all the punction marks and to make all the word lower-case. By removing the punctuation we are able to avoid adding to our vocabulary words with excactly the same meaning. This will give more capacity to our embedding vocabulary.

Lower casing:

Converting a word to lower case (NLP -> nlp).
Words like Book and book mean the same but when not converted to the lower case those two are represented as two different words in the vector space model (resulting in more dimensions).It is one of the most common preprocessing steps where the text is converted into the same case preferably lower case. But it is not necessary to do this step every time you are working on an NLP problem as for some problems lower casing can lead to loss of information.If the text is in the same case, it is easy for a machine to interpret the words because the lower case and upper case are treated differently by the machine. for example, words like Ball and ball are treated differently by machine. So, we need to make the text in the same case and the most preferred case is a lower case to avoid such problems.

Tokenization:

Tokenization is the first step in any NLP pipeline. It has an important effect on the rest of your pipeline. A tokenizer breaks unstructured data and natural language text into chunks of informationthat can be considered as discrete elements. The token occurrences in a document can be used directly as a vector representing that document. This immediately turns an unstructured string (textdocument) into a numerical data structure suitable for machine learning. They can also be used directly by a computer to trigger useful actions and responses. Or they might be used in a machine learning pipeline as features that trigger more complex decisions or behavior.

Punctuation removal:

The punctuation removal process will help to treat each text equally. For example, the word data and data! are treated equally after the process of removal of punctuations. We need to take care of the text while removing the punctuation because the contraction words will not have any meaning after the punctuation removal process. Such as ‘don’t’ will convert to ‘dont’ or ‘don t’ depending upon what you set in the parameter.We also need to be extra careful while choosing the list of punctuations that we want to exclude from the data depending upon the use cases. As string.punctuation in python contains these symbols !"#$%&\'()*+,-./:;?@[\\]^_{|}~`

4 METHODOLOGY

The model that we created is based on the concept that we want to make predictions based on different features of the dataset and to check the resutls based on them. The feautures that seemed to perform better giving a conceptual reason for that are "REVIEW_TEXT", "RATING", "VERIFIED_PURCHASE" given that we want to predict the "LABEL" feauture.
So our model consists of 3 heads under the following architecture :

1.The head that processes the "REVIEW_TEXT" feature, on which we have applied the Word2Vec word embedding layer as a 100 dimesional vector. Then we use Dropout as a technique to avoid overfitting our model. Then we use a Bi-directional LSTM in order to process the sequences and to be able to extract the "useful" information from the individual sentence. Then the output of the Bi-directional LSTM is pass to a fully connected layer which we are going to use later on the concatination step of all our feauture models

2.The second head processes the "RATING" feature using a sequence of dense fully contected which is brought in the same dimensionsion as our first head and thrid layer later on, something that is useful for the concatination process of our heads.

3.The third layer follows the same logic as the second one. On this one we process the "VERIFIED_PURCHASE" feature again using a fully connected dense layer which is brought in the proper dimemsnions for the concatination procedure.

Then our model is compiled using the ADAM optimizer and as our loss function the binary crossentropy loss.We used BI-LSTM(Bidirectional Long Short Term Memory) and Word to Vector.

4.1 BI-LSTM(BIDIRECTIONAL LONG SHORT TERM MEMORY)

Bidirectional long-short term memory(bi-lstm) is the process of making any neural network to have the sequence information in both directions backwards (future to past) or forward(past to future).

In bidirectional, our input flows in two directions, making a bi-lstm different from the regular LSTM. With the regular LSTM, we can make input flow in one direction, either backwards or forward. However, in bi-directional, we can make the input flow in both directions to preserve the future and the past information.

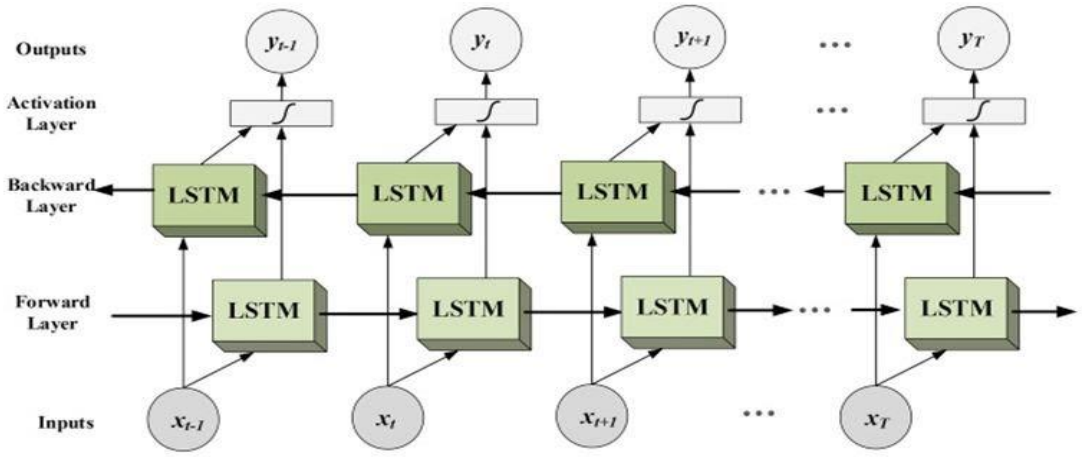

Figure 2 :BI LSTM Model

In the diagram, we can see the flow of information from backward and forward layers. BI-LSTM is usually employed where the sequence to sequence tasks are needed. This kind of network can be used in text classification, speech recognition and forecasting models.

This type of architecture has many advantages in real-world problems, especially in NLP. The main reason is that every component of an input sequence has information from both the past and present. For this reason, BiLSTM can produce a more meaningful output, combining LSTM layers from both directions.

LSTMs have three types of gates they are:

Input gates, forget gates, and output gates which controls the flow of information. The hidden layer output of LSTM includes the hidden state and the memory cell. Only the hidden state is passed into the output layer. The memory cell is entirely internal.

This challenge to address long-term information preservation and short-term input skipping in latent variable models has existed for such a long time. One of the earliest approaches to address this was the long short-term memory (LSTM) . It shares many of the properties of the GRU. Interestingly, LSTMs have a slightly more complex design than GRUs but predates GRUs by almost two decades.

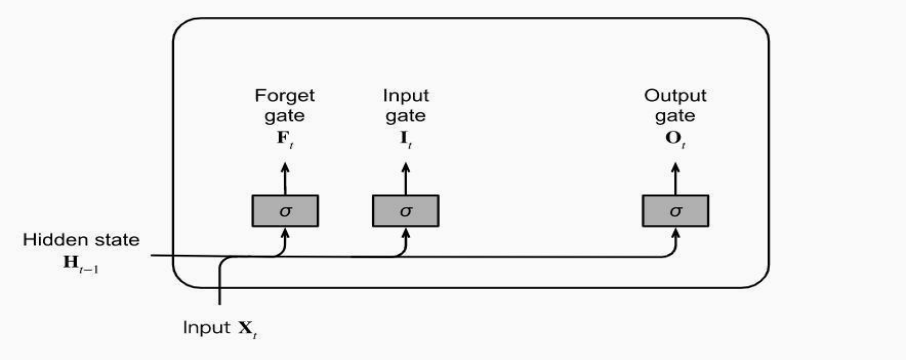

Figure 3: gated memory cell in lstm

4.2 WORD TO VECTOR

Word2Vec model is used for Word representations in Vector Space which is founded by Tomas Mikolov and a group of the research teams from Google in 2013. It is a neural network model that attempts to explain the word embeddings based on a text corpus.

These models work using context. This implies that to learn the embedding, it looks at nearby words; if a group of words is always found close to the same words, they will end up having similar embeddings. To label how words are similar or close to each other, we first fix the window size, which determines which nearby words we want to pick.

The General Flow of the Algorithm

Step-1: Initially, we will assign a vector of random numbers to each word in the corpus.

Step-2: Then, we will iterate through each word of the document and grab the vectors of the nearest n- words on either side of our target word, and concatenate all these vectors, and then forward propagate these concatenated vectors through a linear layer + softmax function, and try to predict what our target word was.

Step-3: In this step, we will compute the error between our estimate and the actual target word and then backpropagated the error and then modifies not only the weights of the linear layer but also the vectors or embeddings of our neighbor’s words.

Step-4: Finally, we will extract the weights from the hidden layer and by using these weights encode the meaning of words in the vocabulary.

Word2Vec model is not a single algorithm but is composed of the following two pre-processing modules or techniques:

Continuous Bag of Words (CBOW)

Skip-Gram.

Both of the mentioned models are basically shallow neural networks that map word(s) to the target variable which is also a word(s). These techniques learn the weights that act as word vector representations. Both these techniques can be used to implementing word embedding using word2vec.

Why Word2Vec technique is created?

As we know that most of the NLP systems treat words as atomic units. In existing systems with the same purpose as that of word2vec, there is a disadvantage that there is no notion of similarity between words. Also, those system works for small, simpler data and outperforms on because of only a few billions of data or less.

So, In order to train the system with a larger dataset with complex models, these techniques use a neural network architecture to train complex data models and outperform huge datasets with billions of words and with vocabulary having millions of words.

It helps to measure the quality of the resulting vector representations and works with similar words that tend to close with words that can have multiple degrees of similarity.

Syntactic Regularities: These regularities refer to grammatical sentence correction.

Semantic Regularities: These regularities refer to the meaning of the vocabulary symbols arranged in that structure.

The proposed technique was found that the similarity of word representations goes beyond syntactic regularities and works surprisingly well for algebraic operations of word vectors.

Continuous Bag of Words (CBOW)

The aim of the CBOW model is to predict a target word in its neighborhood, using all words. To predict the target word, this model uses the sum of the background vectors. For this, we use the pre-defined window size surrounding the target word to define the neighboring terms that are taken into account.

Advantages of CBOW:

1.Generally, it is supposed to perform superior to deterministic methods due to its probabilistic nature.

2.It does not need to have huge RAM requirements. So, it is low on memory.

Disadvantages of CBOW:

1.CBOW takes the average of the context of a word. For Example, consider the word apple that can be both a fruit and a company but CBOW takes an average of both the contexts and places it in between a cluster for fruits and companies.

2.If we want to train a CBOW model from scratch, then it can take forever if we not properly optimized it.

Skip-Gram

1.Given a word, the Skip-gram model predicts the context.

2.Skip–gram follows the same topology as CBOW. It just flips CBOW's architecture on its head. Therefore, the skip-gram model is the exact opposite of the CBOW model.

3.In this case, the target word is given as the input, the hidden layer remains the same, and the output layer of the neural network is replicated multiple times to accommodate the chosen number of context words.

Advantages of Skip-Gram Model

1.The Skip-gram model can capture two semantics for a single word. i.e two vector representations for the word Apple. One for the company and the other for the fruit.

2.Generally, Skip-gram with negative sub-sampling performs well then every other method.

5 FLOWCHART

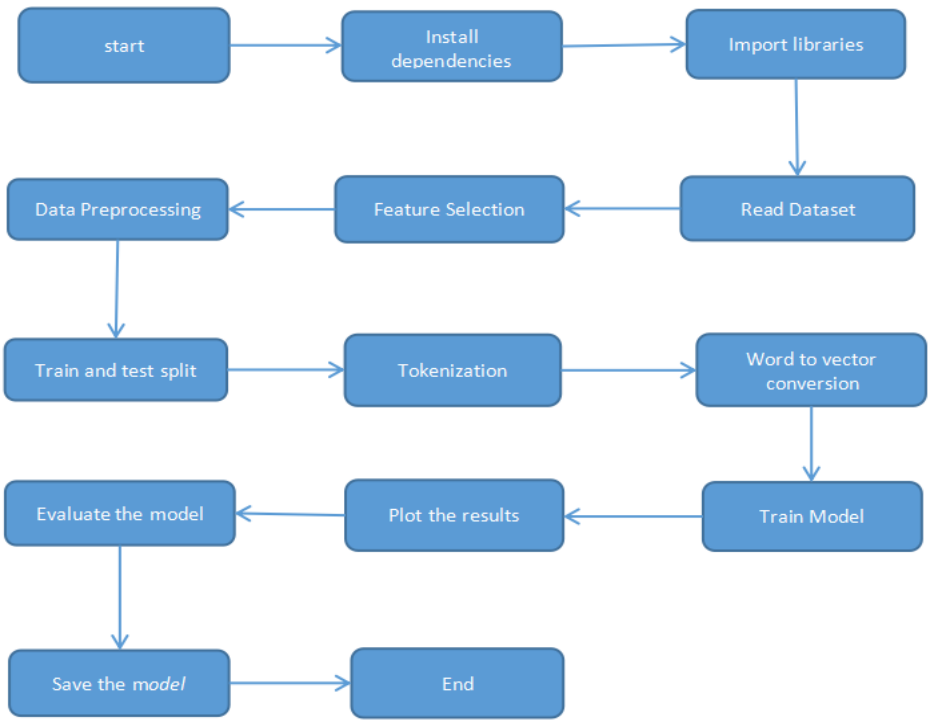

Figure 4- Flow chart of the technique**.**

6 ADAM OPTIMIZER

Adam may be thought of as a cross between RMSprop and Stochastic Gradient Descent with momentum. It scales the learning rate using squared gradients, similar to RMSprop, and it makes use of momentum by utilizing a moving average of the gradient rather than the gradient itself, similar to SGD with momentum.

Adam is an adaptive learning rate approach that calculates individual learning rates for various parameters. Adam employs estimations of the first and second moments of the gradient to change the learning rate for each weight of the neural network, thus the name adaptive moment estimation.N-th moment of a random variable is defined as the expected value of that variable to the power of n.

More formally:

$$m_n = E[X^n]$$

m — moment, X -random variable.

It might be tough to grasp that concept for the first time, so if you don't get it completely, keep reading; you'll eventually understand how algorithms function. Because it is normally assessed on a tiny random batch of data, the gradient of the cost function of a neural network can be regarded as a random variable. The first is mean, while the second is uncentered variance (i.e., we don't remove the mean when calculating variance). We'll look at how we utilize these values later; for now, we must decide how to obtain them. Adam calculates exponentially moving averages based on the gradient of a current mini-batch to estimate the moments.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Moving averages of gradient and squared gradient.

Where m and v are moving averages, g is the gradient on the current mini-batch, and betas are the algorithm's newly introduced hyper-parameters. They both have excellent default values of 0.9 and 0.999. These settings are rarely changed. The first iteration of moving averages starts with zeros in the vectors. Let's look at the predicted values of our moving averages to see how they relate to the instant described in the first equation. We want m and v to have the following attribute since they are estimates of first and second moments:

$$E[m_t] = E[g_t]$$
$$E[v_t] = E[g_t^2]$$

The estimators' anticipated values should match the parameter we're trying to estimate; in our instance, the parameter is also the expected value. If these properties were true, then we would have unbiased estimators. (See Ian Goodfellow's Deep Learning book, Chapter 5 on machine learning foundations, for further information on statistical features of alternative estimators.) We can now observe that this is not the case for our moving averages. The estimators are skewed towards zero since the averages are started with zeros. Let us demonstrate this for m. (the proof for v would be analogous). To demonstrate this, we must apply the formula to the very first gradient. Let's try unrolling a number of m values to see what pattern we'll use:

$$m_0 = 0$$
$$m_1 = \beta_1 m_0 + (1-\beta_1) g_1 = (1-\beta_1) g_1$$
$$m_2 = \beta_1 m_1 + (1-\beta_1) g_2 = \beta_1 (1-\beta_1) g_1 + (1-\beta_1) g_2$$
$$m_3 = \beta_1 m_2 + (1-\beta_1) g_3 = \beta_1^2 (1-\beta_1) g_1 + \beta_1 (1-\beta_1) g_2 + (1-\beta_1) g_3$$

As you can see, the higher the value of m, the less the first values of gradients contribute to the total value, as they are multiplied by smaller and smaller beta. We may revise the calculation for our moving average by capturing this pattern. Now let's look at the predicted value of m to see how it corresponds to the genuine first instance so that we can account for the difference:

$$E[m_t] = E[(1-\beta_1) \sum_{i=1}^{t} \beta_1^{t-i} g_i]$$
$$= E[g_i](1-\beta_1) \sum_{i=1}^{t} \beta_1^{t-i} + \zeta$$
$$= E[g_i](1-\beta_1^t) + \zeta$$

Bias correction for the first momentum estimator

To enlarge m in the first row, we utilize our modified moving average algorithm. Then we use g[t] to estimate g[i]. We may now remove it from the sum because it is no longer dependent on i. The mistake C appears in the formula as a result of the approximation. We just apply the formula for the sum of a finite geometric series in the last line. We should take two things away from that equation.

1. Our estimate is biased. This isn't only true for Adam; it's also true for algorithms that use moving averages (SGD with momentum, RMSprop, etc.).

2. It won't have much of an effect unless you start the training at the beginning, because the value beta to the power of t is rapidly approaching zero. Now we must update the estimator such that the expected value matches the desired value. Bias correction is the name given to this step. Our estimator's final formulae will be as follows:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Bias corrected estimators for the first and second moments

The only thing left to do is to use those moving averages to scale learning rate individually for each parameter. The way it’s done in Adam is very simple, to perform weight update we do the following:

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Where w is model weights, eta (look like the letter n) is the step size (it can depend on iteration).

And that's it, that's the update rule for Adam. For some people it can be easier to understand such concepts in code, so here's possible implementation of Adam in python:

```
for t in range(num_iterations):

g = compute_gradient(x, y)

m = beta_1 * m + (1 - beta_1) * g

v = beta_2 * v + (1 - beta_2) * np.power(g, 2)

m_hat = m / (1 - np.power(beta_1, t))

v_hat = v / (1 - np.power(beta_2, t))

w = w - step_size * m_hat / (np.sqrt(v_hat) + epsilon)
```

6.1 ACTIVATTION FUNCTIONS

6.1.1 RELU activation function :

ReLU stands for Rectified Linear Unit. ReLU has a derivative function and enables for backpropagation while being computationally efficient, even though it seems to be a linear function. The key caveat is that the ReLU function does not simultaneously stimulate all of the neurons. Only if the linear transformation output is less than 0 will the neurons be silenced.

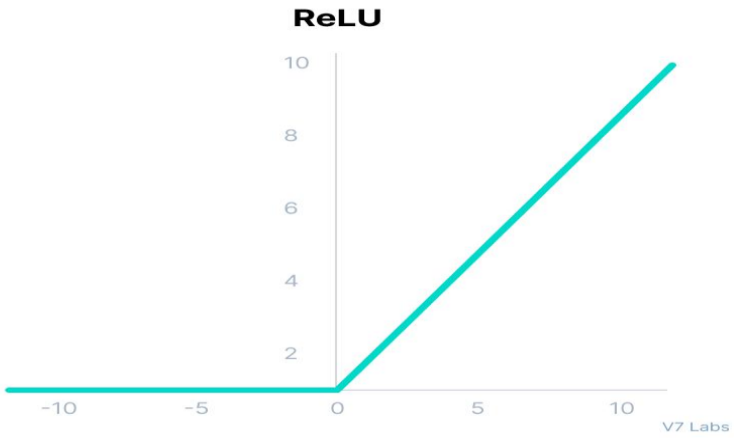

Figure 5 : *ReLU Activation Function*

Mathematically it can be represented as:

$$ReLU$$

$$f(x) = max\ (0, x)$$

The following are some of the benefits of utilizing ReLU as an activation function:

The ReLU function is significantly more computationally efficient than the sigmoid and tanh functions since only a small number of neurons are engaged. Due to its linear, non-saturating characteristic, ReLU speeds the convergence of gradient descent towards the global minimum of the loss function. The Dying ReLU issue is one of the function's limitations.

6.1.2. Sigmoid activation function

The Sigmoid Function curve looks like a S-shape.

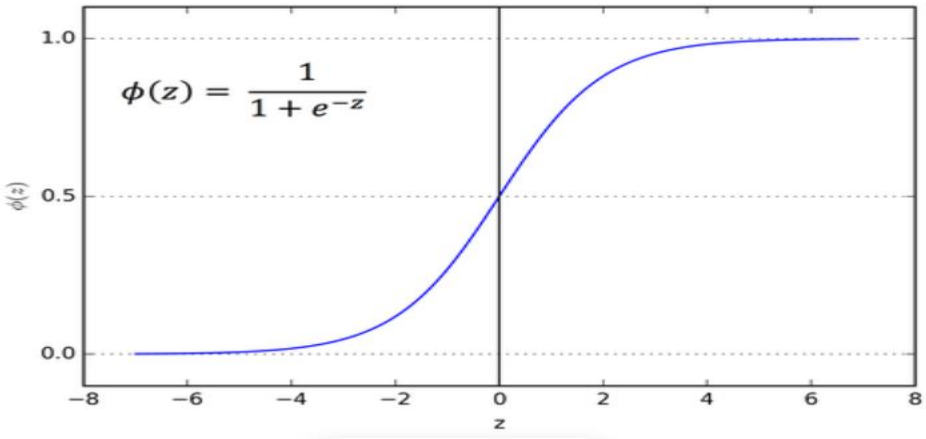

Figure 6 : sigmoid function

The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output.Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.

The function is differentiable.That means, we can find the slope of the sigmoid curve at any two points.

The function is monotonic but function’s derivative is not.

The logistic sigmoid function can cause a neural network to get stuck at the training time.

The softmax function is a more generalized logistic activation function which is used for multiclass classification.

7 RESULTS

This is our first attempt to get the model prediction results having a batch_size of 32 and we train from 20 epochs

We also add the stop early feature in order to avoid training our model while it does not improve.

```
[ ] stop_early_model = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
    history = model.fit([X_train_padded,train_rating,train_pursh], train_target, batch_size=32,epochs=20, validation_data=([X_val_padded,valid_rating,valid_pursh], valid_target),callbacks=[stop_early_model])

    Epoch 1/20
    525/525 [==============================] - 54s 95ms/step - loss: 0.5581 - acc: 0.7515 - val_loss: 0.4920 - val_acc: 0.7962
    Epoch 2/20
    525/525 [==============================] - 49s 93ms/step - loss: 0.4864 - acc: 0.7839 - val_loss: 0.4738 - val_acc: 0.8029
    Epoch 3/20
    525/525 [==============================] - 49s 93ms/step - loss: 0.4649 - acc: 0.7971 - val_loss: 0.4532 - val_acc: 0.8138
    Epoch 4/20
    525/525 [==============================] - 49s 93ms/step - loss: 0.4611 - acc: 0.7989 - val_loss: 0.4517 - val_acc: 0.8062
    Epoch 5/20
    525/525 [==============================] - 49s 93ms/step - loss: 0.4491 - acc: 0.8068 - val_loss: 0.4432 - val_acc: 0.8138
    Epoch 6/20
    525/525 [==============================] - 49s 93ms/step - loss: 0.4401 - acc: 0.8116 - val_loss: 0.4528 - val_acc: 0.8057
    Epoch 7/20
    525/525 [==============================] - 49s 93ms/step - loss: 0.4362 - acc: 0.8118 - val_loss: 0.4432 - val_acc: 0.8105
    Epoch 8/20
    525/525 [==============================] - 49s 94ms/step - loss: 0.4276 - acc: 0.8182 - val_loss: 0.4357 - val_acc: 0.8181
    Epoch 9/20
    525/525 [==============================] - 50s 95ms/step - loss: 0.4200 - acc: 0.8227 - val_loss: 0.4401 - val_acc: 0.8162
    Epoch 10/20
    525/525 [==============================] - 50s 95ms/step - loss: 0.4162 - acc: 0.8254 - val_loss: 0.4383 - val_acc: 0.8233
    Epoch 11/20
    525/525 [==============================] - 50s 95ms/step - loss: 0.4041 - acc: 0.8276 - val_loss: 0.4449 - val_acc: 0.8110
```

Figure-7 : Snippet of code showing epochs and accuracy

```
plt.figure(figsize=(16, 6))
plt.subplot(1, 2, 1)
plot_graphs(history, 'acc')
plt.subplot(1, 2, 2)
plot_graphs(history, 'loss')
```

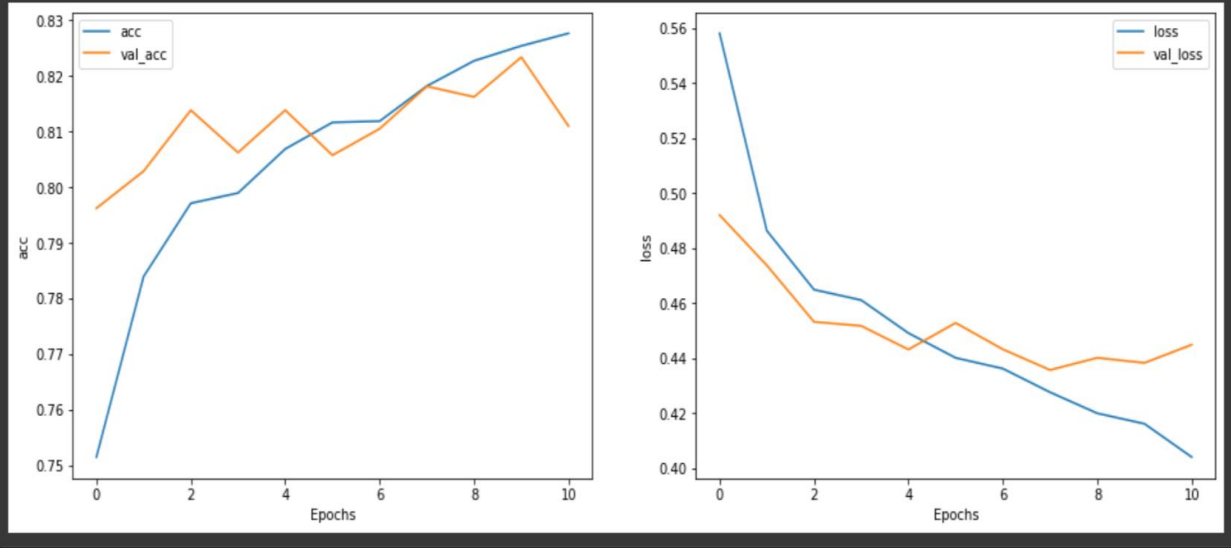

Figure-8:output graph of model with respect to accuracy and epocs.

```
[ ]  model.evaluate([X_test_padded,test_rating,test_pursh], test_target, verbose=2)

     66/66 - 2s - loss: 0.4391 - acc: 0.8190 - 2s/epoch - 28ms/step
     [0.4390980303287506, 0.8190476298332214]
```

Figure 9: Final result analysis of the model

8 CONCLUSION

The previous or existing fake review detection systems used traditional text based machine learning models. The performances are unstable when detecting fake reviews.So,we propose a deep learning model based Bi-LSTM(Bidirectional Long short term memory) method for fake review detection. The neural network deep learning algorithms that we used is Bidirectional LSTM(long- short term memory).

We managed to reach an validation accurasy of 81% by using word2vec embeddings and Bi directional LSTM model. We can be sure with great probability that based on this model architecture we got the maximun in terms of accurasy perfomance of validation results.

Our model is not overfitting with the data since from the graph results we can see that training accuracy and validation accuracy are realtively close and the losses have a decreasing flow.

9 REFERENCES

[1]P Devika and V Prashanthi, "RFID primarily based larceny Detection and vechile observance system victimisation cloud", International Journal of Innovative technology and Exploring Engineering(IJITEE), vol. 8, pp. 737-739, 2019.

[2]G. Anitha and P. Devika, "Secure Data Communication Using isecLEach Protocol In WSNs", International Electronic Journal of Pure and Applied Mathematics (Scopus indexed), pp. 1311-8080.

[3]E. I. Elmurngi and A. Gherbi, "Unfair Reviews Detection on Amazon Reviews using Sentiment Analysis with Supervised Learning Techniques", Journal of Computer Science, pp. 714-726, June 2018.

[4]J. C. S. Reis, A. Correia, A. Murai and K. Elissa, "Supervised Learning for Fake News Detection", IEEE Intelligent Systems, vol. 34, no. 2, pp. 76-81, May 2019.

[5]B. Liu and M. Hu, "Opinion Mining Sentiment Analysis and Opinion Spam Detection", [online] Available: https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#lexicon.

[6] M. N. Istiaq Ahsan , Tamzid Nahian , Abdullah All Kafi , Md. Ismail Hossain , Faisal Muhammad Shah “Review Spam Detection using Active Learning.” 978-1-5090-0996-1, pp.16, (2016).

[7] Michael C., et al. "Survey of review spam detection using machine learning techniques." Journal of Big Data 2.1, pp.9, (2015).

[8] Rajamohana S. P, Umamaheswari K., Dharani M., Vedackshya R., “Survey of review spam detection using machine learning techniques.” ,978-1-5090- 5778-8, pp.17 (2017).

[9] Mevada D. L., Daxini V., “An opinion spam analyzer for product Reviews using supervised machine Learning method.” pp.03, (2015).

[10]C. Hill, “10 Secrets to Uncovering which Online Reviews are Fake,” [Online]. Available: https://www.marketwatch.com/story/10-secrets-to-uncovering-which-online-reviewsare-fake-2018-09-21 [Accessed: March 2019].

[11]10. J. Novak, “List archive Emojis,” [Online]. Available: https://li.st/jesseno/positivenegative-and-neutral-emojis-6EGfnd2QhBsa3t6Gp0FRP9 [Accessed: June 2019].

[12]N. O’Brien, “Machine Learning for Detection of Fake News,” [Online]. Available: https://dspace.mit.edu/bitstream/handle/1721.1/119727/1078649610-MIT.pdf [Accessed: November 2018].