ISSN: 2583-9888
DOI: https://doi.org/10.51219/JAIMLD/Sebastian-Ifeanyi-Obeta/01

URFORUM PUBLISHERS
CONNECTING WITH RESEARCH WORLD

Journal of Artificial Intelligence, Machine Learning and Data Science

https://urfpublishers.com/journal/artificial-intelligence

Vol: 1 & Iss: 1

Research Article

A Comparative study of Long Short-Term Memory and Gated Recurrent Unit

Sebastian Ifeanyi Obeta[1]*, Dr Enrico Grisan[1], Chinazor Vivian Kalu[2]

*[1]Department of Data Science, London South Bank University, UK

[2]Department of Artificial Intelligence with Business Startegy, Aston University

Citation: Obeta, S. I., Grisan, E., & Kalu, C. V. (2023). A Comparative study of Long Short-Term Memory and Gated Recurrent Unit. *J Artif Intell Mach Learn & Data Sci*, 1(1), 1-9. DOI: https://doi.org/10.51219/JAIMLD/Sebastian-Ifeanyi-Obeta/01

***Corresponding author:** Sebastian Ifeanyi Obeta, Data Scientist and NLP Engineer, Cambridge University, UK. E-mail: sebastian.obeta@gmail.com

Received: 23 December, 2022; **Accepted:** 17 January, 2023; **Published:** 20 January, 2023

A B S T R A C T

In natural language processing (NLP), the assumption that a neural network has an independent state among data samples does not apply to sequential data. Hence recurrent neural networks (RNN) have played a key role in sequential dependency in natural language processing with the key features of providing context to the processing and tackling vanishing gradient. Long Short-Term Memory Units (LSTM) are RNN blocks that can retain essential information even if it is far from the current point of analysis (extended memory). Still, they also have a fading effect that favours closer information (short memory). Despite this, they still need to remember vital details far from their current position, which goes against the intent of the extended memory effect. Gated Recurrent Units (GRU) have shown excellent results in sequential data and were introduced to overcome the limitations of LSTM by using two vectors (update gate and reset gate) to decide what information passes to the output. They can also train to keep data for a long time without it washing it through time or removing information irrelevant to the prediction. Some scholars suggest that gated recurrent units could be a suitable replacement for long short term memory.

This comparative study presented the performance difference between LSTM and GRU and their bi-directional-based neural networks when they face the task of classifying text data. The evaluation of Gated Recurrent Unit (GRU) versus Long Short Term Memory (LSTM) and their bi-directional versions were carried out on a task of a website based on its content. Our analysis showed that a gated recurrent unit (GRU) is a good substitute for long short-term memory for text data classification. The bi-directional GRU outperformed the bi-direction LSTM. We recommend a gated recurrent unit as a better alternative to Long Short Term Memory on text data classification.

Keywords: Machine learning, Recurrent neural network, Gated recurrent unit, Long short-term memory, Deep learning.

Introduction

The effect of the internet and the web has generated a large volume of data that increases geometrically from different sources. The data generated are mostly in amorphous form and, to help gather insights from them, it creates an enormous need for data pre-processing and management. One such task comes in the form of text classification or clustering.

Text classification is a significant task in natural language processing (NLP), and it usually involves dealing with unstructured data originating from different sources, such as tweets and newsletters. Websites are a specific source of unstructured data with varying contents as they can be related to newspapers, conferences, sports events, and people's profiles. The text classification task is a concept in NLP that requires the classification of text documents or text data into groups. If we consider the spontaneous growth of the web, it is evident that it has a plethora of content which suggests the need to have organised and filtered information.

Text classification tasks are then linked to sentiment analysis, topic modelling, web information retrieval, and spam filter. There are four classification models in text classification which are:

- Rule-based (rules drafted or defined by a human),
- Probability concept,

- Learning-based (ensemble model, logistics regression, state vector mechanism),
- Deep learning-based (Recurrent Neural Networks, Convolutional Neural Networks, Hybrid Deep Neural Networks). (Zhang et al., 2022) (Charu & Cheng Xiang, 2012).

Traditional text classification tasks comprise three steps. The first is feature engineering which is dominant in Natural Language processing tasks and belongs to a family of models tagged bags of words. The bags of word tasks are word count, term frequency-inverse document frequency (TF-IDF), and N-grams. Although they help extract features from the text, semantics, structure, sequence and context of words are lost due to the inherent nature of the model of being just a bag of words. However, the state of art in feature engineering has improved by including part of speech tags and noun phrases. (David, 1992).

The second form is the feature selection that involves the wrapper and wrapper approach. The strategy for the wrapper is called try and test, which takes time and involves high computational power. In contrast, the filter method picks the informative feature by filtering with some metric measure. The third and final step form is the machine learning algorithms.

The traditional concepts create a gap and a motivation to engage a more sophisticated model that can capture information and represent features as vectors of a word, known as embedding. Data scarcity is another limitation of a bag of words, meaning they must detect sufficient data in a corpus during modelling. The introduction of the deep neural network gave insight into tackling the data paucity and more neural models that improve learning. (Christain, et al., ٢٠٠٣). However, many researchers have proposed using easy data augmentation techniques with four steps (Xu et al., ٢٠٢١). The neural network has proven and performed better in many NLP tasks.

Classification text data from websites is exciting and challenging, as each website differs in many ways, either by structure or content. To some extent, one can find almost everything online, which translates into the availability of many unstructured text data whose full information content can be exploited only upon analysis, understanding and tagging.

RNN is a type of neural network that feds the input of a current step with the output of the previous step. A typical traditional neural network (NN) sees the input and work as independent. A word prediction task requires referencing the previous word, and RNN tries to solve it with the help of a hidden layer (Casanueva et al., 2020). The critical feature of RNN is the hidden layer that considers some sequence information. Recurrent neural network (RNN) has shown outstanding outcomes in many tasks, particularly in machine translation when the output and input of a model are of variable length. (Graves, 2012). The milestone achieved in the machine translation task is not with the vanilla RNN but with LSTM, GRU, and their sophisticated hidden layer. (Sepp & Jurgen, 1997) (Cho et al., 2014).

To further improve RNNs performance and fully exploit the context provided by textual sequence, some additional units specifically targeted at sequential information have been proposed:

LSTM is a subset of RNN designed to recognise the pattern in sequence by taking time and sequence into account, which is the difference between RNN and LSTM from all other neural networks. Research showed that RNN and LSTM are powerful and valuable types of NN; however, in language tasks, attention mechanisms, transformers, and memory networks tend to surpass them. In the Mid-90s, German researchers proposed LSTM as a network that can preserve error and back propagated through time. They maintained a constant error and allowed the recurrent net to learn over time steps. This lead to an open channel which is a challenge to machine learning and AI since algorithms are frequently faced with environments where reward signals are delayed.

GRU is a subset of RNN that uses connection through a sequence of nodes to carry out a machine-learning task that deals with memory and clustering. It helps to adjust the weight of the input in a neural network and thereby solves the gradient problem common with recurrent neural networks.

However, the question of the most appropriate memory unit for text classification and prediction remains unsolved. This work uses GRU and LSTM as memory units for the same deep network and classifies 53,440 website-based content data. Both one-directional and bi-directional variants will be evaluated.

Literature Review

Different research has addressed text classification and proposed techniques and solutions. Classification in the context of text data is gaining momentum and is one of the complex tasks in natural language processing. The text mining task covers classification, topic modelling, spam classification, sentiment analysis, and document classification. Compared with machine learning, deep learning has significantly achieved the natural language processing task by passing through several stages and document vectorisation. Different deep learning techniques have been deployed in various tasks of NLP. A recurrent neural network has proved successful, and deep learning recursive models for sentiment analysis (Casanueva et al., 2020) and language modelling (Richard et al., 2016). (Wu et al., 2022) Speech recognition and sentence production from poems and visuals (Ankit et al., 2006) (Micro et al., 2018).

The ground-breaking work done by Bengio et al. (1994) on word embedding using a neural network model on a language model with each word's preceding contexts (Tomas et al., 2010) inspired Mikolov et al. (2013) to propose two novel architectures for the continuous bag of words and skip-the n-gram approach, which has a vector representation by computing successive terms from a large dataset.

The outcome was measured from a similar word task compared to the previously performed techniques based on different neural networks. The method was used in local and linear contexts and was further transformed into dependency-based word embeddings and global vectors (GIoVe).

Some things could have been improved with the linear and local context. The word embedding in a semantic-based tackled the linear limitations with an introduction of syntactic contexts obtained from the dependency parser. In contrast, the GloVe approach addressed the local limitation by looking at the word-to-word statistics.

Many discoveries have been made with recurrent neural networks for language modelling. One such is the arbitrary dependency as proposed by (Pengfei et al., 2016). Moreover, the ability of a deep learning framework to train models creates a whole way to generate relationships between labels and features and hence makes prediction accurate. The success of RNN was seen in the automatic extraction of features in a document using

bi-directional LSTM and attention mechanism. (Srividhya & Anitha, 2010).

LSTM and GRU are algorithms in the recurrent neural network. It is imperative to discuss the need for RNN architecture and the shortcoming of neural networks in sequence data modelling. There has been a generally accepted view about neural networks, which is the independent state among data samples, and this assumption cannot be applied when sequential data is involved. Such sequential data includes speech, language, time series, and media data (Akash, 2022) because they display a level of dependency within them across time.

The neural network sees the data samples individually, which results in the model losing the result that would have been seen when being treated as sequential information. To account for sequential dependency will be equivalent to joining a fixed number of consecutive data points and seeing them as one data point. (Alex et al., 2013). The standard technique in representing data sequences defines it as a sequence of words. The language modelling task with RNN takes words one by one, and the output is the probability of the predicted words.

(Tobias & Matthias, 2017) Applied RNN to model the sequence of customer interaction in a webshop. They used the frequency of customers to predict the probability of a customer placing an order within seven days. The input data was a one-hot encoder vector which stood for the previous or past action of customer orders and was compared to logistic regression with the help of feature engineering for a space of 5 months. Both achieved a similar result and showed by visualisation how consumer behaviour changes over time. The static nature of feedforward neural networks over a dynamic classifier necessitated the RNN architecture. To extend the feedforward neural network to a dynamic classifier, they fed the signals from the previous timestamp into the network. They were called recurrent neural networks. (Paul, 1990).

The modified version of feedforward in NN was the capability to reference the previous time frame because the feedback was either vanishing or exploding. The RNN is usually limited to 10-time steps, which is also a significant limitation. Long Short-Term Memory (LSTM) was introduced to learn as long as 1,000 time steps depending on the complexity of the network (Sepp & Jurgen, 1997) by enforcing a constant error flow through a continuous merry-go-round approach in a unit.

Although LSTM achieved remarkable success in long-lasting unit memory compared to RNNs, it still needs to remember pieces of information that seem far from the current point or position. It became a big challenge, and more pronounced when (Guozheng et al., 2018) attempted to use LSTM for document-level sentiment classification. Many researchers took up the task of modifying LSTM to store information, and the outcome gave rise to different models of LSTM. (Ke et al., 2016) Proposed adding external memory to LSTM, but the result could have been better concerning time because of the vast memory matrix. (Duyu et al., 2016) Considered the bi-directional LSTM with attention model on document sentiment and had a 95% success.

GRU was demonstrated as an extension of LSTM by (Junyoung et al., 2014). By eliminating the output gate, which is in charge of writing the contents of the memory cell to a more significant net at each time step, his design reduces the complexity of LSTM. The result of Junyoung et al. on a music dataset showed that the GRU made faster progress in actual CPU time, although the effect is inconclusive. They suggested that the choice of framework depends on the dataset and corresponding task.

RNN

RNN architecture has a significant drawback which (Bengio et al., 1994) highlighted the difficulty in training an RNN because of its vanishing gradient descent. Researchers tried to solve this by proposing two main approaches to reduce the negative impact. The first approach uses a better learning algorithm rather than stochastic gradient descent. (Bengio et al., 1994), (Pascanu, et al., 2013), (Van Gompel et al., 2022). The second suggestion is to use an activation function which is more than just serving as an activation function with affine transformation and a gating unit. The second suggestion gave birth to another activation function or a recurrent unit called long short-term memory (LSTM) (Sepp & Jurgen, 1997). Shortly after, another recurrent (gated recurrent unit GRU) was proposed by (Cho et al., 2014). The two frameworks (LSTM and GRU) have been shown to perform well in tasks requiring long-time dependencies. (Sutskever, et al., 2014) (Dzmitry, et al., 2014).2.2 Limitations of Multilayer Perceptron Layer.

MLP cannot learn from sequence information from data. Simple expressed by saying:

- it loses its sequence information hence,
- Context is lost.

RNN Architecture

The word recurrent in RNN suggests that it performs the same task for every sequence by being dependent on the previous step. It has two key features:

- A hidden state that is distributed in nature and allows information storage of the past and,
- Allows the hidden states to update themselves in complicated and nonlinear dynamics.

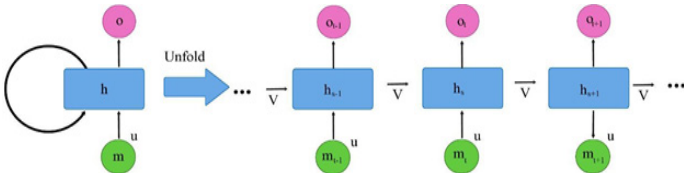


Figure 2.1: RNN Architecture (Pascanu, et al., 2013).

Given a sequence of value $m = (m_1, m_2, \ldots\ldots, m_t)$ RNN will update its hidden state h_s as:

$$h_s = 0, s\ is\ unit\ of\ time = 0\ \varphi,\ \left(h_{s-1}, m_{t,}\right) \quad (1)$$

The implementation of Eq 1 can take this form:

$$h_s = g\ (w_{hh}\ h_{s-1} + w_{xh} m_{t,} + b_1) \quad (2)$$

Where g is the sigmoid function, w_{xh} is input to the hidden layer weight matrix, h_s is a hidden state vector with time, w_{hh} is hidden to hidden weight matrix and b_1 is the bias. In text data, a traditional neural network looks at the data points in isolation; however, RNN considers the word's sequence. If we have a task to identify a name in a given sentence with a concept called name entity, we need the knowledge of other words surrounding it to carry out the task.

Consider the diagram in figure 2.2, where t_1, t_2 are time steps, with input x_1, output y_1, weight matrix of the input w_{xy}, weight matrix of the hidden layer w_{hh}, and weight matrix that controls the information from the hidden state to output (w_{hy}).

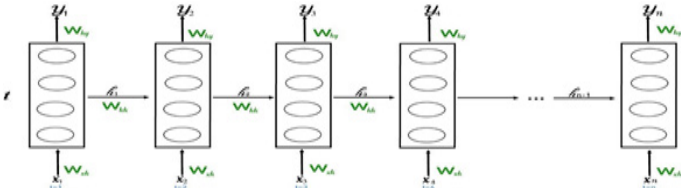


Figure 2.2: Training RNN Model (Pengfei, et al., 2016).

A time step t_1 with a word x_1 as input will compute some activation to give output y_1 which is determines if the word is a person's name or not. The time step t_2 will take word x_2 and give output y_2. The difference between MLP and RNN is the time step t_2 in making a prediction. It will not only consider the input x_2 but also take the hidden state of the previous word h_1 because it has the information that has been computed from the previous one. Similar at each time step when making a prediction, it will not take only the input but the hidden state of the previous time step that has the information of all words processed. It will use all the information to make predictions. The first step in forwarding propagation in RNN is:

Calculate the current hidden state:

$h_s = g\,(w_{hh} * h_{s-1} + w_{xh} m_{t,} + b_1)$ where g is tanh/ReLU (3)

Calculate the current output $y_t = g\,(w_{hy} * h_s + b_1)$ where g is sigmoid/softmax (4)

RNN architecture has a significant drawback which (Bengio et al., 1994) highlighted the difficulty in training an RNN because of its vanishing gradient descent. Researchers tried to solve this by proposing two main approaches to reduce the negative impact. The first approach uses a better learning algorithm rather than stochastic gradient descent. (Bengio et al., 1994), (Pascanu, et al., 2013). The second suggestion is to use an activation function which is more than just serving as an activation function with affine transformation and a gating unit. The second suggestion gave birth to another activation function or a recurrent unit called long short-term memory (LSTM) (Sepp & Jurgen, 1997). Shortly after, another recurrent (gated recurrent unit GRU) was proposed by (Cho et al., 2014). The two frameworks (LSTM and GRU) have performed well in tasks requiring long-time dependencies. (Sutskever, et al., 2014) (Dzmitry, et al., 2014).

Need for LSTM-and GRU

The input weight (w_{xh}), Output weight (w_{hy}), and hidden state (w_{hh}) weights are randomly initiated during the forward propagation state and at backpropagation state, we compute the loss of the network by updating the weight of the matrix for better prediction. The gradients are calculated for the backpropagation of the loss value concerning the weight. The gradient descent algorithm updates the weight so as to have a minimal loss as shown in fig 2.3.

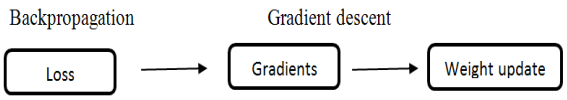


Figure 2.3: Backpropagation process.

There are two main challenges with RNN, the first is the vanishing gradient caused by the network's inability to learn long-term dependency, and the second is the exploding gradient. Here, the gradient is too large and tends to crash the model during training due to numerical overflow. To overcome the vanishing gradient, initialize the weight so the gradient will not vanish. However, it is hard to achieve such; hence LSTM and GRU are designed to accomplish that. The exploding gradients can be mitigated by truncating the backpropagation process or using the techniques of gradient clipping.

The known methods to train an RNN are backpropagation through Time (BPTT) and Real-Time Recurrent Learning (RTRL). (David, et al., 1986) (Paul, 1990), though backpropagation is the standard method. The way the two are weighted differently makes a difference. The identical backpropagation procedure is followed by BPTT, except the chain rule is used repeatedly instead of only once.

The objective function will depend on activating the hidden layers and their influence within the time step. The vanishing gradient occurs during training in RNN (backpropagation), predominantly seen if the training involves long input sequences or many layers. The error gotten during the training updates the weight of the network towards the right with its magnitude. Mathematically this is achieved with the chain rule. It will pass through matrix multiplication either by shrinking or blowing up exponentially for long sequences. It then means that having a too-small gradient will result in the weight needing to be updated effectively, whereas large gradients can cause instability. There are gates in LSTM and GRU designed to solve this with their additive components. They keep the existing hidden state and add new content to it. This allows the error gradients to go through the backpropagation process without vanishing or exploding too quickly.

Long Short-Term Memory

Various strategies were proposed to tackle the issue of vanishing gradient descent in RNN. Some such attempts simulated error propagation (Bengio et al., 1994) made in the 1990s. Bengio et al. introduced time delays (Kelvin & Alex, 1990), sequence compression (Micheal, 1991) and the LSTM architecture. (Sepp & Jurgen, 1997). The first development of LSTM was to reduce the vanishing gradient rate and make RNN effective for long-term memory tasks. The architecture of LSTM has four gates. The input, forget gate, output, and memory cell. The gate determines what should be written to or read from the memory cell, where the information is kept. The gates are the medium of transport of information based on the weights. However, some weights, such as the input and the hidden state, are adjusted during the learning rate. Like a filter on a waterway that keeps contaminants from going through, these gates aim to filter out any undesired input or information.

However, the gates are trained to accurately detect what is helpful from what is not.

$$\text{Forget gate } (f_S) = \sigma(w * x_t + u * h_{s-1} + b) \quad (5)$$

$$\text{Input gate } (I_s) = \sigma(w * x_t + u * h_{s-1} + b) \quad (6)$$

$$\text{Output gate } (o_s) = \sigma(w * x_t + u * h_{s-1} +) \quad (7)$$

$$\text{Memory cell } (c_s) = f_S * c_{s-1} + I_s * \tanh(w * x_t + u * h_{s-1} + b) \quad (8)$$

$$o_s = f\,(w * h_s + b) \quad (9)$$

Where σ is an activation function is (sigmoid), x_t is the input vector with time, h_{s-1} is the hidden with respect to time, w is the hidden input weight matrix and is the bias.

The input gate detects what information is to be stored in the long-term memory but can work with the information from the current input short-term memory from the previous time step and will filter the information that are not necessary. This is achieved mathematically with two layers. The First layer selects the information to pass to the next layer bypassing it with short-term memory through a sigmoid activation function. This layer is trained through backpropagation, where the weight is updated

and trained to detect helpful information as seen in Eq 6. The second layer in the input gate takes the current input and passes it through a tanh function to regulate the network.

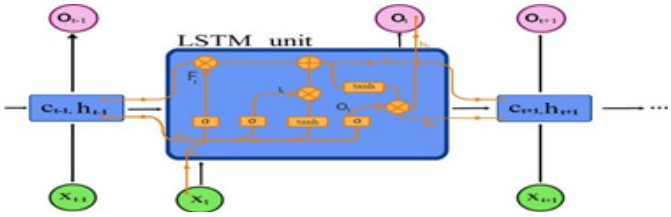


Figure 2.4: LSTM Framework (Pascanu, et al., 2013).

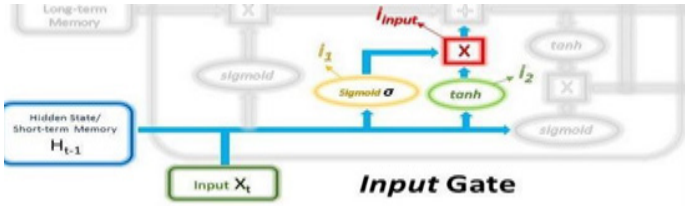


Figure 2.5: Input gate Flow (Gabriel, 2019).

The forget gate acts as a filter but takes the product of the forget vector seen from the current input and incoming short-term memory. To generate the forget vector, the long-term memory and the current input are passed through a sigmoid function similar to the input layer, which determines what information to dis-regard and pass through to the next cell, as seen in Eq 5.

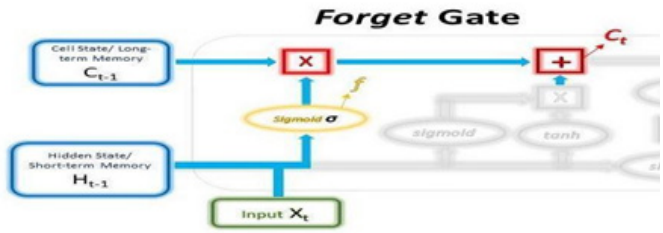


Figure 2.6: Forget gate flow (Gabriel, 2019).

The output does most of the work. Here it takes the current input and previous short-term memory state and adds to an entirely new long-term memory to produce a recent short-term memory that will be passed through to the next step. The previous short-term memory and current input will pass through the sigmoid function with different weights to create a third filter.

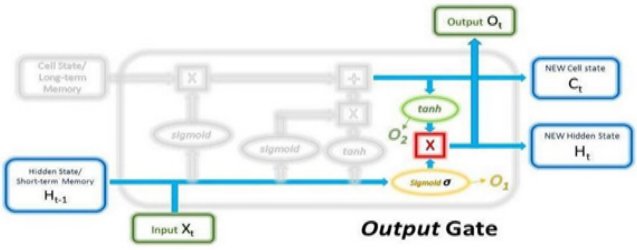


Figure 2.7: Output gate flow (Gabriel, 2019)

A Bi-directional LSTM is made up of two RNNs. The first function of the RNN is responsible for the sequence in regular order or forward movement, and the second is in charge of the reading in a reverse way. The training of Bi-directional LSTM does not capture the encoding in any particular information; instead, the encoded vectors are fed further into the network layer to a point a prediction is made, and a loss is incurred. This helps it extract all relevant information for the task at hand from the input sequence.

Gated Recurrent Unit (GRU)

GRU is a branch of LSTM, introduced by (Cho, et al., 2014). It is a replica of LSTM but without an output gate which allows the flow of content from the memory cell to the large net at each time step. It is said to be fast for training purposes and its internal makeup is not complex but requires few computations to update the hidden layer. GRU is made up two gates:

Reset gate ( ) and update gate (. The reset gate is responsible for the combo of input with the memory cell and the update gate looks after and defines how much of the memory cell to store. These gates act like filters but are highly trained.

$$k_s = \sigma(w * x_t + u * h_{s-1}) \qquad (10)$$

The difference between LSTM and GRU is that the former contains two different states that pass between the hidden and the cell state. While the later has only one hidden state transferred between time steps. The single hidden state has the ability to hold both the long term memory and the short term at the same time with the help of the gated mechanism that hidden and input information goes through.

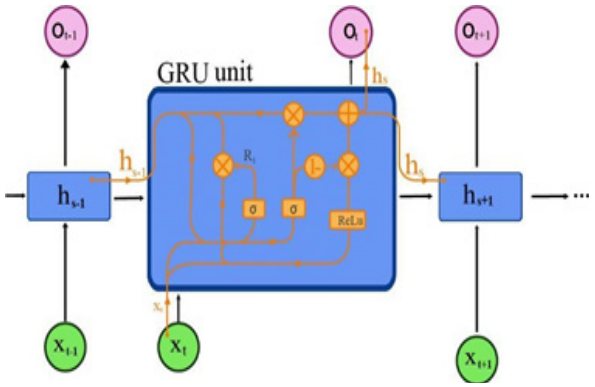


Figure 2.8: GRU Framework (Charu, 2018).

The reset gate is generated using the hidden state from the previous time step and inputting data at the current time step. Representing it mathematically, we take the product of the previous hidden state and current input and sum them with their respective weight before passing it through a sigmoid function which will squash them into 0 and 1, as seen in LSTM gates. When the entire model is trained with backward propagation, the weights will be updated, and by then, we will learn to keep only important information.

The Update gate is just like the reset, and the hidden gate computes the previous and hidden state and the current hidden state. The reset and update gate is obtained with the same formula but with a slight difference in the weight multiplier between the hidden and input state are unique. This means that the final output or vector for each gate will be different and makes them serve specific functions.

Methodology/Data Analysis

This research was built with Keras because it is simple and easy to use, supports the hardware of GPUs for parallel Matrix Multiplications, supports python programming language, and has a large community that contributes to its development. We looked at the embedding layer, model, and evaluation and saved the trained model. The experiment was performed on a google collab connected to Python 3 with a google compute engine power of RAM 3.14 GB/12.72 GB and a disk of 31.73 GB/68.40 GB. Two models were developed for each architecture. The experiments first evaluated the performance of LSTM, BiLSTM, BiGRU, and GRU on 53 447 websites' content by using the sequence information to categorise the website based on its

content. The experiment was done with python packages such as pandas and NumPy, used for data manipulation, and NLTK package for text processing. The dataset was first cleaned by removing the noise associated with text data; label encoding was carried out, validation splitting by 70% and 30%, padding and finally, the model build.

Data description

The dataset contains 53,447 rows meaning there are 53,447 websites; each website has associated tags with text content, some with multiple tags with suitable varieties such as websites of news, publication, profile (LinkedIn), conferences, forums, clinical trials, and thesis.

Data Preprocessing

For text data, preprocessing is an essential task in NLP. The purpose of data preprocessing is to remove noise and incomplete and complete data. Data from different sources may have other formats. 80% of data or text mining tasks are done at this stage. (Sebastian & Joddelle, 2020), for this research, the preprocessing done in this task is cleaning, Label encoding, validation split, and padding.

Cleaning

Text cleaning aims to remove unwanted or useless information present in text data. Examples of such words are stop words, commonly called common entities. Examples are hashtags, punctuations and numbers. We have slang sometimes seen in text data and human errors (spelling and grammar errors). There are different reasons why text data contain noise. A few reasons are human errors, which could result from data errors, software digitalization accuracy, machine translation, or web scraping. It becomes necessary for text cleaning to be done. Text cleaning, therefore, is a systematic removal of text noise that helps reduce text data's dimension; algorithms learn better and fast, removing repetitive information and helping to focus on the entity present in a text.

The following cleaning task was done on the dataset:

- Convert all text to lowercase for simplicity and uniformity when working and training the data.
- Removal of links and hyperlinks. Generally, links or hyperlinks play no role in text data analysis or any form of texting mining.
- Special character, punctuation, number removal, and text inside {}, (). Removing special characters in text data is essential to avoid concatenating between words and making them unavailable.
- Removal of stop words. This eliminates unimportant words in giving intelligent patterns or information to the task at hand. We used the NLTK package to remove stop words in this task.

Label Encoding.

The main challenge with text data is converting text or categorical data to numerical data and making the algorithm accept it and make sense of it. The neural network makes use of numerical values as input. There are many ways to carry out such a task; I used a label encoder for this research, and label encoding was only done for the target column. Label encoder is a package from the SciKit-learn library in python. The dataset contains different tags, which are our target variables. Hence, we labelled the encoder to convert it to numerical data for our deep learning model to learn.

Validating Splitting

This research was split into two. Training and testing. The training is where the model learns and tests to validate our model prediction on the subset. The split was 7:3 where the evaluation is on 30% of the data. In this task, I used the text split function from SciKit-learn where y is the label encoder target.

Padding

Sequence data have a common context of having different lengths. Padding in this task is to ensure all of my sequences has consistent length. To find out the length to be used for this research, I checked for the sequence distribution of the dataset. I plot a number of words in each sentence as shown in fig 3.

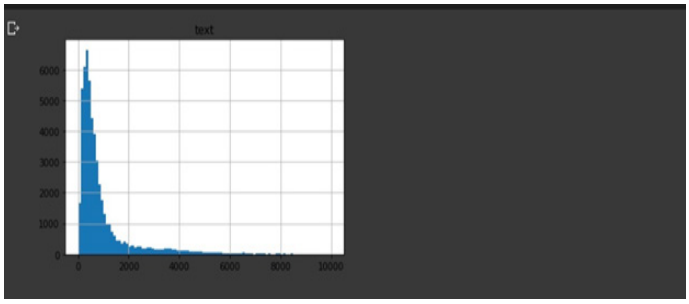


Figure 3: Words distribution.

The histogram shows that most of the sequence falls within length 500. Hence, 500 was picked as the maximum length for 500 for the padding. All sequences less than 100 will have zero values added. Before padding the sequence, the text will be tokenised with Keras and further label encoded with text to sequence the individual words in the text. Text to sequence is another form of dense vector representation of words which is a class approach called embedding. Embedding is a transformed way of using a bag of words techniques to represent each word in a corpus with a sparse vector or an entire vocabulary. With embedding techniques, words are seen as dense vectors showing the projection of words in vector space. The position of tokens in the vector space is learned from the corpus and considering the surrounding words when it is used. Two popular methods for word embedding are word2Vec and GloVe.

This research used the Keras framework which covers embedding layers for neural networks on text data. It is flexible and used in different ways such as:

- To learn word embedding which can be re-used in another model,
- In deep learning model, it can equally learn with the model itself
- It can accept pre-trained models.

The task achieved is to create vocabulary and assign an index to every unique word, use the word index to convert word sequence to integer sequence, and append zero to the maximum length of the sequence to obtain a uniform sequence of length. The last step before the model build is to use two categorical functions of Keras to make the target variables as one-hot encoding.

Implementations and Hyperparameters

For this task, two different recurrent networks (LSTM and GRU) and their bi-directional. As the primary aim is to compare each model fairly with the same number of parameters. The implementation of this model is with deep learning framework Keras (Keras, n.d.). This is a deep learning API running a

machine learning platform TensorFlow without showing the expression of gradients as they are computed automatically. The Embedding layer in Keras automatically creates the matrix which maps the integer to the embedding. They are learned during the backward propagation.

In Keras there two ways of building a model. The first is through a sequential API or functional API.

Due to the long time required to train the model, we tried a small set of values. The first layer is the embedding which passes vocabulary size and input length. I chose 100 as my embedding layer which means that each word will be represented by a vector of 100 numbers. This is similar to word2Vec. The input length shows the size of the sequence in which 500 was used as seen in fig 4.3. I imported the sequence, LSTM, BiLSTM, dense, embedding layer, GRU, and BiGRU. I imported the early stopping which helps the training process in the neural network at the right time. When the validation loss keeps increasing for some epoch which is specified, the training stops.

To avoid the model from overfitting, we used the early stopping techniques and stopped at a point when the evaluations stopped to improve.

The early stopping saves a copy of the model. We choose 300 as the LSTM layer, a time step of 0.1, recurrent drop out as 0.2 dense layers as 64 neurons, output Layer 9 neuron of the 9 categorical tags. The optimizer used was Adam and loss categorical cross-entropy for classification. Batch size at 100 and epoch of 100.

Result and Analysis

In this section, I looked at the model built for each model, its diagnostic plots for a better understanding of the performance of the model over time.

LSTM

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 500, 100)	46770400
lstm_1 (LSTM)	(None, 300)	481200
dense_1 (Dense)	(None, 64)	19264
dense_2 (Dense)	(None, 9)	585

Total params: 47,271,449
Trainable params: 47,271,449
Non-trainable params: 0

None

Figure 4.1: LSTM model summary.

A step forward was to fit the model to the training data and evaluate on hold outset. The metrics to test the model is accuracy. The early stopping model will be saved when we get the best size. I used a batch size of 1200, epoch as 100 although we have early stopping in place. Each of the epoch for LSTM training takes approximately 3525 seconds but on a fast GPU, it took 80 seconds to train. The validation accuracy kept increasing and after some time the validation loss and accuracy stops improving. That's when our model stops training. Fig 5.2 shows the training loss continued to decrease but validation loss decreased at a point became stable. We used the classification model to create a report on a validation set with F1 score and Fig 5.3 gives us a score of 0.91 percent.

GRU

The architecture was swapped and further to fitting to the training data and evaluate on hold-outset. The metrics to test the model is accuracy. The early stopping model is saved at the best size. A batch size of 1200, epoch as 100 although we have early stopping in place was set. Each of the epoch for GRU training takes approximately 207 seconds but on a fast GPU, it took 48 seconds to train. The validation accuracy kept increasing and after some time the validation loss and accuracy stop improving. The classification model to create a report on the validation set with F1 gives us a score of 0.92 percent in Fig 4.5

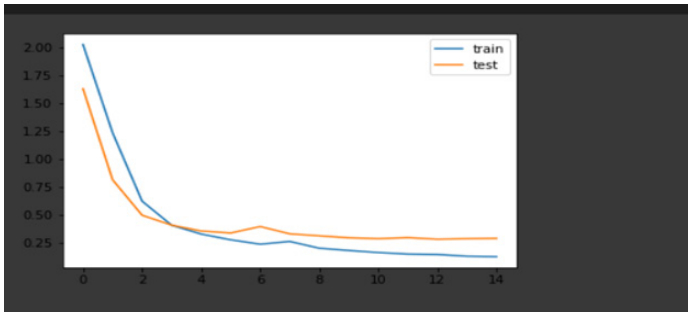


Figure 4.2: Diagnostic plot.

	precision	recall	f1-score	support
0	0.99	0.97	0.98	868
1	0.91	0.89	0.90	1408
2	0.99	0.99	0.99	1364
3	0.96	0.87	0.91	389
4	0.84	0.87	0.85	2425
5	0.88	0.90	0.89	5241
6	0.93	0.86	0.89	1556
7	0.95	0.94	0.95	2257
8	1.00	1.00	1.00	527
/ total	0.91	0.91	0.91	16035

Figure 4.3: LSTM score.

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 500, 100)	46770400
gru (GRU)	(None, 300)	361800
dense (Dense)	(None, 64)	19264
dense_1 (Dense)	(None, 9)	585

Total params: 47,152,049
Trainable params: 47,152,049
Non-trainable params: 0

None

Figure 4.4: GRU Model Summary.

	precision	recall	f1-score	support
0	0.99	0.95	0.97	868
1	0.90	0.88	0.89	1408
2	0.99	0.99	0.99	1364
3	0.92	0.86	0.89	389
4	0.86	0.87	0.87	2425
5	0.90	0.92	0.91	5241
6	0.94	0.87	0.90	1556
7	0.93	0.94	0.93	2257
8	0.99	1.00	0.99	527
avg / total	0.92	0.92	0.92	16035

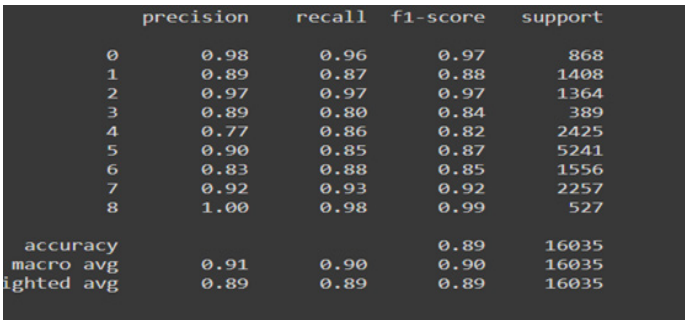
Figure 4.5: GRU score.

Bi-direction analysis of GRU and LSTM.

The same hyperparameters for GRU and LSTM was used for their bi-directional and below are the outcome. BiGRU performed better than BiLSTM.

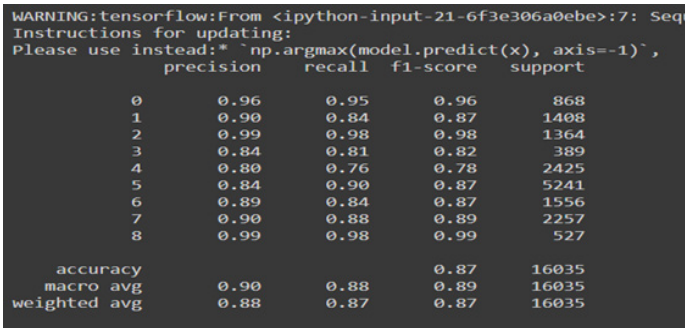
BiLSTM, the epoch training time was 225 seconds. Recall that it took LSTM 3525 seconds. BiLSTM trained with half of the time it will take LSTM to train.

BiGRU trained with 45 seconds each epoch which is 20% of the time it took a full GRU to train. Very fast indeed.



	precision	recall	f1-score	support
0	0.98	0.96	0.97	868
1	0.89	0.87	0.88	1408
2	0.97	0.97	0.97	1364
3	0.89	0.80	0.84	389
4	0.77	0.86	0.82	2425
5	0.90	0.85	0.87	5241
6	0.83	0.88	0.85	1556
7	0.92	0.93	0.92	2257
8	1.00	0.98	0.99	527
accuracy			0.89	16035
macro avg	0.91	0.90	0.90	16035
ighted avg	0.89	0.89	0.89	16035

Figure 4.6: Bi-directional GRU.



```
WARNING:tensorflow:From <ipython-input-21-6f3e306a0ebe>:7: Seq
Instructions for updating:
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,
```

	precision	recall	f1-score	support
0	0.96	0.95	0.96	868
1	0.90	0.84	0.87	1408
2	0.99	0.98	0.98	1364
3	0.84	0.81	0.82	389
4	0.80	0.76	0.78	2425
5	0.84	0.90	0.87	5241
6	0.89	0.84	0.87	1556
7	0.90	0.88	0.89	2257
8	0.99	0.98	0.99	527
accuracy			0.87	16035
macro avg	0.90	0.88	0.89	16035
weighted avg	0.88	0.87	0.87	16035

Figure 4.7: Bi-direction LSTM.

Conclusion

This research aimed to achieve effectively three main goals:

- Categorising website-based content.
- Utilising the sequence information to make a prediction based on the context.
- Providing an analysis of the characteristics of LSTM and GRU with an extension to their bi-directional on-text data with a recommendation on the best model for text data classification.

The goals were achieved on a text dataset that contains 54,445 website contents. Python programming language was used on google Colab with Keras to perform all necessary text cleaning and model build. The LSTM, GRU, BiLSTM, and BiGRU were built with the Keras function but first was the embedding layer that creates the embedding matrix by mapping the integer to the embedding for learning during the backpropagation. The output label is converted to an integer and a one-hot vector. Padding, sequence distribution, and tokenisation were done on the text data as part of text classification. The model layer (GRU, LSTM, BiGRU, and BiLSTM) was built with regularisation techniques and a recurrent dropout to drop recurrent hidden units between time steps.

While there has been a different school of thought that GRU could be a better replacement for LSTM, the outcome of such research hinted LSTM performs better than GRU, although it was done on a small text dataset. Apart of small text data, computing power was the case of the experiment, which took more than 24 hours for one model to be built. Hence, they decided to make a small data set with LSTM performing better.

My approach to this work and using a large text dataset provides new insight and clearly illustrates that GRU performed better than LSTM in terms of accuracy and training time. It is a better algorithm for text data classification. Comparing BiLSTM and BiGRU, BiGRU performed better with 20% less time than BiLSTM.

To better understand the implication of these studies and how gated unit improves learning and solidify this work's contribution, more experiments will be required in the future with.

Reference

1. Akash, S. (2022). Anomaly detection for temporal data using long short term. *Sweden, Royal Institute of Technology*.
2. Alex, G., Abdel-rahman, M. & Goeffery, H. (2013). Speech recognition with deep recurrent neural network. *IEEE internationa conference*, pp. 6645-6649. doi: https://doi.org/10.1109/ICASSP.2013.6638947
3. Ankit K, Peter O, Mohit I, James B, Ishaan G, Victor Z, Romain P, & Richard S. (2016). Ask me anything: Dynamic memory networks for natural language processing. s.l., *Computer and language*. arXiv:1506 07285v5. doi: https://doi.org/10.48550/arXiv.1506.07285
4. Bengio, Y., Simard, P. & Fransconi, P. (1994). Learning long term dependencywith gradient descent is difficult. s.l., *IEEE transaction on neural network*. 1994;5(2), pp. 157-66. doi: 10.1109/72.279181
5. BSCS and Videodiscovery, I., (2000). [Online] Available at: https://science.education.nih.gov/supplements/webversions/BrainAddiction/guide/lesson2-1.html [Accessed 21/07/2020 July 2020].
6. Charu, A. C., & Cheng Xiang, Z. (2012). Minint Text Data. 4 ed. s.l.: *Springer, Boston, MA*.
7. Charu, C. A. (2018). Neural Networks and Deep Learning. Switzerland: *Springer International Publishing AG*, part of Springer Nature. Retrieved from file:///C:/Users/RAJ/Downloads/9783319944623.pdf
8. Casanueva, I. Temčinas, T., Gerz, D., Henderson, M., & Vulić, I. (2020). "Efficient intent detection with dual sentence encoders," *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI [Preprint]*. Available at: https://doi.org/10.18653/v1/2020.nlp4convai-1.5
9. Cho, k., Bart van, M., Dzmitry, B. & Yoshua, B. (2014). Encoder Decorder Approach. *Computer Science and Language*, pp. 1724-1734.
10. Christain, J., Yoshua, B., Rejean, D. & Pascal, V. (2003). A neural Probabilistic Language Model. *Journal of Machine Learning Research*, pp. 1137-1155. Retrieved from https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf
11. David, L. D. (1992). An Evaluation of Phrasal and Clustered Representations on a Text Categorisation Task. *Center for Information and Languiage Studies University of Chicago*, Volume 1. Doi: http://dx.doi.org/10.1145/133160.133172
12. David, R. E., Geoffrey, H. E. & Ronald, W. J. (1986). Learning representation by back propagating errors. *Semantic scholar, Volume* 323, pp. 533-538. doi: https://doi.org/10.1038/323533a0
13. Duyu, T., Bing, Q., Xiaocheng, F. & Ting, L. (2016). Effective LSTM for target-dependent sentiment classification. *26th conference on computational and linguistics*, pp. 3298-3307. Retrieved from https://aclanthology.org/C16-1311.pdf
14. Dzmitry, B., Kyunghyun, C. & Yoshua, B. (2014). Neural machine translation by jointly learning to align and translate. *Computational language[cs.CL]*. doi: https://doi.org/10.48550/arXiv.1409.0473
15. Frank, R. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological Review, *Volume 65*, p. 6. doi: https://doi.org/10.1037/h0042519
16. Graves, A. (2012). Supervised sequence labelling with recurrent neural networks. 2 ed. s.l.: ISBN 978-3-642-24797-2.
17. Guozheng, R., Weihang, H., Zhiyoung, F. & Qiong, C. (2018). LSTM with sentence representation for documents-level sentiment classifcation. *Neurocomputing, Volume* 308, pp. 49-57. doi: https://doi.org/10.1016/j.neucom.2018.04.045

18. Junyoung, C., Caglar, G., Kyung Hyun, C. & Yoshua, B. (2014). Emperical evaluation of gated recurrent neural networks on sequence modeling. *Neural and Evolutionary Computing*. arXiv:14123555y1. doi: https://doi.org/10.48550/arXiv.1412.3555

19. Kelvin, L. & Alex, W. H. (1990). A time delay neural network architecture for isolated word recognition. *Pergamon Press Plc, Volume* 3(1), pp. 22-43. doi: https://doi.org/10.1016/0893-6080(90)90044-L

20. Ke, T., Arianna, B. & Christof, M., (2016). Recurrent memory networks for language modeling. *Proceeding of NAACL-HLT*, pp. 321-331. doi: http://dx.doi.org/10.18653/v1/N16-1036

21. Micheal, M. C. (1991). Induction of multiscale temporal structure. s.l.: *Neural information processing system conference*. Retrieved from https://papers.nips.cc/paper/1991/hash/53fde96fcc4b4ce72d7739202324cd49-Abstract.html

22. Micro, R., Philemon, B., Maurizio, O. & Yoshua, B. (2018). Light gated recurrent units for speech Recognition. *Electrical Engineering and system science Audio and Speech processing, Volume* 2(2), pp. 92-102. doi: https://doi.org/10.48550/arXiv.1803.10225

23. Mikolov, T., Kai, C., Greg, C. & Jeffrey, D. (2013). Efficient estimation of word representations in Vector space. aXiv:1301.3781v3[cs.CL]. doi: https://doi.org/10.48550/arXiv.1301.3781

24. Missinglink.ai, n.d. Missinglink.ai. [Online] Available at: https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/ [Accessed 25 July 2020].

25. Pascanu, R., Mikolov, T. & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *30th International Conference proceeding on Machine Learning. PMLR 28(3)*, pp. 1310-1318. Retrieved from https://proceedings.mlr.press/v28/pascanu13.html

26. Paul, W. J. (1990). Backpropagation through time: what it does and how to do it. s.l., *Computer Science, Mathematics. Proceeding of IEEE, Volume* 78, pp. 1550-1560. doi: https://doi.org/10.1109/5.58337

27. Pengfei, L., Xipeng, Q. & Xuanjing, H. (2016). Recurrent neural network for text classifcation with multi-tasking learning. *Computation and Language (cs.CL)*. arXiv.org. doi: https://doi.org/10.48550/arXiv.1605.05101

28. Piccinni, G. (2004). The First Computational Theory of Mind and Brain: A Close Look at Mcculloch and Pitts's "Logical Calculus of Ideas Immanent in Nervous Activity". *Synthese, 141(2)*:175–215, 2004, p. 222. doi: https://doi.org/10.1023/B:SYNT.0000043018.52445.3e

29. Richard, S. et al., (2016). Recursive deep model for semantic compositionality over a sentiment treebank. Standard ford University 94305 USA. pp. 1631–1642. Retrieved from https://aclanthology.org/D13-1170/

30. S. Sheng, M, Holbrook, P. Kumarahuru, L.F. Cranor, & J. Downs. (2010). Who falls fo phish? a demographic analysis of phishing susceptibility and effectiveness of interactions. New York, NY, USA ACM, s.n. pp. 373–382. doi: https://doi.org/10.1145/1753326.1753383

31. Sebastian, R. (2016). An overview of gradient descent optimisation algorithms. *Computer science, Machine Learning. Volume* 1. arXiv:1609.04747, doi: https://doi.org/10.48550/arXiv.1609.04747.

32. Sepp, H. & Jurgen, S. (1997). Long short term memory. *Neural computation, Volume* 9(8), pp. 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

33. Srividhya, V. & Anitha, R. (2010). Evaluating preprocessing techniques in Text categorisation. s.l., *International Journal of Computer Science and application*. Retrieved from http://sinhgad.edu/ijcsa-2012/pdfpapers/1_11.pdf

34. Sutskever, I., Oriol, V. & Quoc, V. (2014). Sequence to sequence learning with neural networks. arXiv:1409.3215y3[cs.CL]. https://doi.org/10.48550/arXiv.1409.3215

35. Tobias, L. & Matthias, R. (2017). Understanding Consumer Behaviour with recurrent neural networks. s.l.: s.n. Retrieved from https://doogkong.github.io/2017/papers/paper2.pdf

36. Tomas, M., Martin K., Luka´s B., Jan "Honza" C., & Sanjeev K. (2010). Recurrent neural network based language model. *Interspeech 2010*, pp. 1413-1421. Retrieved from https://www.fit.vutbr.cz/research/groups/speech/publi/2010/mikolov_interspeech2010_IS100722.pdf

37. Vidhya, A. 207. Analytic Vidhya. [Online] Available at: https://courses.analyticsvidhya.com/courses/take/natural-language-processing-nlp/lessons/6014710-getting-started-with-neural-network [Accessed 22 July 2020].

38. Xu, N., Mao, W., Wei, P., & Zeng, D. (2021) "MDA: Multimodal Data Augmentation Framework for boosting performance on sentiment/emotion classification tasks," IEEE Intelligent Systems, 36(6), pp. 3–12. doi: https://doi.org/10.1109/mis.2020.3026715

39. Zhang, Y., Zhang, R., Mensah, S., Liu, X., & Mao, Y. (2022). "Unsupervised sentence representation via contrastive learning with mixing negatives,". *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(10), pp. 11730–11738. doi: https://doi.org/10.1609/aaai.v36i10.21428

40. Jiang, T., & Gao, X. (2022). "Deep learning of subject context in ideological and political class based on recursive neural network,". *Computational Intelligence and Neuroscience, Vol* 2022, pp. 1–8. doi: https://doi.org/10.1155/2022/8437548

41. Wu, D. Wu, L., Huang, J., & Wang, X. (2022). "Isa-PredRNN: An improved self-attention predRNN network for Spatiotemporal Predictive Learning,". *2022 International Conference on Image Processing, Computer Vision and Machine Learning (ICICML)* [Preprint]. doi: https://doi.org/10.1109/icicml57342.2022.10009868

42. Van Gompel, J., Spina, D. & Develder, C. (2022). "Satellite based fault diagnosis of photovoltaic systems using recurrent neural networks,". *Applied Energy, 305*, p. 117874. doi: https://doi.org/10.1016/j.apenergy.2021.117874