Deep Learning Models vs Traditional ML Models for Sentiment Analysis

Haja Amir Rahman (P2100803)
School of Computing (SoC)
Singapore Polytechnic, Singapore
amirrahman517804@gmail.com

***Abstract*—The study is focused on comparing the effectiveness and test accuracy when using a Simple-RNN model, LSTM model and GRU model using pre-trained Embedding Layers. Both LSTM and GRU are under the Recurrent Neural Network Family.**

I. INTRODUCTION

Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are models that are typically used on Sentiment Analysis Problems.

The recurrent neural network (RNN) is a model that is typically used on Sentiment Analysis Problems. It is a type of neural network that has a hidden layer between the input and output layers. The hidden layer is used to process sequential data such as words in sentences or words in paragraphs. The output of the hidden layer is then used as the input to the next layer. The following diagram shows how an RNN works:

The long short-term memory (LSTM) is a type of RNN that has two additional layers between the input and output layers. The first additional layer is called the gate and it is used to process sequential data such as words in sentences or words in paragraphs. The second additional layer is called the forget gate and it is used to forget previous states when processing new sequential data.

Traditional Machine Learning Machine learning refers to the study of computer systems that learn and adapt automatically from experience, without being explicitly programmed to do so. Machine learning algorithms use historical data as input to predict new output values. In this paper, for comparison, I am using ML models under Supervised Machine learning. In this type of machine learning, we supply algorithms with labelled training data and define the variables we want the algorithm to assess for correlations. Both input and output of the algorithm are always specified.

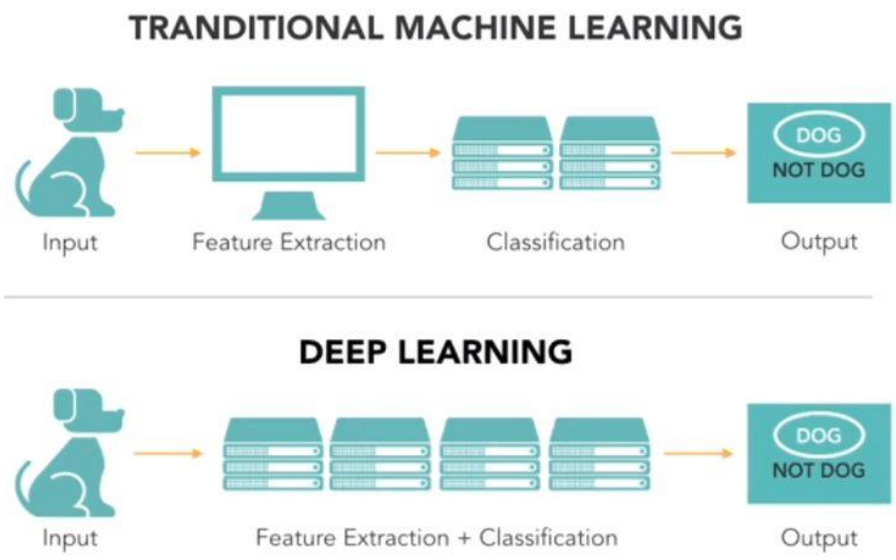


Fig 1. An illustration of the difference between Deep Learning and Traditional Machine Learning

II. METHADOLOGY

A. *Data Collection*

For this paper, I used a dataset containing reviews/comments from the Twitter App consisting of 162980 rows and 2 columns which made it very suitable to run our models. As mentioned in the title, I will be training three different Deep Learning Model Architectures RNN, LSTM and GRU on this dataset. In theory when comparing LSTM and GRU models, GRU supports gating and a hidden state to control the flow of information. To solve the problem that comes up in RNN, GRU uses two gates: the *update gate* and the *reset gate.* However, LSTM consists of three gates: the input gate, the forget gate, and the output gate. Unlike LSTM, GRU does not have an output gate and combines the input and the forget gate into a single update gate.

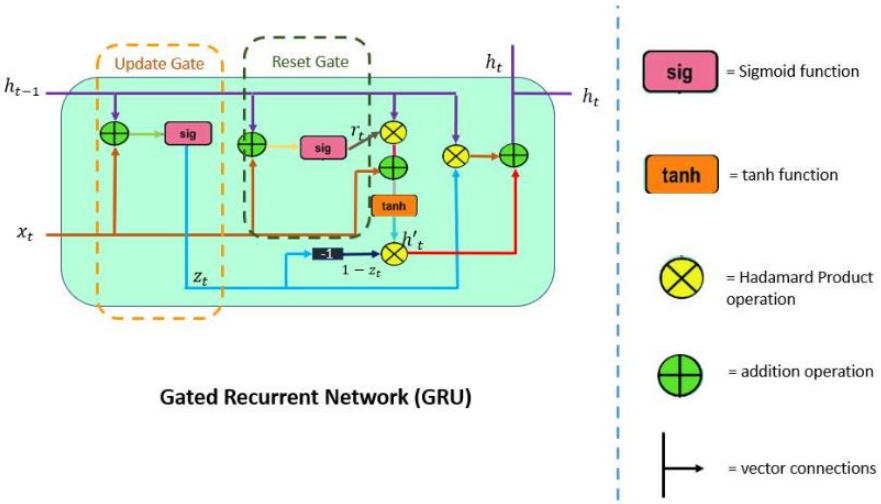


Figure 1: Gated Neural Network

B. *Dataset*

	clean_text	category
0	when modi promised "minimum government maximum...	-1.0
1	talk all the nonsense and continue all the dra...	0.0
2	what did just say vote for modi welcome bjp t...	1.0
3	asking his supporters prefix chowkidar their n...	1.0
4	answer who among these the most powerful world...	1.0
...	...	...
162975	why these 456 crores paid neerav modi not reco...	-1.0
162976	dear rss terrorist payal gawar what about modi...	-1.0
162977	did you cover her interaction forum where she ...	0.0
162978	there big project came into india modi dream p...	0.0
162979	have you ever listen about like gurukul where ...	1.0

Fig 2. Twitter Dataset information

From here we can see that the main data column which contains the reviews has been cleaned for us.

For the ‘category column’, we have 3 categories according to the Kaggle site:

- 0 Indicating it is a Neutral Tweet/Comment
- 1 Indicating a Positive Sentiment
- -1 Indicating a Negative Tweet/Comment

III. EXPLORATORY DATA ANALYSIS (EDA)

A. Column Data Types

Since this dataset is specifically meant for sentiment analysis, there are only 2 columns which are the reviews column and the category column that tell us whether the review is positive, negative, or neutral. When I found the dataset, the review column data was already cleaned so I did not need to perform much data cleaning.

B. Number of reviews per category

However, when I checked the count of reviews per category in the dataset, I found that the balance was not equal.

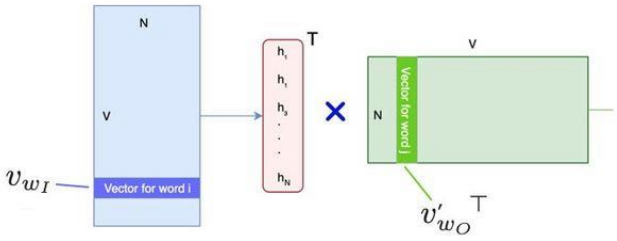
- The positive review category ‘1’ had 15830 counts of reviews
- The negative category ‘-1’ had 13142 counts of reviews
- The neutral category ‘0’ had 8277 counts of reviews.

Due to this, I performed some restricting to make the data count for each category equal and fair. Given that the lowest data count was 8277, I performed data sampling to make the positive review and negative review category’s value counts equal to 8277.

C. Using glove model for embedding layers

When I did some research regarding how to use embedding layers for my models, I came across a method called glove model loading for pretrained embedding layers. I decided to use this method because, embedding layer enables us to convert each word into a fixed length vector of defined size. The resultant vector is a dense one with having real values instead of just 0’s and 1’s. The fixed length of word vectors helps us to represent words in a better way along with reduced dimensions.

This way embedding layer works like a lookup table. The words are the keys in this table, while the dense word vectors are the values. There are 2 ways of using embedding layers – one is directly implementing the existent embedding layer available in Keras and the other is using pre-trained word embedding such as GloVe. To utilize the pre-trained word embeddings, I created some util functions to load the pretrained embedding layers.



$$p(w_O|w_I) = \frac{\exp\left(v'_{w_O}{}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left(v'_{w}{}^{\top} v_{w_I}\right)}$$

sum over all words in the vocabulary (normalization)

where v_w and v'_w are the “input” and “output” vector representations of w

Fig3. Word Embedding and GloVe for GRU model

- load_glove_model load the twitter embedding model we downloaded. This model is trained on 2 billion tweets, which contains 27 billion tokens, 1.2 million vocabs
- remove_stopwords remove the stop words in a sentence
- lemmatize perform lemmatization on a sentence
- sent_vectorizer convert a sentence into a vector using the glove_model. This function may be used if we want a different type of input to the RNNs.

Then I converted the reddit review text to sequence format that will be feed into RNNs.

```
# load the glove model
glove_model = load_glove_model("glove.twitter.27B.200d.txt", encoding='utf-8')
# number of vocab to keep
max_vocab = 18000
# length of sequence that will generate
max_len = 15

tokenizer = Tokenizer(num_words=max_vocab)
✓ 2m 3.6s

[INFO]Loading GloVe Model...
[INFO] Done...1193514 words loaded!
```

Fig4. Code for loading GloVe model

Next I prepared the word embeddings using the GloVe Model. The number of words is 44113 and the number of null word embeddings is 12999. The reason for using embedding layers in the model building function is because I did not one-hot encode the data as it is not a feasible embedding approach due to the large storage space required for the word vectors thus reducing model efficiency.

IV. MODEL BUILDING

Next I created a custom Model Building Function as my primary purpose is to compare the validation and test accuracy results of RNN, LSTM and GRU on the same Reddit dataset.

The reason for using a function and not directly building the model is to use an if-else statement for the models to chronologically run from RNN to LSTM to GRU. The function also uses another if-else statement to decide whether to add the pre-trained Embedding Layer to the model appropriately.

All three main layers of the three model have a parameter of 256, followed by a 2 Dense layers with ‘relu’ activation and the last Dense layer with ‘softmax’ activation. All the models are built using the ‘categorical_crossentropy’ loss function, ‘adam’ optimizer and the ‘accuracy’ metric.

Since these are Deep Learning Models, I set the parameters for all 3 as follows:

- Run at 30 epochs
- Included Early Stopping Callback Function monitoring for max validation accuracy with a patience of 10
- Batch size of 120
- ReduceLRonPlateau monitoring validation loss with a patience of 5, factor of 0.5 and minimum learning rate of ‘1e-6’

V. MODEL ARCHITECTURE AND EVALUATION

A. RNN Model

The **Simple-RNN model** performed the worst among the 3 models with a test accuracy of 74.61%. I also generated a classification report for precision, accuracy and f1 score results.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_2 (Embedding)      (None, 15, 200)           17515000

 simple_rnn_2 (SimpleRNN)     (None, 256)               116992

 dense_6 (Dense)              (None, 512)               131584

 dense_7 (Dense)              (None, 512)               262656

 dense_8 (Dense)              (None, 3)                 1539

=================================================================
Total params: 18,027,771
Trainable params: 512,771
Non-trainable params: 17,515,000
```

```
Best Acc Test Loss: 0.8793469667434692
Best Acc Test Accuracy: 74.61 %
Best Loss Test Loss: 0.6589160561561584
Best Loss Test Accuracy: 73.22999999999999 %
```

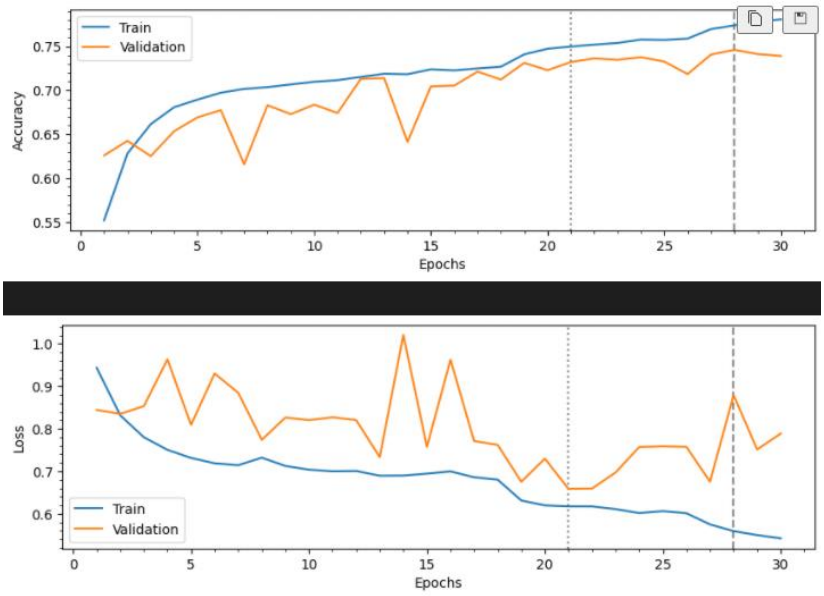


Fig4. RNN Model Architecture, evaluation for test accuracy and loss and graphical representation of accuracy and loss for training and validation respectively

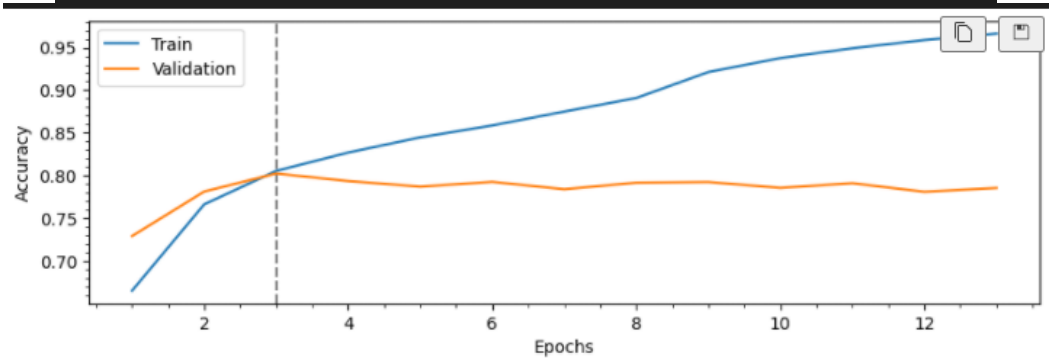
B. *LSTM Model*

The LSTM model performed slightly better than RNN with a test accuracy of 80.24%

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 15, 200)	17515000
lstm (LSTM)	(None, 256)	467968
dense_9 (Dense)	(None, 512)	131584
dense_10 (Dense)	(None, 512)	262656
dense_11 (Dense)	(None, 3)	1539

Total params: 18,378,747
Trainable params: 863,747
Non-trainable params: 17,515,000

```
Best Acc Test Loss: 0.5158692598342896
Best Acc Test Accuracy: 80.24 %
Best Loss Test Loss: 0.5158692598342896
Best Loss Test Accuracy: 80.24 %
```



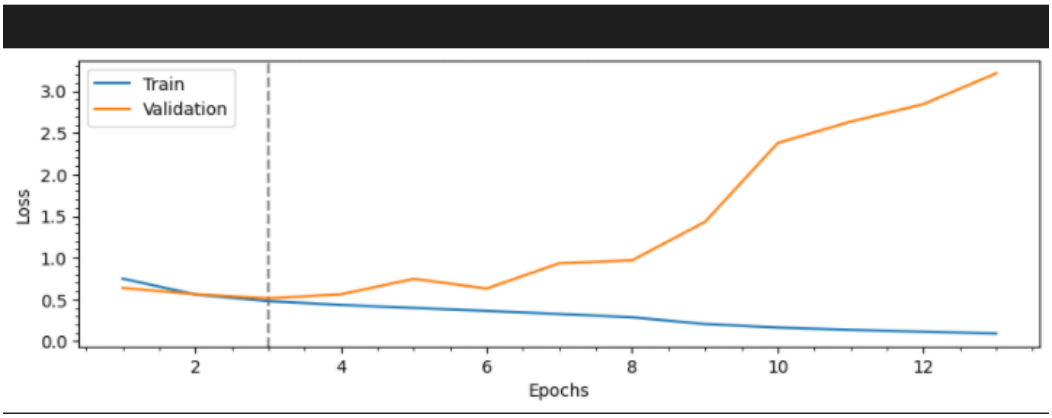


Fig 5. LSTM Model Architecture, Evaluation and graphical representation of test accuracy and loss for training and validation

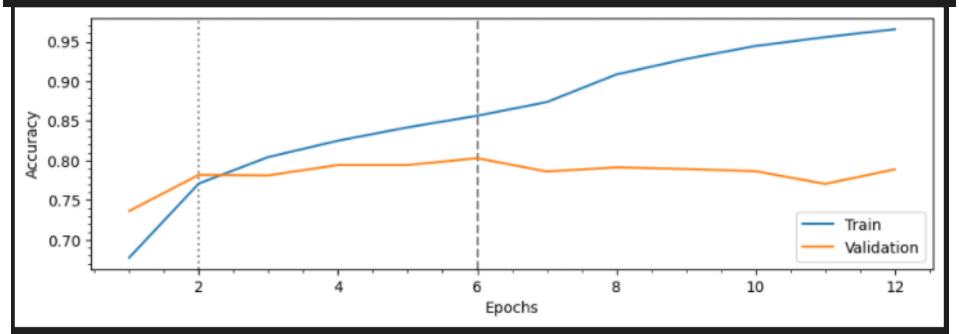
C. *GRU Model*

The GRU model performed the best of all the 3 models with a test accuracy of 80.28%

Layer (type)	Output Shape	Param #
embedding_4 (Embedding)	(None, 15, 200)	17515000
gru (GRU)	(None, 256)	351744
dense_12 (Dense)	(None, 512)	131584
dense_13 (Dense)	(None, 512)	262656
dense_14 (Dense)	(None, 3)	1539

Total params: 18,262,523
Trainable params: 747,523
Non-trainable params: 17,515,000

```
Best Acc Test Loss: 0.6783221364021301
Best Acc Test Accuracy: 80.28999999999999 %
Best Loss Test Loss: 0.5378003716468811
Best Loss Test Accuracy: 78.19 %
```



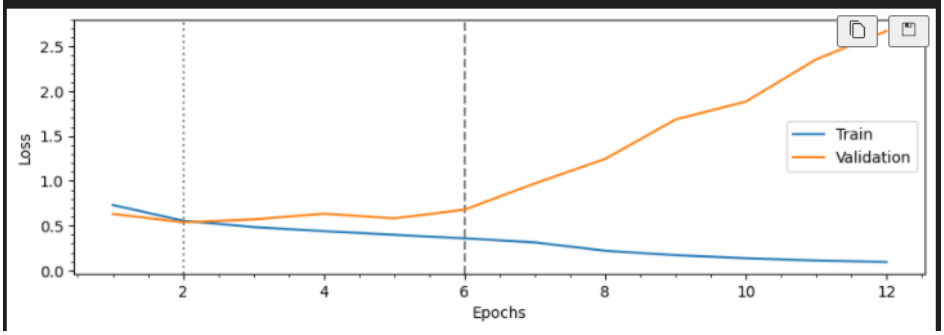


Fig 6. GRU model architecture, evaluation and graphical representaion of test accuracy and loss for training and validation

VI. USING A NON-DEEP LEARNING MODEL (TFDIF VECTORIZATION) AND TRADITIONAL ML MODELS

Since I only compared the model accuracy and efficiency of three deep learning models from the same family – RNN, I decided to use some traditional machine learning models to compare based on the same dataset used for the deep learning models.

I made use of TFIDF Vectorization to fit and transform the x_train data after running a fresh train-test-split.

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_features=10000, ngram_range=(1,2))
x_train_vec = vectorizer.fit_transform(x_train)
x_test_vec = vectorizer.transform(x_test)

x_train_vec.shape, x_test_vec.shape
```

```
((85224, 10000), (21306, 10000))
```

Fig 7. Using TFDIF Vectorization and limiting max features to 10000 for x_train and x_test

A. Random Forest Classifier

For the Random Forest Classifier Model, I did a simple model.fit, model.score and model.predict to find out how it did. Below are the classification report results and the confusion matrix respectively:

	precision	recall	f1-score	support
-1.0	0.79	0.78	0.79	7052
0.0	0.81	0.90	0.85	7121
1.0	0.87	0.78	0.82	7133
accuracy			0.82	21306
macro avg	0.82	0.82	0.82	21306
weighted avg	0.82	0.82	0.82	21306

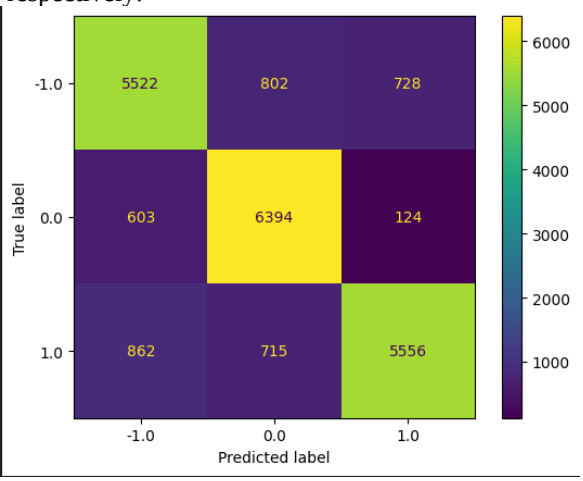


Fig 8. Random Forest Classifier Classification Report and Confusion Matrix Display

As seen from the classification report, this model has done better than all 3 deep learning models with a precision score of 82%.

A. Logistic Regression Model

For the Logistic Regression Model, I did the same as Random Forest and below are the results

	precision	recall	f1-score	support
-1.0	0.90	0.88	0.89	7052
0.0	0.89	0.97	0.93	7121
1.0	0.92	0.87	0.90	7133
accuracy			0.90	21306
macro avg	0.91	0.90	0.90	21306
weighted avg	0.91	0.90	0.90	21306

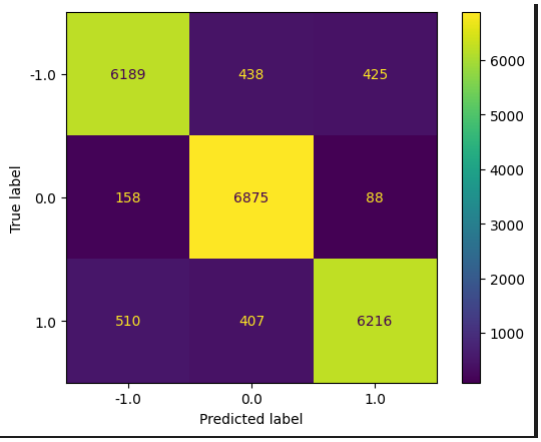


Fig 8.Logistic Regressor Model Classification Report and Confusion Matrix Display

From the above results, we can see that Logistic Regression performed the best out of all the models run with a test score of 91%

VII. MODEL RESULTS AND COMPARISON BETWEEN DL AND ML MODELS

So, to summarize what has been done so far, I compared the test accuracy of three different models – RNN, LSTM and GRU based on the same dataset, parameters, and layers.
Deep Learning Models:

- The GRU model performed the best with a test accuracy of 74.61%
- LSTM got a test accuracy score of 80.24%
- RNN got a test accuracy score of 80.289%

Traditional ML Models

- The Random Forest Classifier model got a precision score of 82%
- The Logistic Regression model got a precision score of 91%.

VIII. CONCLUSION

However, since the reddit dataset was already cleaned and labelled with nearly 40000 rows of data, the traditional machine learning models used for comparison did better after going through TFDIF Vectorization. When comparing the 3 Deep Learning Models, the GRU model performed best and when comparing Deep Learning Models with Traditional Machine Learning Models, the Logistic Regression Model did the best with a precision of 91%.

REFERENCES

[1] Chen, E. (2020) *Sentiment Analysis using SimpleRNN, LSTM and gru*, *Eric Chen's Blog*. Available at: https://haochen23.github.io/2020/01/nlp-rnn-sentiment.html#.Y3sIcXZByUk (Accessed: November 23, 2022).

[2] Yadav, T. (2018) *Glove.twitter.27b.200d.TXT*, *Kaggle*. Available at: https://www.kaggle.com/datasets/fullmetal26/glovetwitter27b100dtxt (Accessed: November 23, 2022).

[3] Saxena, S. (2021) *Understanding embedding layer in Keras*, *Medium*. Analytics Vidhya. Available at: https://medium.com/analytics-vidhya/understanding-embedding-layer-in-keras-bbe3ff1327ce (Accessed: November 23, 2022).

[4] Kapoor, A. (2020) *Deep Learning vs Machine Learning: A simple explanation*, *HackerNoon*. Available at: https://hackernoon.com/deep-learning-vs-machine-learning-a-simple-explanation-47405b3eef08 (Accessed: November 23, 2022).

[5] *Recurrent neural networks (RNN) with Keras : Tensorflow Core* (no date) *TensorFlow*. Available at: https://www.tensorflow.org/guide/keras/rnn (Accessed: November 23, 2022).

[6] Singhal, G. (2020) *Gaurav Singhal, Pluralsight*. Available at: https://www.pluralsight.com/guides/lstm-versus-gru-units-in-rnn (Accessed: November 25, 2022).

[7] Burns, E. (2021) *What is machine learning and why is it important?*, *SearchEnterpriseAI*. TechTarget. Available at: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML (Accessed: November 25, 2022).