Ort Braude College of Engineering
Software Engineering Department

Call Attention to Rumors: Deep Attention Based Recurrent Neural Networks for Early Rumor Detection

In Partial Fulfillment of the Requirements for
Final Project in Software Engineering (Course 61771)

Karmiel, January, 2021

Authors:
Yana Mamedov, ID: 316882646
Shay Axelrod, ID: 312592199

Supervisor:
Prof. Zeev Volkovich

CONTENT

1. INTRODUCTION

The rise in social media usage in recent years has made it a powerful platform for spreading rumors. The spread of rumors can pose a threat to cybersecurity, social, and economic stability worldwide. This project aims to detect the spread of rumors by identifying them on social networks in the early stages of their propagation in an automated way. RNNs have been proven to be effective in recent machine learning tasks for handling long sequential data. Three main challenges in early rumor detection must be addressed: (1) the system must adopt new features automatically and should not be hand-crafted; (2) the solution uses RNN. This algorithm has some well-known problems that prevent the processing of exceedingly long sequences of texts; (3) many duplications of posts with different contextual focuses must be handled. An attention-based RNN powered by long short-term memory (LSTM) with term frequency-inverse document frequency (tf-idf) mechanisms is proposed to overcome these challenges. In the project, the system detects rumors automatically using a deep attention model based on recurrent neural networks (RNN). For simplicity, the model is pre-trained using a dataset from social media sources. The model gets textual sequences of information from posts as input and constructs a series of feature matrices. Then, the RNN with an attention mechanism automatically learns new and hidden text representations. The attention mechanism is embedded in the system to help focus on specific words for capturing contextual variations of relevant posts over time. In the end, an additional hidden layer with a sigmoid activation function uses those text representations and predicts, as output, whether a text is a rumor or not. Furthermore, the system enables a trainer to train the algorithm with some new datasets.

2. THEORY

2.1 Background And Related Work

The work is related to early rumor detection and attention mechanism. In the following subsections, you will be briefly introduced to those and some mathematical backgrounds that are being used in the solution.

2.2 Early Rumor Detection

The problem of rumor detection can be cast as a binary classification task. Finding and selecting the different features significantly affects the performance of the classifier. Many real-time rumor detection approaches require intensive manual efforts and are extremely limited by the data structure and relatively slow. Recently, recurrent neural networks were found to be effective in rumor detection, utilizing sequential data to spontaneously capture temporal textual characteristics. However, lacking a large dataset with differentiable contents in the early stages of the rumors drops the performance of these methods significantly because they will fail to distinguish essential patterns. To overcome these challenges, the system must learn new features automatically, and it must also have a large dataset during training.

2.3 Recurrent Neural Networks (RNN)

Recurrent neural networks, or RNNs, are a family of feed-forward neural networks for processing sequential data. RNNs work as follows: Words get transformed into machine-readable vectors. Then, the RNN processes the vectors, one element x_t at a time. While processing, it passes its hidden state h_t to the next step of the sequence. The hidden state acts as the neural network's memory and remembers a history of the previous steps. Every step, the RNN gets input and the previous hidden state and combines them to form a vector. That vector now has information on the current input and all the previous ones. The vector then goes through a $tanh$ activation and outputs a new hidden state h_t, also

known as the memory of the network. The forward propagation begins with a random initial state h_0, then, for each time step t from $t = 1$ to $t = \tau$, the following update equations are applied:

$$h_t = \tanh(Ux_t + Wh_{t-1} + b),$$
$$o_t = Vh_t + c \quad \textit{(Final output of the RNN).} \tag{1}$$

2.4 Long Short-Term Memory (LSTM)

RNNs suffer from short-term memory loss. If a sequence is long enough, an RNN will have a hard time remembering information from earlier time steps. Thus, when trying to process long sequences of posts to make predictions, RNN's may forget essential details from previous posts. The gradient computation of RNNs involves performing backpropagation through time (BPTT) [1]. During backpropagation, recurrent neural networks suffer from the "vanishing gradient" problem. Gradients are values that are used to update the neural network weights. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it does not contribute too much learning. To overcome this well-known problem, LSTMs were created [2, 3, 4]. LSTMs have internal mechanisms called gates that can regulate the flow of information. These gates help the RNN learn which data in a sequence is crucial to remember or forget. Finally, the RNN can pass relevant information down a long chain of sequences to make better predictions. The structure of LSTM is formulated as:

$$\begin{aligned} i_t &= \sigma(U_i h_{t-1} + W_i x_t + V_i c_{t-1} + b_i), \\ f_t &= \sigma\big(U_f h_{t-1} + W_f x_t + V_f c_{t-1} + b_f\big), \\ o_t &= \sigma(U_o h_{t-1} + w_o x_t + V_o c_t + b_o), \\ c_t &= f_t c_{t-1} + i_t \, tanh(U_c h_{t-1} + W_c x_t + b_c), \\ h_t &= o_t \, tanh(c_t), \end{aligned} \tag{2}$$

where i_t, f_t, o_t, c_t, h_t are the *input gate*, *forget gate*, *output gate*, *cell state*, *hidden state* at step t respectively. All of them are the same size as the hidden vector h. The W/U/V are weight matrices [5]. For example, U_i is the hidden-input gate matrix, w_0 is the input-output gate matrix etc. b is the bias vector.

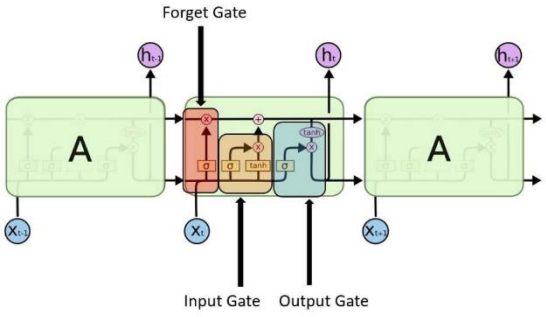


Fig. 1: A visual example of an SLTM with all its gates. The forget gate, input gate, and output gates are highlighted in red, orange, and blue, respectively.

2.4.1 Hyperbolic tangent activation (tanh)

$tanh$ is a hyperbolic tangent, non-linear function. This activation is used to help regulate the values flowing through the network. The $tanh$ function squishes values to always be between -1 and 1. Due to

various math operations, when vectors are flowing through a neural network, they undergo many transformations. A value that is continuously multiplied can grow to become astronomical, causing other values to seem insignificant. The $tanh$ function ensures that values stay between -1 and 1, thus regulating the output of the neural network. The hyperbolic tangent function is defined by the formula:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3)$$

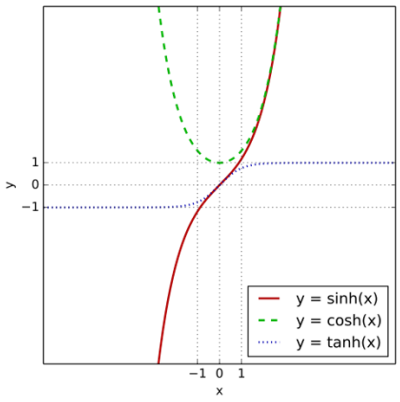


Fig. 2: The graph of the hyperbolic tangent where tanh(x), cosh(x), and sinh(x) are marked with a blue-dotted line, green-dashed line, and a red-solid line respectively

2.4.2 Sigmoid activation (σ)

A sigmoid activation is similar to the tanh activation. Instead of squishing values between -1 and 1, it squishes values between 0 and 1. This helpful technique lets the RNN remember or forget data because any number getting multiplied by 0 is 0, causing values to disappear or be "forgotten." Any number multiplied by 1 is the same value; therefore, that value remains the same and is "remembered." This is how the RNN learns which data is not important and, therefore, can be forgotten or which data is important to keep. The sigmoid function is defined by the formula: $\sigma(x) = \frac{1}{1+e^{-x}}$ (4)

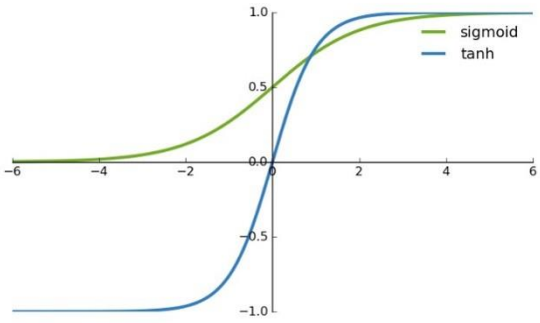


Fig. 3: The graph shows the difference between tanh and sigmoid.

2.4.3 Forget gate

This gate oversees deciding what information should be forgotten or remembered. Information from the previous hidden state h_{t-1} and information from the current input x_t are passed through the sigmoid

function. Values come out between 0 and 1. Values closer to 0 means forget, and values closer to 1 means remember.

2.4.4 *Input gate*

This gate oversees updating the current cell state. First, the previous hidden state h_{t-1} is combined with the current input x_t and are then inserted into a sigmoid function. The sigmoid function decides which values should be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. At the same time, a copy of $(h_{t-1} + x_t)$ are inserted into a $tanh$ function that squishes the values between -1 and 1. This helps regulate the network. Then a multiplication occurs between the output of the $tanh$ and the output of the sigmoid functions $(\sigma(h_{t-1} + x_t) \cdot \tanh(h_{t-1} + x_t))$. The sigmoid output will decide which information is essential to keep from the tanh output.

2.4.5 *Output gate*

The output gate oversees deciding what the next hidden state h_t should be. By now, the hidden state contains information on all the previous inputs. First, the previous hidden state and the current input are passed into a sigmoid function $\sigma(h_{t-1} + x_t)$. Then the newly modified cell state is passed into a $tanh$ function $(\tanh(c_t))$. Then a multiplication occurs between the $tanh$ output and the sigmoid output to decide what information the hidden state should carry. The output is the new hidden state h_t. The new cell state and the new hidden state are then carried over to the next time step $t + 1$.

2.4.6 *Cell state*

As the current cell state travels along the LSTM module, it is first multiplied by the forget vector. This has the possibility of forgetting irrelevant values. It is then combined with the output of the input gate, which updates the cell state to new values that the neural network finds relevant. That creates a new cell state.

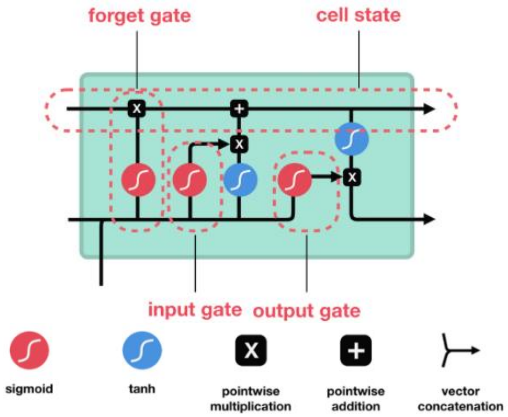


Figure 4: visual example of an LSTM with all its gates.

2.5 Attention Mechanism

The neural processes involving attention is studied in Neuroscience and Computational Neuroscience [6, 7]. Attention mechanism has a significant impact on neural computation by its ability to select the essential pieces of information being relevant to compute the neural response. An attention model is a method that takes *n* sequential arguments $y_1, \ldots, y_n$, and a context *c*. It returns a vector *S* a "summary" of *yi* focusing on information linked to the context *c* as the weighted arithmetic mean of the *yi*, and the

weights chosen according to the relevance of each *yi* given the context *c*. There are two main types of attention mechanisms, soft attention and hard attention. Both mechanisms use a softmax function, which calculates a probability distribution consisting of *n* probabilities proportional to the exponentials on the input numbers. In other words, every output *Si* of the softmax lies between 0 and 1 and is summarized up to 1. Furthermore, the most significant components correspond to the highest probabilities. The formula defines the standard softmax function:

$$s_i = \frac{(e^{m_i})}{\sum_{j=1}^{n} e^{z_i}} \; for\; i = 1, \dots, n \; and \; m = (m_1, \dots, m_n) \in \mathbb{R}^n \tag{5}$$

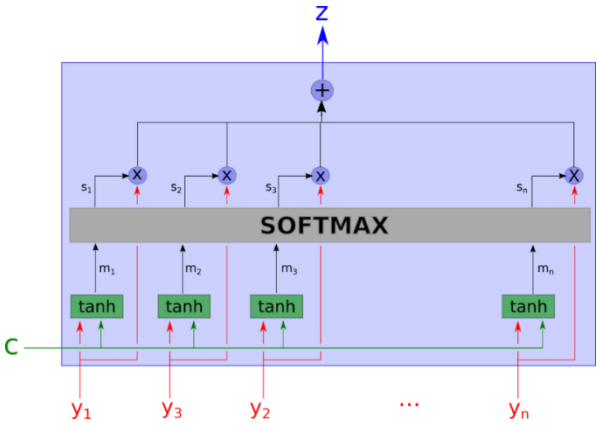


Fig. 5: Visual explanation to how a soft attention mechanism works. Input is entered from the bottom, undergoes some mathematical manipulations and outputs from the top

2.6 Backpropagation

Backpropagation is an efficient algorithm intended to calculate the derivative of the lost function concerning each parameter of the neural network [8]. The loss function estimates how a model is performing respectively, to the right output. The main principle behind backpropagation is that $\boldsymbol{error} = (\boldsymbol{right\_answer}) - (\boldsymbol{actual\_answer})$. Error interval is being calculated after any input processed through the neural network to improve the neural network reliability. This is done by updating weights and biases that will minimize the lost-function value, in this case, using the Adam optimization algorithm [9].

3. CALLATRUMORS: EARLY RUMOR DETECTION WITH DEEP ATTENTION BASED RNN

This section presents the details of the framework with deep attention for classifying social events into rumors and non-rumors.

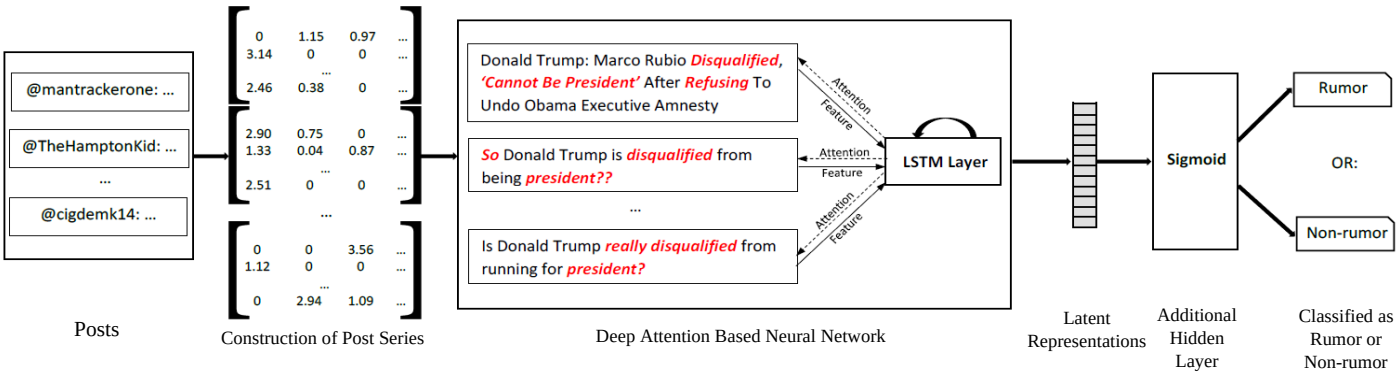


Fig. 6: Schematic overview of the framework

3.1 Data Preprocessing And Construction Of Post Series

Data preprocessing allows for removing unwanted data using data cleansing; this cleans unnecessary noise and corruptions from the data, allowing it to contain valuable information and improve performance. Before the data passes to the model, it gets lowercased, and some strings are replaced with others or even deleted. i.e., "\n" is replaced with spaces, punctuations are deleted, and URLs are replaced with the string "<url>". Individual social posts contain, as usual, limited content. But a more considerable amount of relevant posts can be easily collected to describe the central content more faithfully than a single post [10]. Hence, the system focuses on detecting rumors on the batch-level. After preprocessing Algorithm 1 groups posts into batches according to a fixed amount of N posts. This process is called: construction of variable-length post series. For datasets containing more than N posts, the algorithm iteratively takes N posts out of the dataset and puts them into a new batch. Batches containing less than N posts are ignored.

Input: Dataset with posts *DS*, amount of posts *total*, wanted amount of posts per batch *N*

Output: Batches $B = \{B_1, \dots, B_{\lfloor \frac{total}{N} \rfloor}\}$

```
1   /* Initialization */
2   currBatchIndex = 0;
3   maxBatchIndex = ⌊total/N⌋;
4   currentPostIndex = 0;
5
6   while currBatchIndex < maxBatchIndex
7       currentPostIndex = 0;
8       if total − (currBatchIndex × N)≥ N
9           /* Build another batch */
10          while currentPostIndex < N
11              B[currBatchIndex][currentPostIndex] =
12                              DS[(currBatchIndex × N) + currentPostIndex]
13              currentPostIndex++
14          end
15      else
16          /* Ignore. Do not include batches containing less than N batches */
17      end
18      currBatchIndex ++
19  end
```

Alg. 1: Constructing variable-length post series

For evaluating the occurrences of the different words, the system calculates the tf-idf for the most common *K* terms within all posts. Tf-idf is a numerical statistic estimating the importance of a word is to a document in a collection [11]. In this manner, for each batch, a *K*-word dictionary is created, indicating the importance of words in posts where 0 corresponds to a word not appearing in a post. Finally, a tf-idf array is constructed as the model input.

3.2 Deep Attention Based Neural Network And An Additional Hidden Layer.

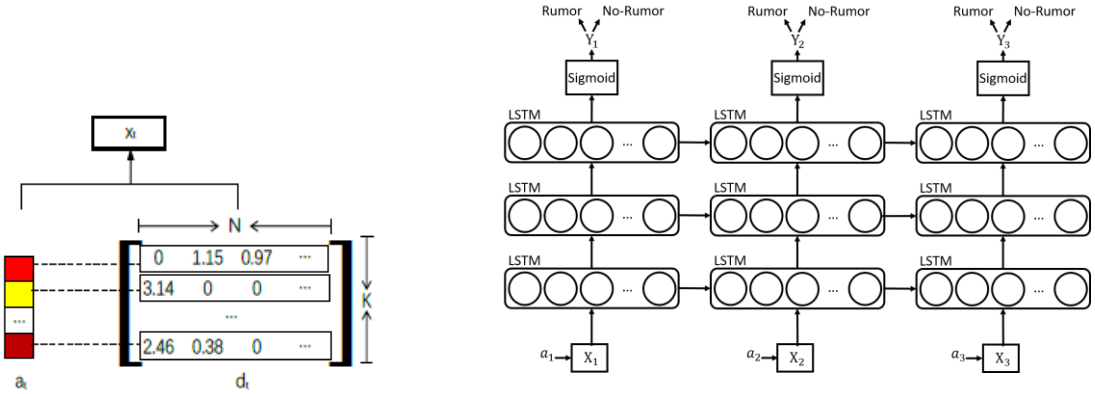


Fig. 7: On the left: Image illustrating how a_t is combined with the td-idf matrix (d_t) to form the input x_t. On the right: A visual preview of how the attention based RNN works

In this stage, three Long Short-Term Memory (LSTM) layers are employed to learn different representations for rumors. The structure of LSTM is formulated, as shown in (2). The context vector x_t is a dynamic representation of the relevant part of the post input at time t. To calculate x_t, an attention weight $a_t[i]$, i=1, …, K with softmax activation, corresponding to the feature extracted at different element positions in a tf-idf matrix d_t, is applied. y_t is a binary class of rumors and non-rumors after an additional hidden layer with sigmoid activations (see Fig.7 (right)). To reduce overfitting between existing LSTM layers, a dropout layer is added and applied to the previous layer's output. During the backpropagation stages, the system learns where to tweak the weights and the bias to become more accurate as more training occurs using the Adam optimization algorithm.

3.3 The Training Procedure

Deep attention-based neural networks are trained by measuring the derivative of the loss through the backpropagation [12] algorithm. When a deep network is trained, the main goal is to find the optimum values on each of the word matrices. The error value is being calculated to make the network reliable by adjusting the weights in each layer until the minimum cost is achieved. The model training, cross-entropy loss, is employed, encourages the model to pay attention to every element of the input word matrix.

3.3.1 Dataset

The dataset contains ~18,000 labeled posts. Some rows are empty, and some are gibberish. A preprocessing procedure is done on the dataset to ensure that all of the posts have relevant information and none are invalid. Eighteen batches are constructed from this dataset and divided into Train and Validation data, fiftenn batches for training, and three for testing. The Train data is used to train the model and modify the weights, while the Validation data is used to qualify the performance.

3.3.2 The Training Process

The training took place on a Windows 10 desktop with an AMD Ryzen 2700X, 8-core CPU, and 32GB of RAM. The model contains 3 LSTM layers consisting of 1024, 512, and 64 hidden units, respectively. Applied to each LSTM layer is a dropout of 0.3. The last fully-connected layer is activated with the sigmoid, and the Adam optimizer with sparse-categorical-cross entropy is used to calculate the loss. With a minute apart from each test, five consecutive tests showed almost identical results. To prevent overfitting, the model uses an early stopping callback to stop the training process early if it does not improve the accuracy over five successive epochs. The average results are as follows: A total training time of 72.2 seconds, 18.6 epochs, 0.28 loss, and 0.888 for accuracy.

4. SOFTWARE ENGINEERING DOCUMENTS

4.1 Requirements

ID	Requirement
1	The system will classify if the given post (text) is a rumor or not.
2	The user can load a text for the analysis.
3	The user can choose a different trained dataset for rumor detection.
5	The trainer can train the model with a new dataset of his own.
6	The trainer can save the trained state of the system.
7	The trainer can use the new trained data for classifying rumors.
8	The trainer can see the accuracy and the loss graphs of the training process.
9	The trainer can stop / pause / play the training process.
10	The system will show the percentage of training that has been completed.

Table 1: The requirements of the software

4.2 UML Diagrams

4.2.1 Use case diagram

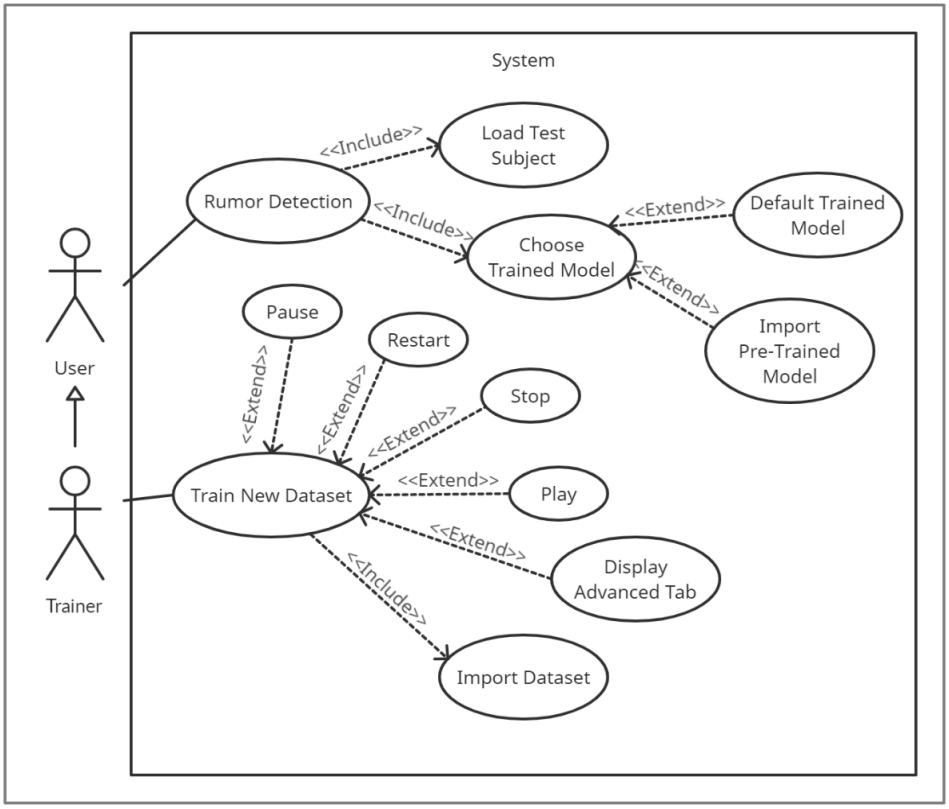


Fig. 7: The Use case diagram of the model shows two different users performing actions in the system.

4.2.2 Class Diagram

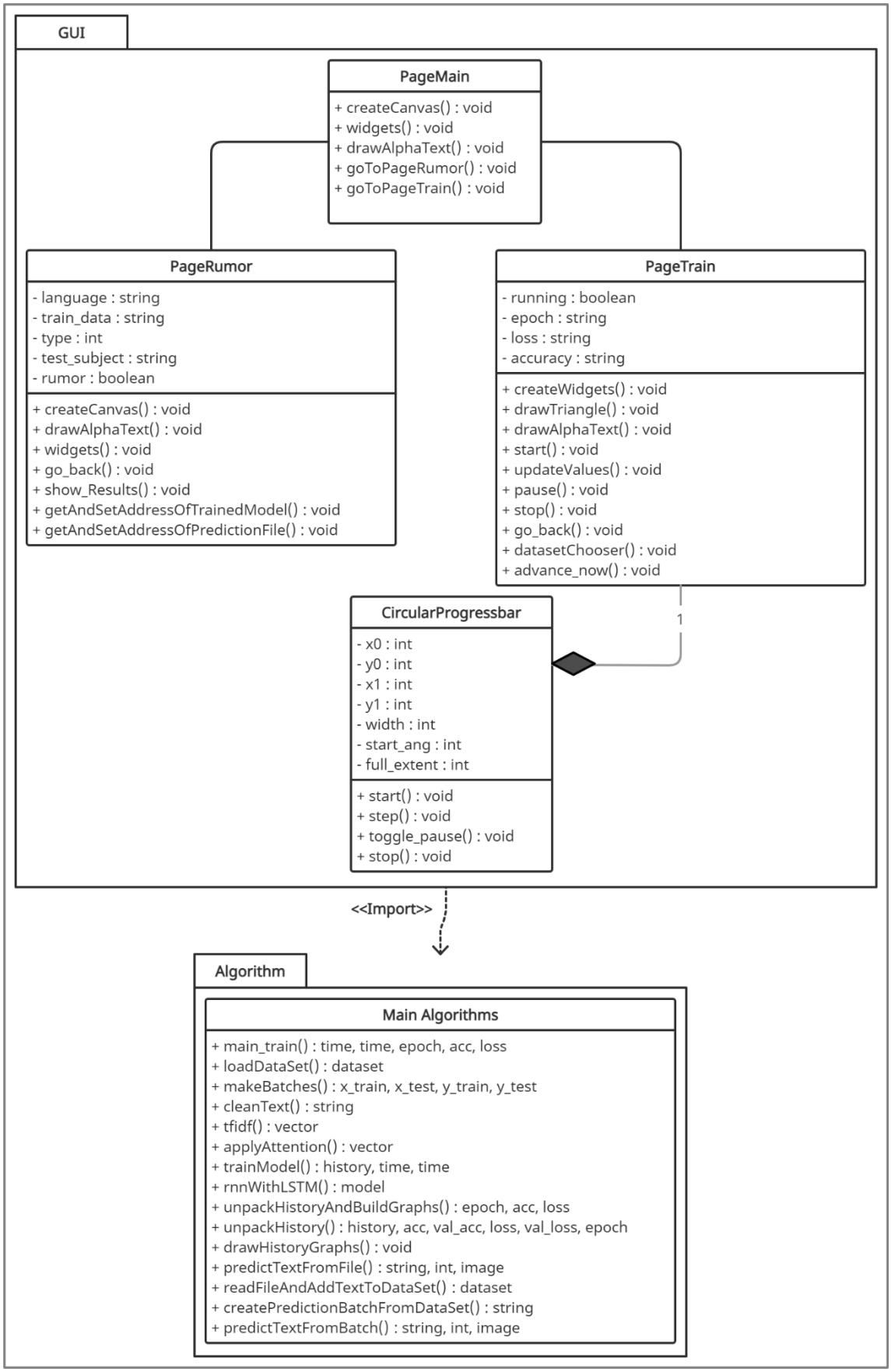


Fig. 8: The class diagram of the model shows two packages, one for the GUI, one for the algorithm.

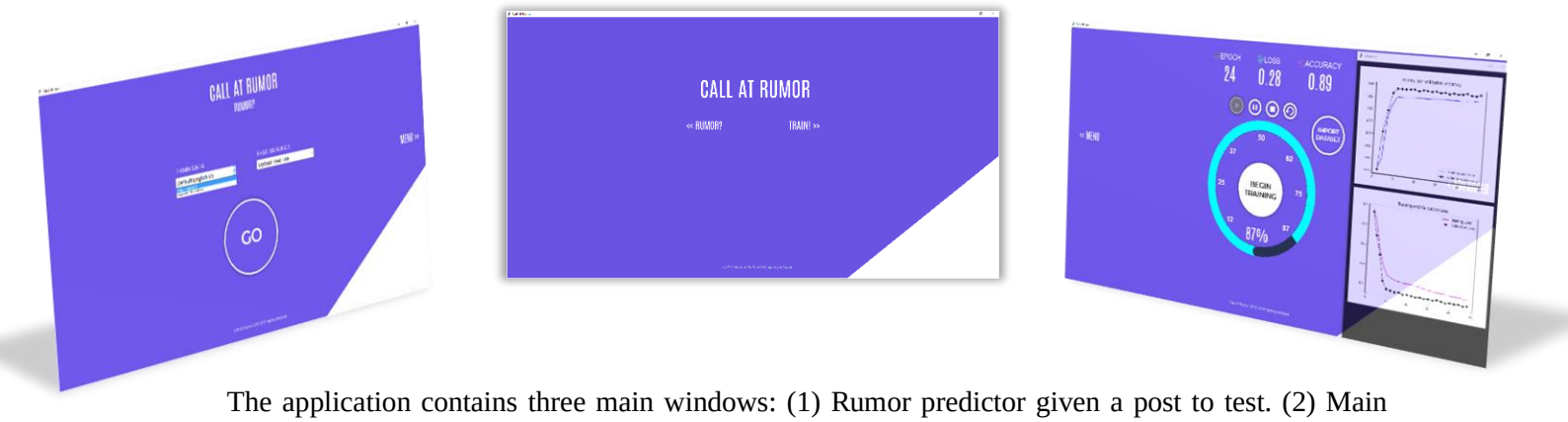
4.3 Testing Plan

To examine the system performance and accuracy, a different kind of test was performed as follows:

Test No.	Description	Expected result	Pass/Fail
1.	Upload an empty text file on the rumor prediction page and press on the "Go" button.	An error window will pop-up – "file must be not empty."	Pass
2.	Upload an incorrect file type as the dataset for training on the Training Page.	An error window will pop up – "Make sure it is valid .CSV file"	Pass
3.	Press on the "Go" button on the rumor prediction without uploading a text file and pre-trained data.	The model will run on a default pre-trained data set "Default English US" a default text.	Pass
4.	Import a new pre-trained dataset (H5 file) on the rumor prediction.	The open dialog window will pop-up.	Pass
5.	Import an incorrect file type for a pre-trained model on the rumor prediction page.	An error window will pop up – "Make sure it is a valid .h5".	Pass
6.	Upload a text file that is not from the trained dataset, labeled as a rumor, and press on the "Go" button.	The GUI will show "Rumor" with a percentage below.	Pass
7.	Upload a text file that is not from the trained dataset, labeled as a not rumor, and press on the "Go" button.	The GUI will show "Not A Rumor" with a percentage below.	Pass
8.	Press on the button - "Import Dataset" on the training Page for uploading a new CSV file.	The open dialog window will pop-up.	Pass
9.	Press on the "Begin Training" button on the training page after importing a dataset.	The progress bar will start progressing, the play button will be dark, the pause and stop buttons will be white. The epoch, loss, and accuracy values will be displayed.	Pass
10.	Press on the "Advanced" button without training a dataset first.	Display an empty window on the right.	Pass
11.	Press on the "Advanced" button after training a dataset first.	Open a new window on the right with the loss and the accuracy graph.	Pass
12.	Press on the "Manu" button for returning to the main page	The main page window will open.	Pass

Table 2: The testing plan of our model

4.4 Graphic User Interface



The application contains three main windows: (1) Rumor predictor given a post to test. (2) Main menu. (3) Training the model. The functionality displayed as follows:

Fig. 9: This is the main menu of the application. Here a user can choose either to train a model or use a pre-build one to test for a rumor. Clicking **TRAIN!** will navigate the user to the training window where he/she can train the model. Clicking **RUMOR?** will navigate the user to the rumor classification window where he/she is able to test a test subject.

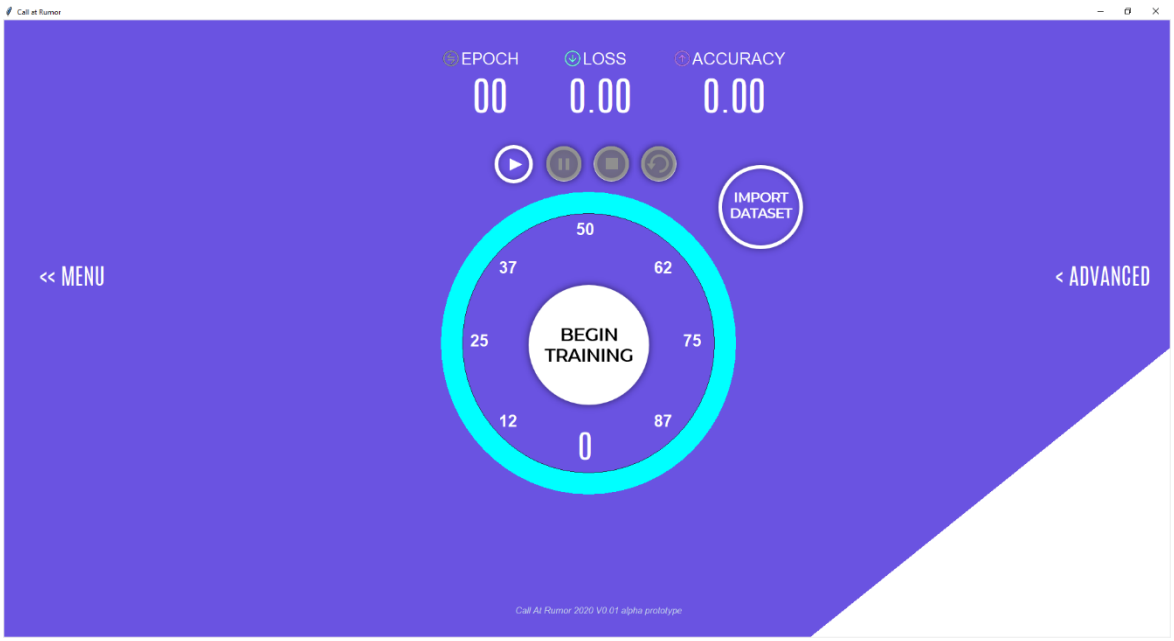


Fig. 10: In the training window, the user can train an RNN with his own dataset! This enables users to train an RNN to work in their local languages as well as the default English language. The user will only need to provide the dataset and click on the **BEGIN TRAINING** button.

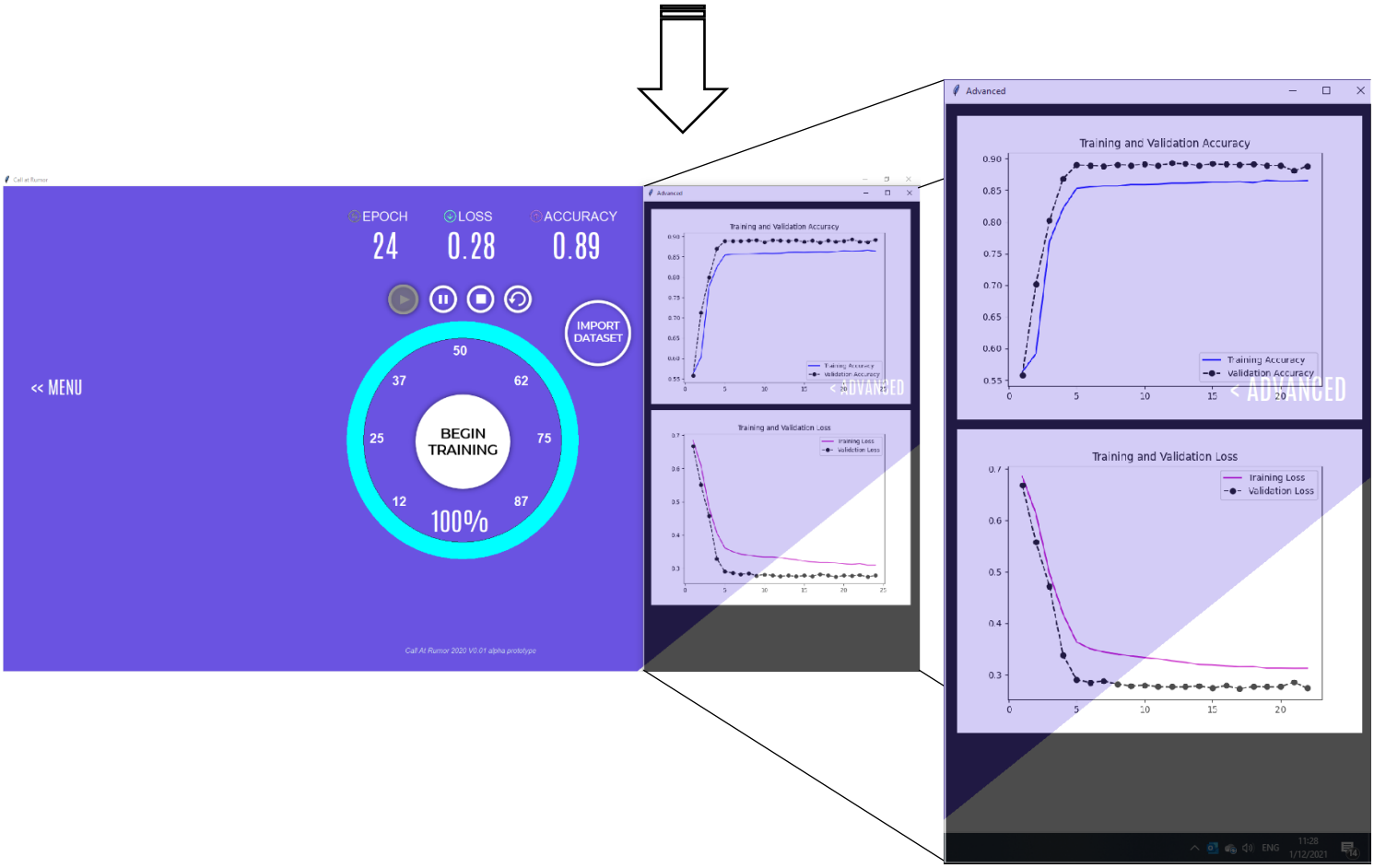


Fig. 11: Post the training process the user can view the available information in the **ADVANCED** tab by clicking it. Here he/she will see 2 graphs, the first illustrates the accuracy of the RNN and the second graph illustrates the loss of the RNN.

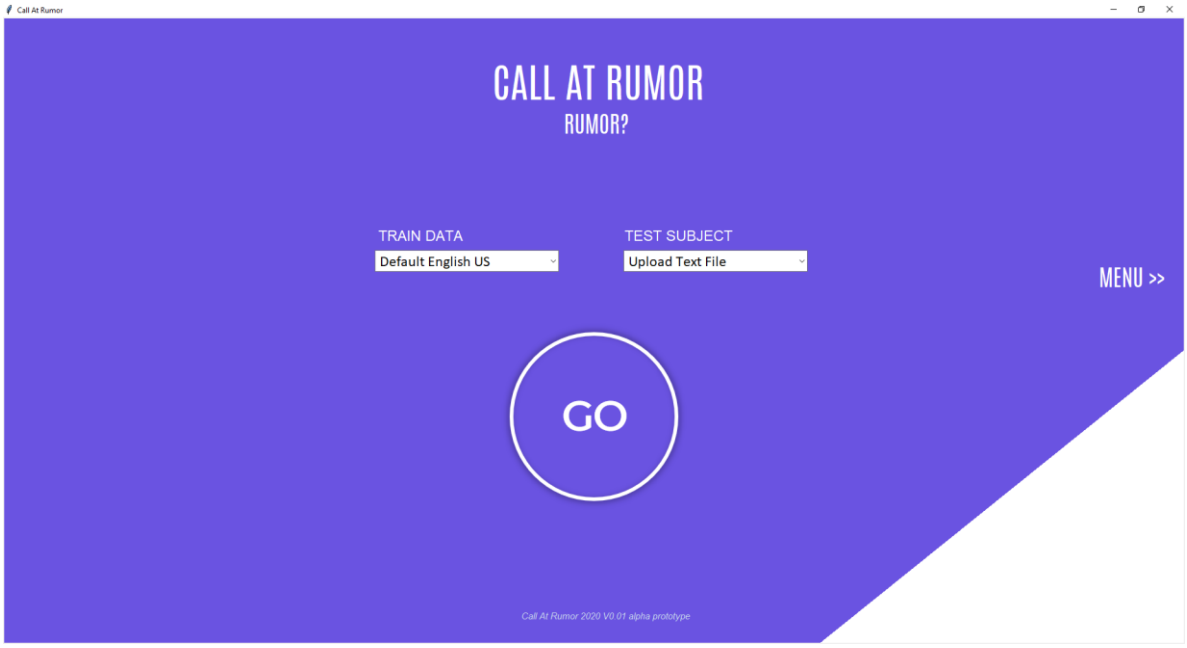


Fig. 12: This is the 'rumor prediction' window where the user can test a test subject and find out if it is a rumor or not. First, the user will have to select his/hers trained model. This is the data for the trained RNN, his test subject will be entered into this trained RNN for rumor classification. After selecting the correct configurations, the user will enter his/hers test subject. Finally, the user will hit the **GO** button and the classification process will begin.

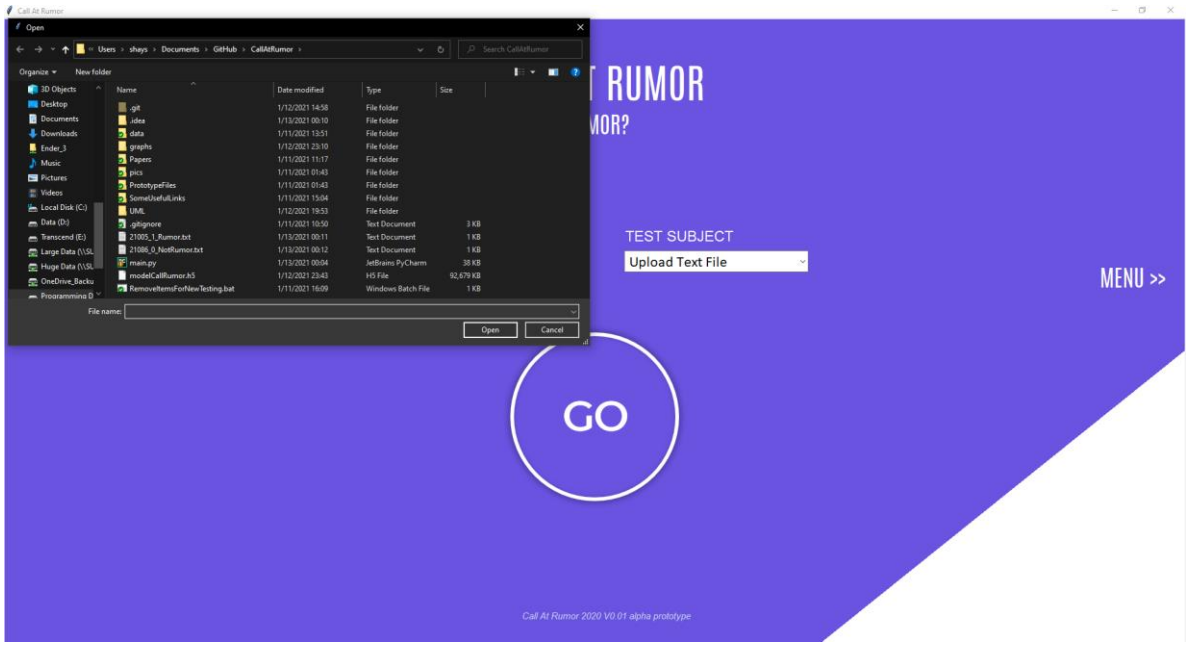


Fig. 13: Whenever a user chooses a custom Trained-Model, or any test subject, a windows dialog will popup allowing the user to select the desired file.

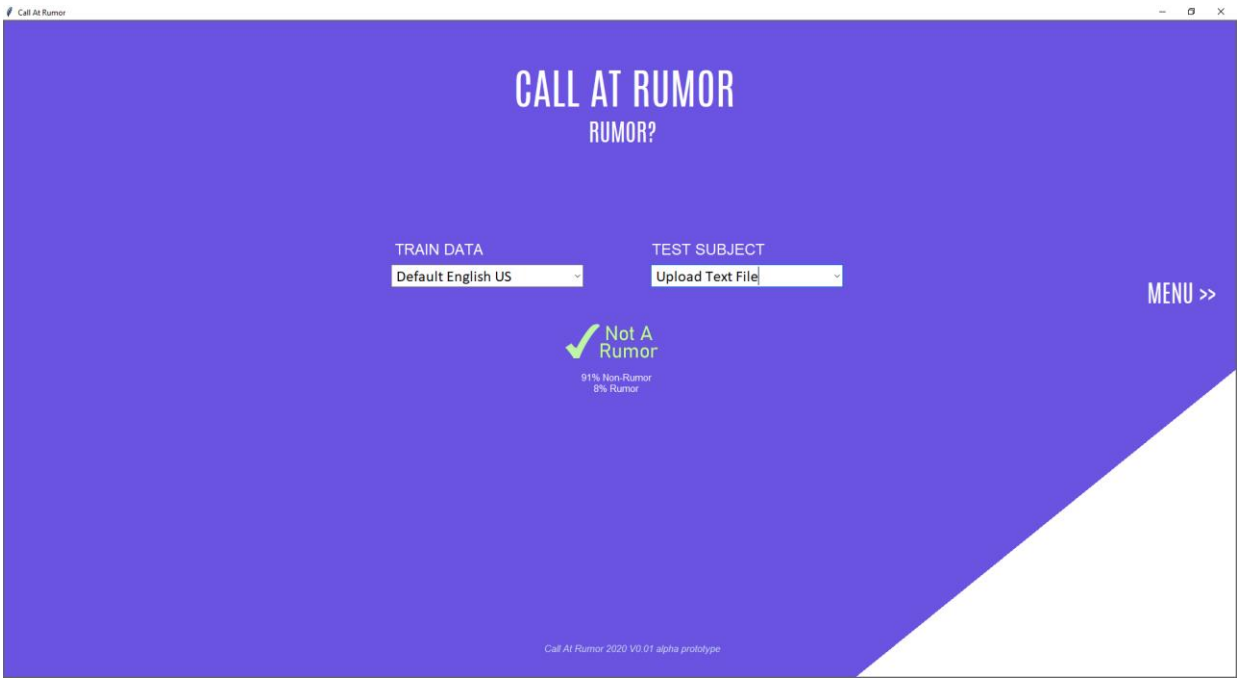


Fig. 14: After a prediction is made, an image displaying rumor/non-rumor will be shown along with the percentages. This is what a non-rumor prediction looks like.

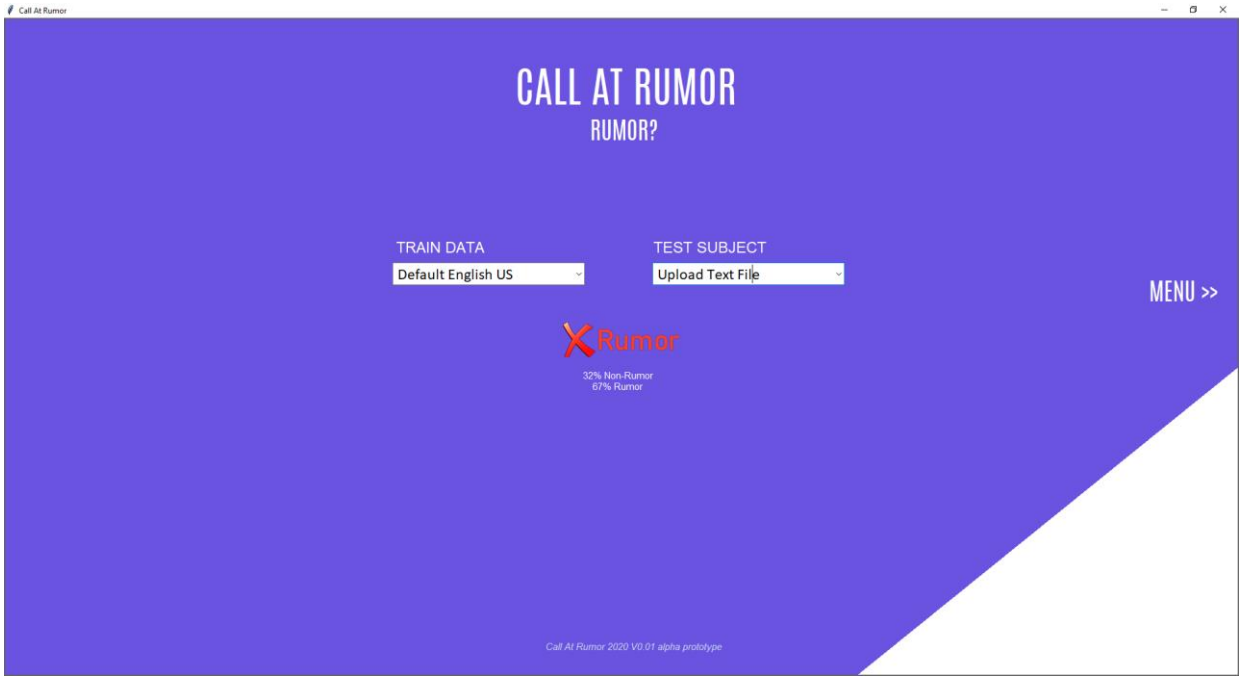


Fig. 15 example of how the rumor prediction looks like

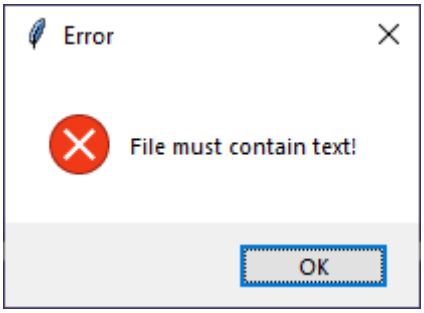


Fig. 16: If the user attempts to upload an empy text file an Error window will popup.

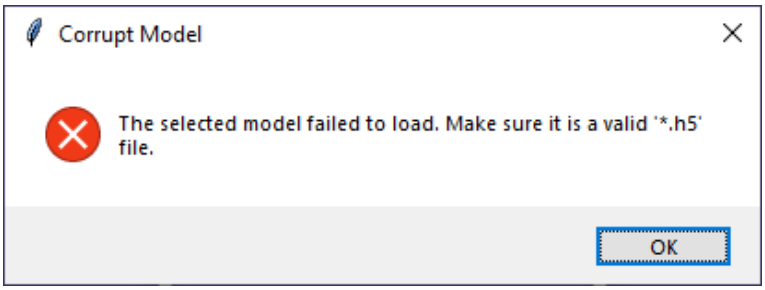


Fig. 17: If the user attempts to upload a pre-trained model of the wrong file type an error window will popup.

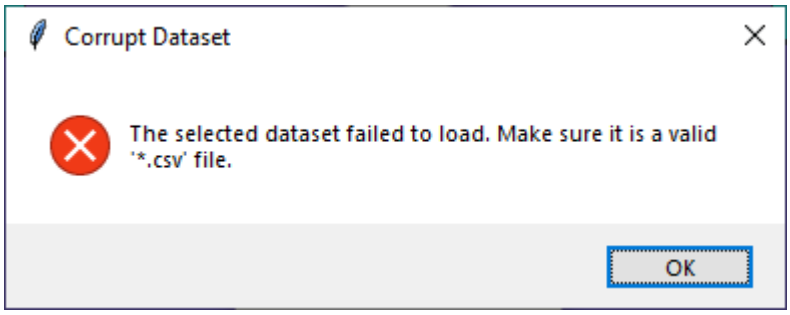


Fig. 18: If the user attempts to import a dataset of the wrong file type an Error window will popup.

5. RESULTS AND CONCLUSIONS

5.1 Results

This section provides the results of different software behaviors when different parameters are tested. As explained in the previous chapters, the model achieves 89% accuracy and 28% loss within only 17 epochs, when 83% of the dataset are assigned for training and 17% for validation (see Fig.19):

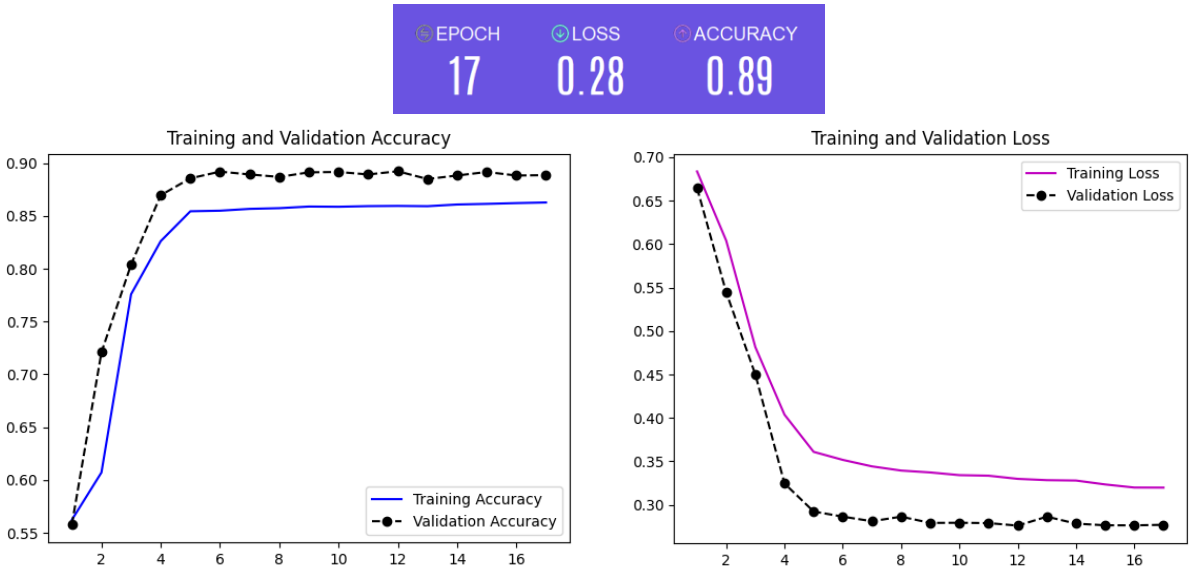


Fig. 19 Our model achivments

A test was made to evaluate the effectiveness of the preprocessing procedure. It is clear that when the model is trained on the preprocessed data (see Fig.20), it receives a higher accuracy rate (Yes_Accuracy) and a lower loss.

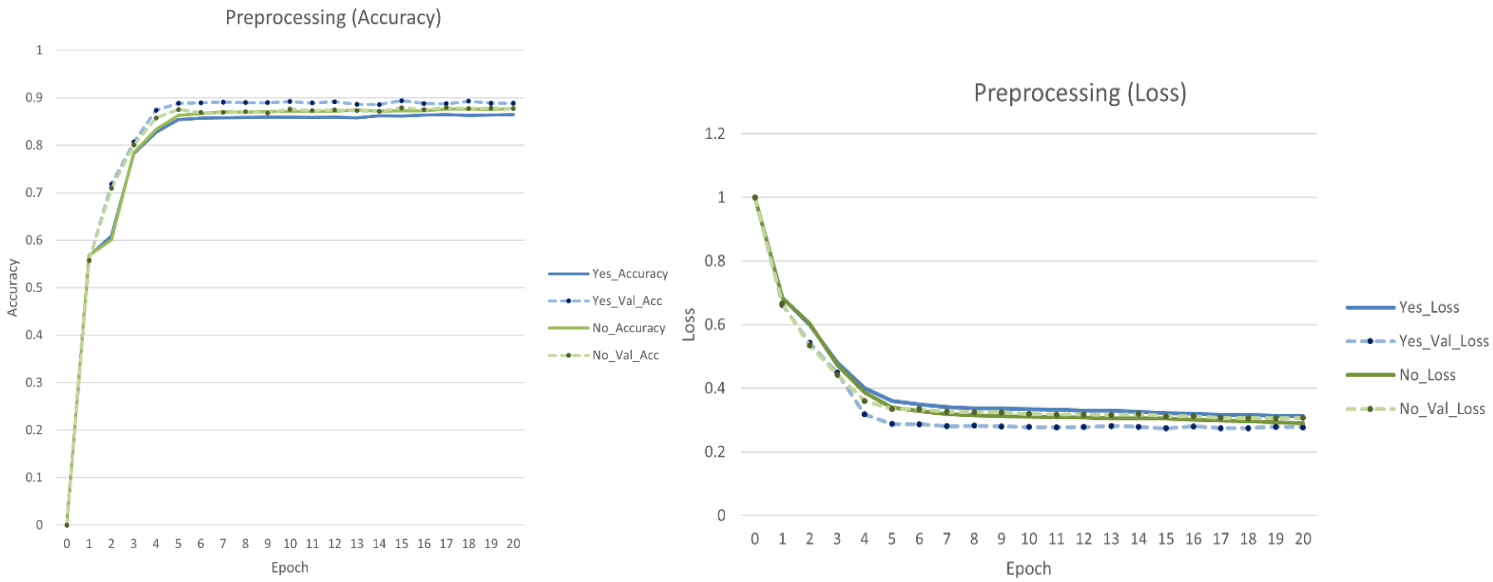


Fig. 20 Evaluation of Preprocessing

Testing different optimization algorithms are essential for finding the one that works best for the system during the backpropagation process. These test results (see Fig.21) show that Adam and the RMSprop achieve a steep learning curve while the SGD does not work well in this case.

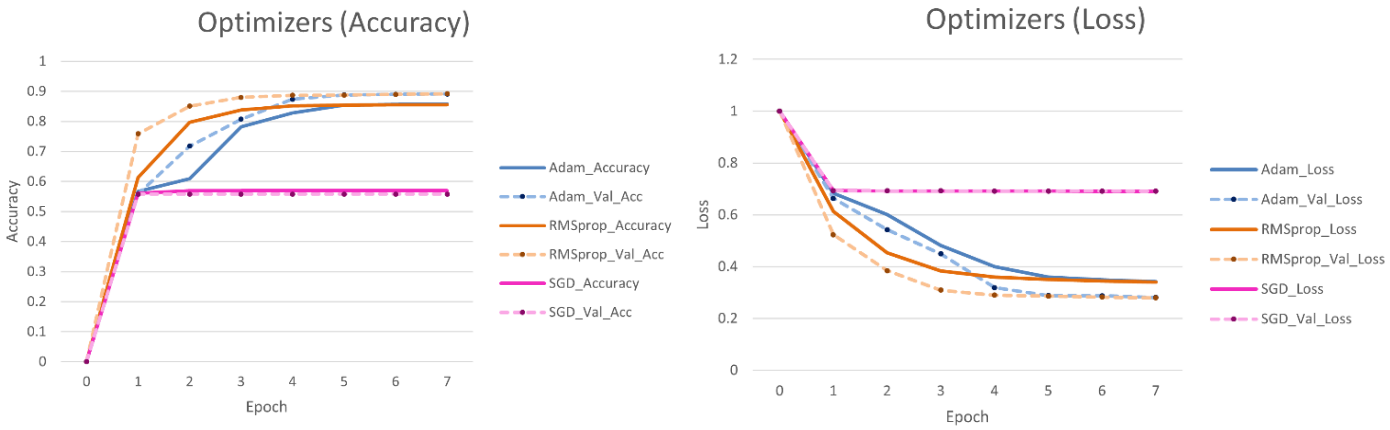


Fig. 21 Evaluation of different optimizers

The original partition sizes are 83% for training and 17% for testing. Understanding how to partition the data correctly to work best for this case is vital and can lead to an earlier stopping and fewer epochs. In the following tests are two different partitions sizes (see Fig.22). As expected, the learning rate is slower as more data is taken from the training set.

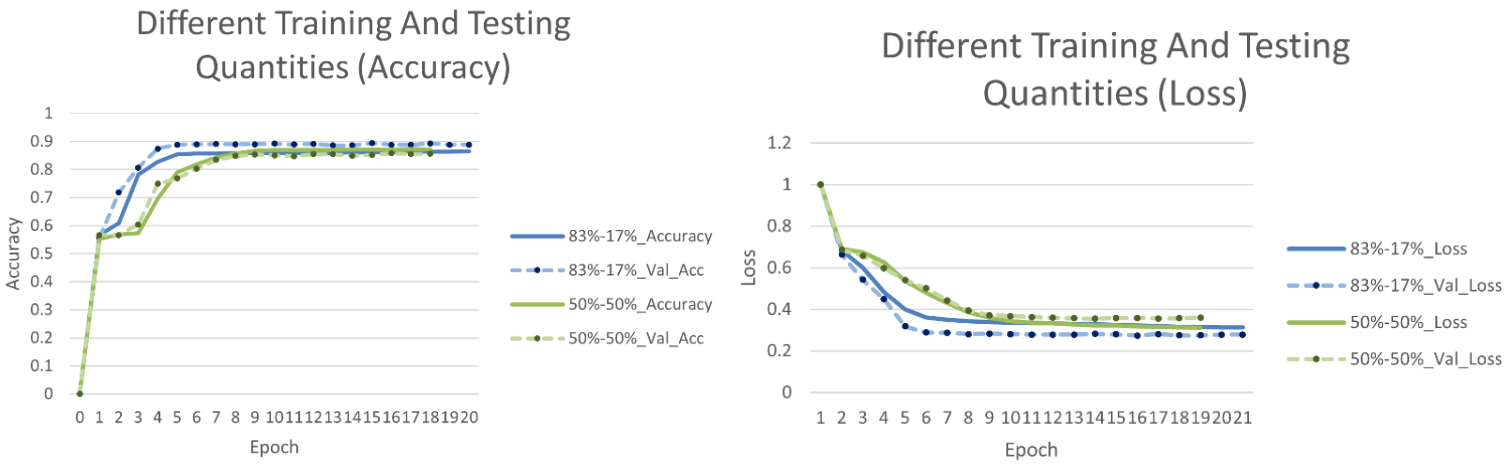


Fig. 22 Evaluation of spliting the dataset differently

Choosing the right activation function can significantly impact the outcome of the model (see Fig. 23). Different activation functions are tested to find the one that works best. It is clear that the sigmoid activation function drastically improves the overall performance of the model.

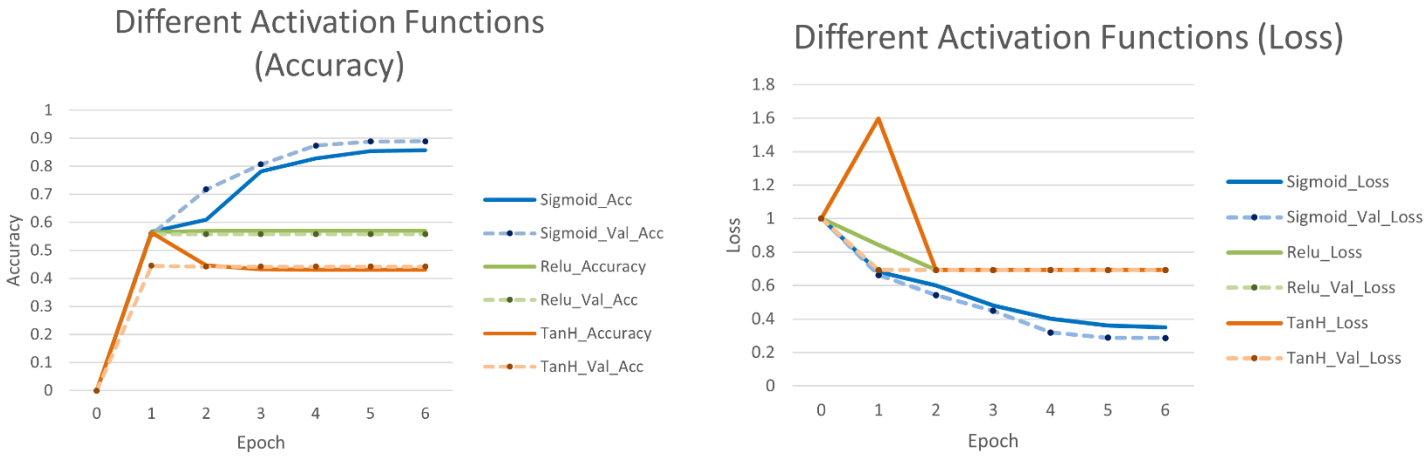


Fig. 23 Evaluation of different activation

For evaluating the effectiveness of the attention-mechanism, the data is only passed through tf-idf and into the model's input, skipping the attention process. From the results of (Fig. 24), it's clear that the attention mechanism provided a performance boost over the attention-less model.

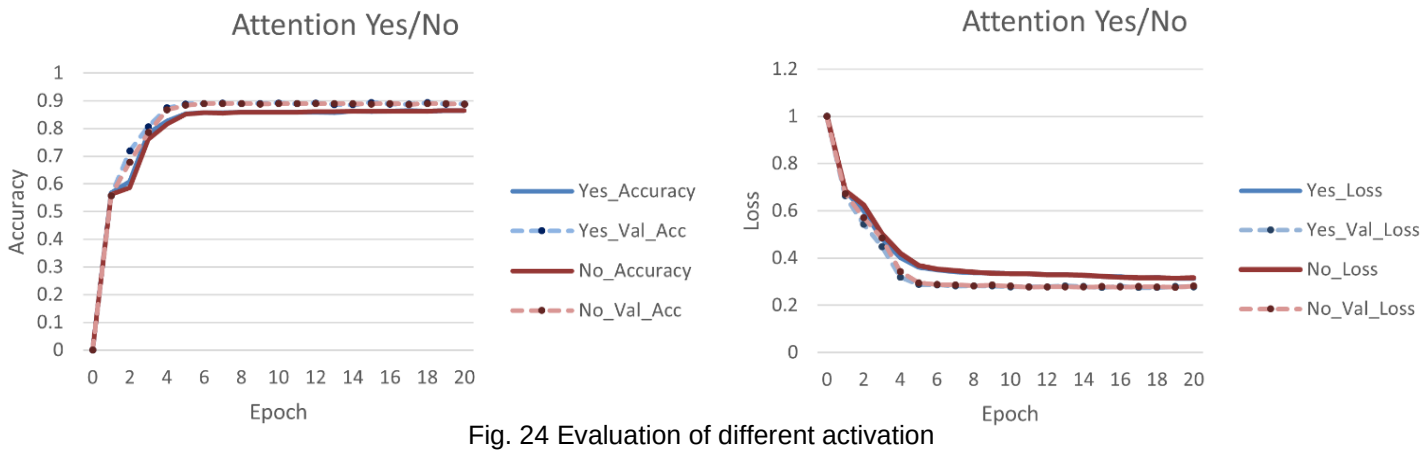


Fig. 24 Evaluation of different activation

The next table summarizes all of the examined parameter differences and their results:

The Examined Subject	Parameters	Validation Accuracy	Validation Loss	Epoch	Results explanation
Preprocessing	Without Preprocessing	0.884	0.310	31	When the model is trained on the preprocessed data, it receives a higher accuracy rate and lower loss within only 15 epochs.
	With	**0.894**	**0.274**	**15**	
Optimization algorithms	**Adam**	**0.894**	**0.274**	**15**	These results show that Adam optimizer achieves a much higher accuracy rate and a lower loss than SGD. The RMSprop optimizer has some good results but not better than Adam.
	RMSprop	0.893	0.279	15	
	SGD	0.557	0.692	2	
Different partition of the dataset	**83% Training 17% Testing**	**0.894**	**0.274**	**15**	When the dataset is split to 50%-50%, the learning rate is slower as more data is taken from the training set, the accuracy and the loss are better in the original version.
	50% Training 50% Testing	0.857	0.355	19	
Activation function	Tanh	0.445	0.693	2	The sigmoid activation function drastically improves the overall performance of the model.
	Sigmoid	**0.894**	**0.274**	**15**	
	Relu	0.557	0.693	2	
Attention mechanism	**With Attention**	**0.894**	**0.274**	**15**	When the attention mechanism is embedded in the system it receives a higher accuracy rate and lower loss and the learning rate is faster (only 15 epochs).
	Without	0.893	0.276	21	

Table 3: Final Results

5.2 Conclusion

The system automatically detects rumors, using a deep attention model based on recurrent neural networks (RNN). A soft attention mechanism is embedded in the system to help focus on specific words for capturing contextual variations of relevant posts over time. To overcome the RNN, which prevents the processing of exceedingly long sequences of texts, LSTM with tf-idf mechanisms are embedded in the system. As it occurs from various tests, data preprocessing can make the learning process more efficient. Furthermore, splitting 83% of the dataset for training and 17% for validation and embedding attention mechanism achieves the best performance of 89% accuracy and 28% loss within only 17 epochs. Also, it is noticeable that Adam and the RMSprop optimization both can fit this system well.

6. REFERENCES

[1] D.E Rumelhart, G.E. Hinton, and R.J. Williams, "Learning representations by back-propagating errors", *Cognitive modeling* 5, 3, 1, 1988.

[2] A. Graves, "Generating sequences with recurrent neural networks", *arXiv preprint*, arXiv: 1308.0850, 2013.

[3] Y. Xiang, Q. Chen, X. Wang, and Y. Qin, "Answer Selection in Community Question Answering via Attentive Neural Networks", *IEEE Signal Processing Letters* 24, 4: 505–509, 2017

[4] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization", *arXiv preprint*, arXiv: 1409.2329, 2014.

[5] A. Graves, "Generating sequences with recurrent neural networks", *arXiv preprint*, arXiv:1308.0850, 2013.

[6] L. Itti, C. Koch, and E. Niebur, "A model of saliency-based visual attention for rapid scene analysis." *IEEE Transactions on Pattern Analysis & Machine Intelligence* 11: 1254–1259, 1998.

[7] R. Desimone and J. Duncan, "Neural mechanisms of selective visual attention." *Annual review of neuroscience* 18.1: 193–222, 1995.

[8] Michael A. Nielsen. "Neural Networks and Deep Learning", *Determination Press. http://neuralnetworksanddeeplearning.com/*, 2015.

[9] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint*, arXiv:1412.6980, 2014.

[10] J. Ma, W. Gao, P. Mitra, S. Kwon, B.J. Jansen, K.F. Wong, and M. Cha, "Detecting rumors from microblogs with recurrent neural networks", *In Proceedings of IJCAI*, 2016.

[11] J. Leskovec, A. Rajaraman, and J.D. Ullman. "Mining of massive datasets", *Cambridge University Press*, 2014.

[12] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch", *Journal of Machine Learning Research* 12: 2493–2537, Aug. 2011.