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

Abstract. *The rise in social media usage in recent years has made it a powerful platform for spreading rumors. They pose a threat to cause a negative influence on people all around the world. Thus, the need for early detection of those rumors is crucial. An authentic way is creating an automatic, self-learning tool, without any intensive manual efforts that will enable any user to detect rumors even within a single post. In the project, the system detects rumors automatically in their early stage of distribution using a deep attention model based on recurrent neural networks (RNN). It can learn continuous hidden representations of contextual variations of posting series. For simplicity, the algorithm is based on a state-of-the-art machine learning algorithm, and it is pre-trained using a couple of different datasets from social media sources. A soft attention mechanism is embedded in the system to help focus on specific words for capturing contextual variations of relevant posts over time. Furthermore, the system enables a trainer to train the algorithm with new datasets in different languages.*
Keywords: Recurrent neural network, Deep attention models, Early rumor detection, LSTM

1. INTRODUCTION

Today's widespread use of social media platforms has made them a place for rumors. The spread of rumors can pose a threat to cybersecurity, social and economic stability worldwide. For example, on April 23rd, 2013, a fake news post claimed two explosions had happened in the White House, and Barack Obama got injured. An unknown hacker posted the report on a Twitter account named Associated Press. The economic damage done by emphasizes rumor caused a loss of $136.5 billion in the stock market even though the White House and the Associated Press quickly denied the rumor. 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.

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. RNNs have been proven to be effective in recent machine learning tasks for handling long sequential data. The attention mechanism can selectively associate more importance with relevant features, which means some features are valued more than others.

The model gets streaming textual sequences of information from posts that are related to one event as an input and constructs a series of feature matrices. Then, the RNN with an attention mechanism automatically learns new and hidden text representations. Furthermore, an additional hidden layer with a sigmoid activation function uses those text representations and predicts, as output, whether an event is a rumor.

1.1 Organization of The Paper

Section 2 presents the theoretical background behind the system implementation. The section starts with describing the recurrent neural network algorithm. It then proceeds to explain about LSTM, Hyperbolic Tangent Activation, Sigmoid, Forget Gate, Input Gate, Output Gate, Cell State, and finally, Attention Mechanism. Section 3 presents the system flow and discusses the early rumor detection with deep attention-based RNN. Section 4 discusses how the neural network training is being performed. Section 5 predicts expected results, and finally, section 6 presents preliminary Software Engineering documents.

2. BACKGROUND AND RELATED WORK

The work is related to early rumors 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.

2.1 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 also extremely limited by the data structure and are relatively slow. Recently, recurrent neural networks were found to be effective in rumor detection, utilizing sequential data to capture temporal textual characteristics spontaneously. However, lacking an abundant 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.

2.2 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 memory of the neural network and remembers a history of the previous steps. Every step, the RNN gets an 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:

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

2.3 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.

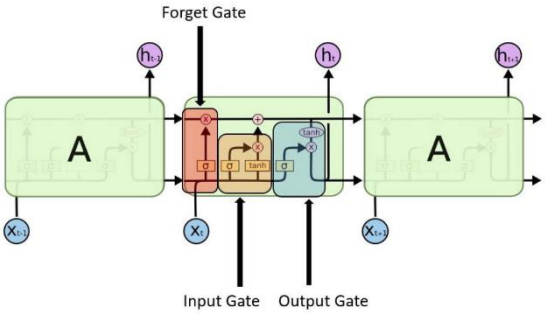


Fig. 1: A visual example of an SLTM with all its gates. The forget gate, input gate, and output gates are highlighted in red, orange, and blue, respectively.

2.3.1 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:

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

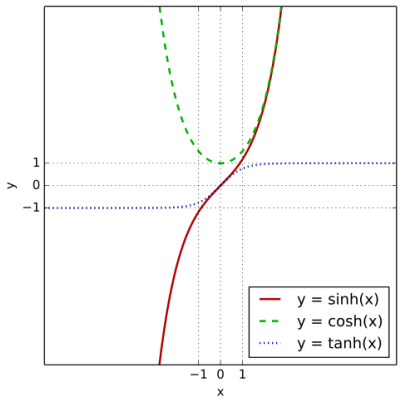


Fig. 2: The graph of the hyperbolic tangent where tanh(x), cosh(x), and sinh(x) are marked with a blue-dotted line, green-dashed line, and a red-solid line respectively

2.3.2 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: $\sigma(x) = \frac{1}{1+e^{-x}}$

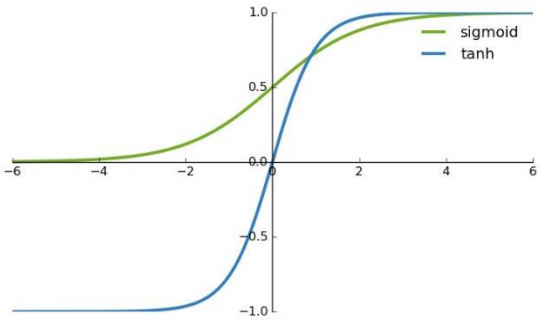


Fig. 3: The graph shows the difference between tanh and sigmoid.

2.3.3 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.

2.3.4 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.

2.3.5 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 $t + 1$.

2.3.6 Cell state

As the current cell state travels along the LSTM module, it is first multiplied by the forget vector. This has a 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 the new cell state.

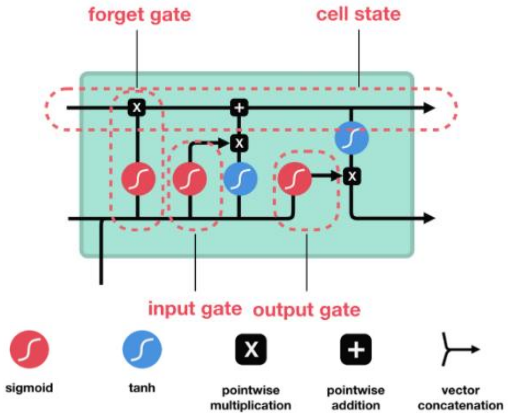


Figure 4: visual example of an LSTM with all its gates.

2.4 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 summarized up to 1. Furthermore, the most significant components correspond to the highest probabilities. The formula defines the standard softmax function:

$$s_i = \frac{(e^{m_i})}{\sum_{j=1}^{n} e^{z_i}} \; for\; i = 1, \ldots, n \; and \; m = (m_1, \ldots, m_n) \in \mathbb{R}^n \qquad (3)$$

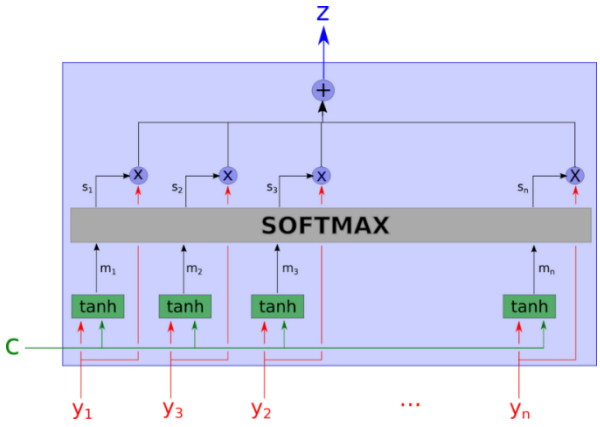


Fig. 5: Visual explanation to how a soft attention mechanism works. Input is entered from the bottom, undergoes some mathematical manipulations and outputs from the top

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.

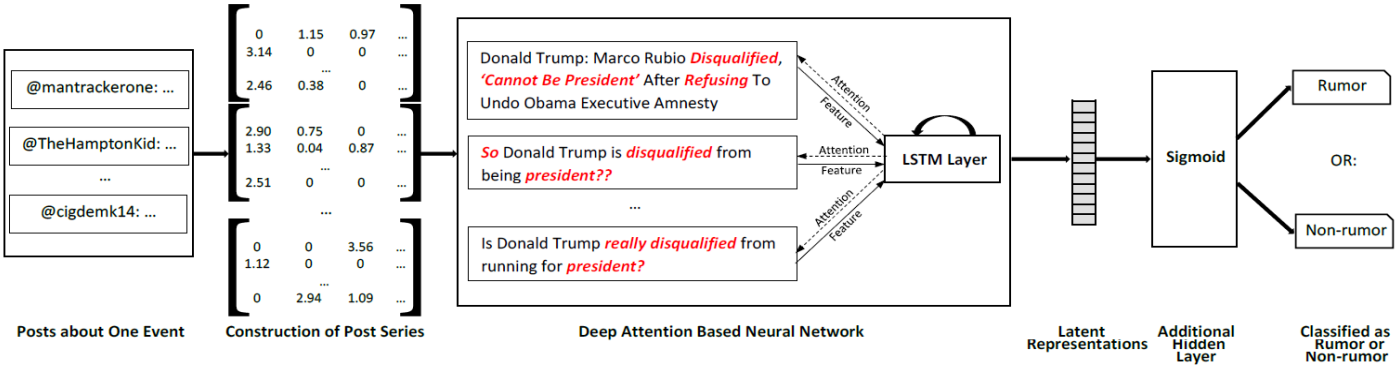


Fig. 6: Schematic overview of the framework

3.1 Construction of Post Series

Individual social posts contain, as usual, limited content. But a more considerable amount of relevant posts regarding one claim can be easily collected to describe the central content more faithfully than a single post [8]. Hence, the system focuses on detecting rumors on the event-level. Sequential posts related to the same topics are batched together to constitute an event, and the model determines whether an event is a rumor or not. $E = \{E_i\}$ denotes a set of given events. For each event E_i a set of relevant post series is the input of the model. Then, algorithm 1 groups posts into batches according to a fixed amount of N posts. This process is called: construction of variable-length post series. More precisely, for every event $Ei = \{(p_{i,j}, t_{i,j})\}_{j=1}^{n_i}$ post series are built with variable lengths due to different amounts of posts. A minimum series length *Min* is set to control the sequential property for all events. For events containing more than $NxMin$ posts, the algorithm iteratively takes the first N posts out of E_i and put them into a time interval T. The last $ni - N\left\lfloor \frac{n_i}{N} \right\rfloor$ posts are treated as the last time interval. For events containing less than $NxMin$ posts, the algorithm puts $\left\lfloor \frac{n_i}{N} \right\rfloor$ posts to the first $Min - 1$ intervals and assign the rest into the last interval $TMin$.

Input: Event-related posts $Ei = \{(p_{i,j}, t_{i,j})\}_{j=1}^{n_i}$, post amount N, minimum series length *Min*

Output: Post Series $S_i = \{T_1, \dots, T_v\}$

```
/*Initialization*/
v = 1; x = 0; y = 0;
while true do
    if n_i ≥ N × Min then
        /* Amount of posts >= N */
        while v ≤ ⌊n_i/N⌋ do
        x = N × (v − 1) + 1;
        y = N × v;
        Tv ← (p_i, x, …, p_i, y);
        v++;
        end
        Tv ← (p_i, y + 1, …, p_i, n_i);
    else
        /* Amount of posts < N */
        while v < Min do
            x = ⌊n_i/Min⌋ × (v − 1) + 1;
            y = ⌊n_i/Min⌋ × v;
            Tv ← (p_i, x, …, p_i, y);
            v++;
        end
        Tv ← (p_i, y + 1, …, p_i, n_i);
    end
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 [9]. In this manner, for each post, 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 matrix of KxN is constructed as the model input.

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

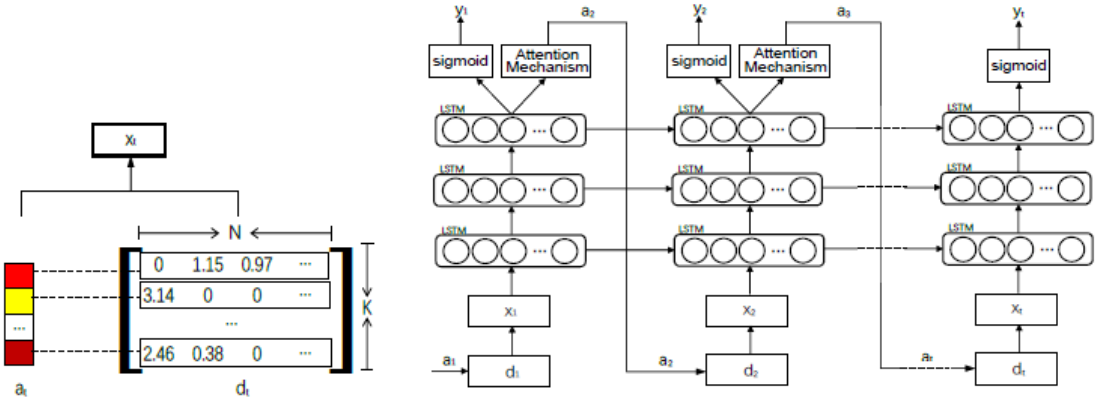


Fig.7: On the left: Image illustrating how a_t is combined with the td-idf matrix (d_t) to form the input x_t. On the right: A visual preview of how the attention based RNN works

In this stage, the Long Short-Term Memory (LSTM) unit is 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, corresponding to the feature extracted at different element positions in a tf-idf matrix d_t is applied.

Next, a_{t+1}, a softmax over K positions, and y_t , a softmax over the binary class of rumors and non-rumors with an additional hidden layer with sigmoid activations (see Fig.7 (right)). The location softmax [10] is applied over the hidden states of the last LSTM layer to calculate a_{t+1}, the attention weight for the next input matrix d_{t+1}

$$a_{t+1}[i] = P(\mathrm{L}_{t+1} = i \mid \mathrm{h}_t) = \frac{e^{\mathrm{W}_i{}^T \mathrm{h}_t}}{\sum_{j=1}^{K} \mathrm{W}_j{}^T \mathrm{h}_t}, \quad (4)$$

where $a_{t+1}[i]$ is the attention probability for the *i*-th element (word index) at time step t + 1, W_i is the weight matrix allocated to the *i*-th element. L_{t+1} is a random variable representing the word index and takes 1-of-K values. The attention vector a_{t+1} is a probability distribution representing the importance attached to each word in the input matrix d_{t+1}. The model puts a higher focus on words suggested to be distinct in learning rumor/non-rumor representations. After calculating these probabilities, the soft deterministic attention mechanism [11] computes the expected value of the input at the next time step $x_{t+1}$by taking an expectation over the word matrix at different positions:

$$x_{t+1}[i] = \mathrm{E}_p\,(\mathrm{L}_{t+1} \mid \mathrm{h}_t)[d_{t+1}] = \sum_{i-1}^{k} a_{t+1}[i] d_{t+1}[i], \quad (5)$$

d_{t+1} is the input matrix at time step t + 1 and $d_{t+1}[i]$ is the feature vector of the *i*-th position in the matrix d_{t+1}. Thus, (4) expresses a deterministic attention model by computing a soft attention weighted word vector $\sum_i a_{t+1}[i] d_{t+1}[i]$. During the backpropagation stages, the system learns where to tweak the weights and the bias to become more accurate as more training occurs.

4. TRAINING

A 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. The loss function is defined as follows:

$$\sum_{t=1}^{\tau} a_{t,i} \approx 1\,,$$
$$\mathcal{L} = -\sum_{t=1}^{C}\sum_{i=1}^{C} \mathrm{y}_{t,i}\,\log \hat{\mathrm{y}}_{t,i} \;+\; \lambda \sum_{i=1}^{K}(1 - \sum_{i=1}^{K} a_{t,i}\,)^2 + \gamma\,\phi^2, \tag{6}$$

y_t is the one hot label vector, $\hat{\mathrm{y}}_t$ is the vector of binary class probabilities at timestamp t τ is the total number of timestamps, C = 2 is the number of output classes (rumors or non-rumors), λ is the attention penalty coefficient, γ is the weight decay coefficient, and ϕ represents all the model parameters.

The cell state and the hidden state for LSTM are initialized using the input tf-idf matrices:

$$\mathrm{C}_0 \;=\; f_c\Big(\frac{1}{\tau}\sum_{i=1}^{K}\Big(\frac{1}{\mathrm{K}}\sum_{i=1}^{K} d_t[i]\,\Big)\Big)\,,$$
$$\mathrm{h}_0 \;=\; f_h\Big(\frac{1}{\tau}\sum_{i=1}^{K}\Big(\frac{1}{\mathrm{K}}\sum_{i=1}^{K} d_t[i]\,\Big)\Big)\;\;, \tag{7}$$

where f_c and f_h are two multi-layer perceptrons, and τ is the number of timestamps in the model. These values are used to compute the first location softmax a_1 which determines the initial input x_t .

5. EXPECTED RESULTS

We expect that applying a new sophisticated deep learning model can provide more accurate results in comparison with other methods. The system's purpose is to learn different representations of posts so that the model will be able to classify posts in an early stage. Due to the adoption of a pre-trained model, and a massive data training set, it is expected to obtain sufficiently good results from the system. In other words, the probability of identifying if an event is a rumor or not is expected to be high. So that at the process finishing a user will be able to identify rumors easily.

6. PRELIMINARY SOFTWARE ENGINEERING DOCUMENTS

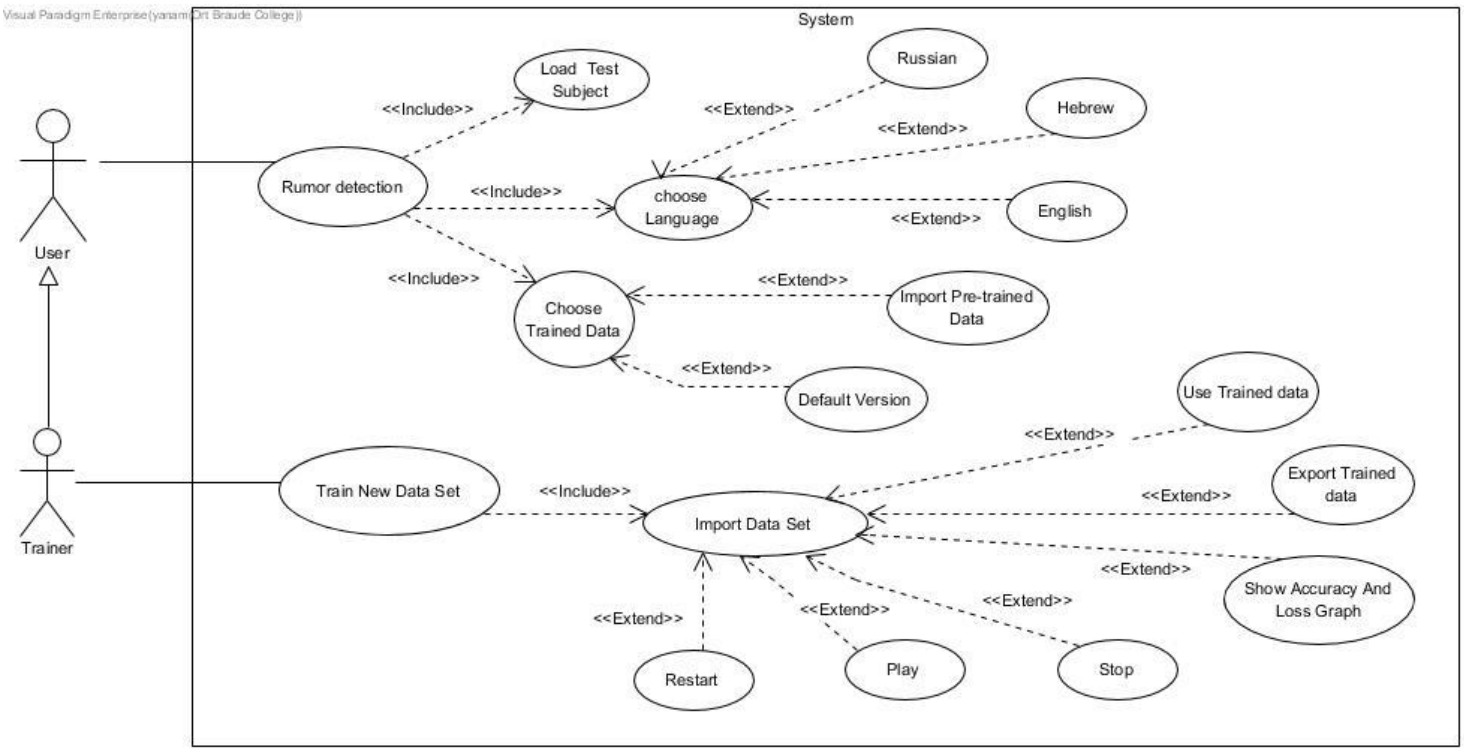
6.1. Requirements and Use-Cases

ID	Requirement
1	The system will classify if the given post (text) is a rumor or not.
2	The user can load one or many texts for the analysis.
3	The user can choose the language and the version of the trained dataset.
4	The system will alert the user if the language and the trained dataset he chose do not match.
5	The trainer can train the model with his dataset.
6	The trainer can save the trained state of the system.
7	The trainer or a user can both use the new trained data for classifying rumors.
8	The system will alert if the input is not in the correct format.
9	The trainer can see the accuracy and the loss graphs of the training process.
10	The trainer can stop / pause / play the training process.

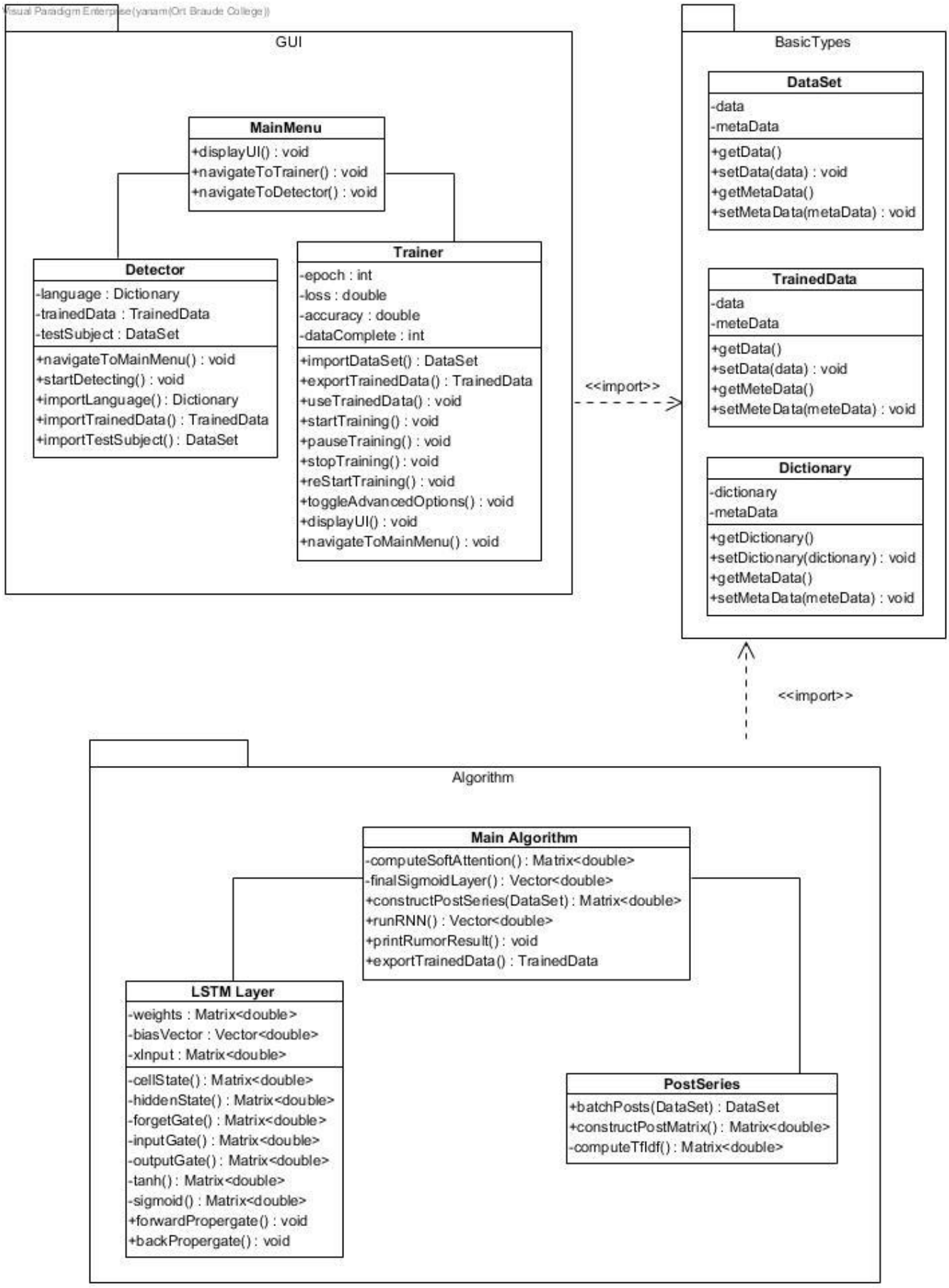
11	The system will show the real-time percentage of the training that has completed.
12	The system will have a default trained dataset in the English language.

6.2. UML Diagrams

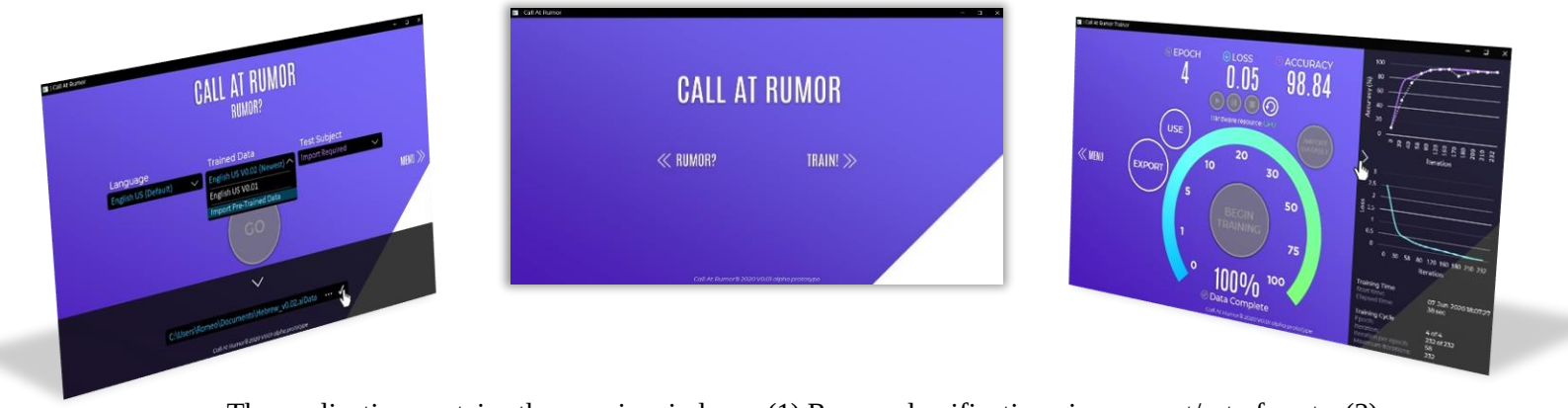
6.2.1. Use case diagram



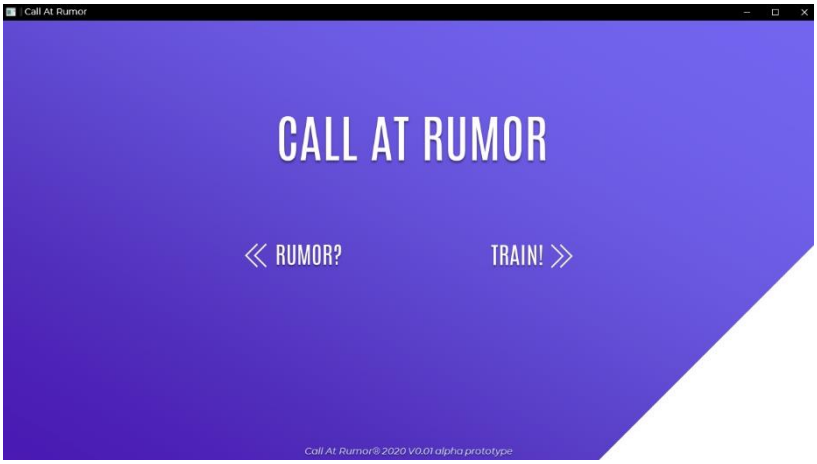
6.2.2. Class Diagram



6.2.3. Graphic User Interface



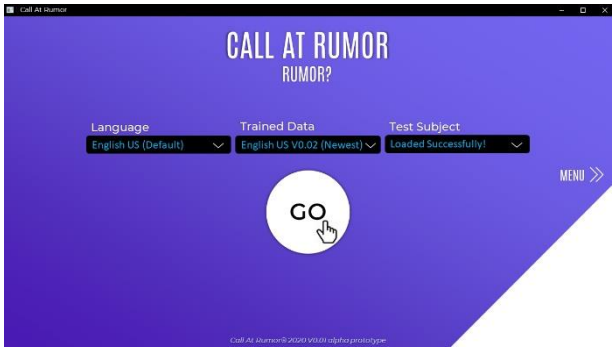
The application contains three main windows: (1) Rumor classification given a post/set of posts. (2) Main menu. (3) Training the model.



*This is the main menu of the application. Here a user can choose if he/she wants to train an RNN 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.*

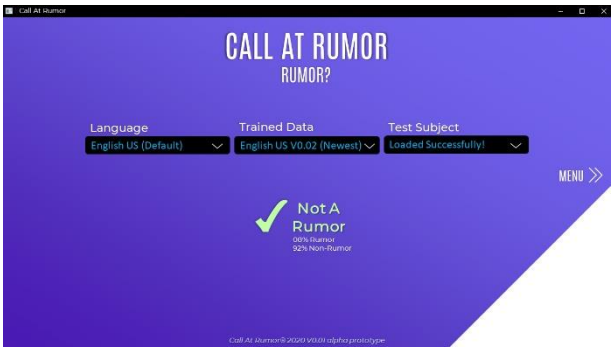
*In the training window, the user can train an RNN with his own dataset! This enables users to train an RNN to work in their local languages as well as the default English language. The user will only need to provide the dataset and click on the **BEGIN TRAINING** button.*

*During 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. The user is also able to view some more raw information about the training process like training times and some training cycle information.*



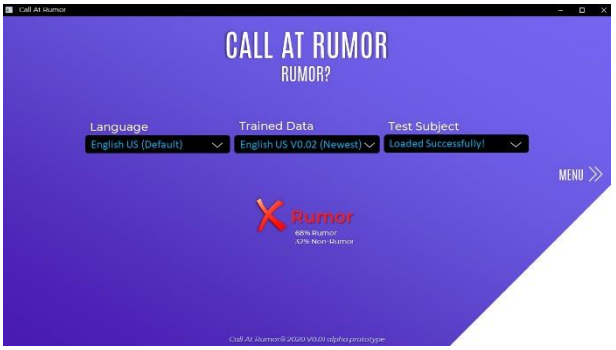
*This is the 'rumor detection' window where the user can test a test subject and find out if it is a rumor or not. First, the user will have to choose the language of the rumor, he/she will then proceed to select his/hers trained data. This is the data for the trained RNN, his test subject will be then entered into this trained RNN for rumor classification. After selecting the correct configurations, the user will enter his/hers test subject, whether it is a single post or a batch of them (the language must match in all of the configuration options). Finally, the user will hit the **GO** button and the classification process will begin.*

Classified as Non-Rumor



The test subject has been classified to be a non-rumor

Classified as a Rumor



The test subject has been classified to be a rumor

6.3 Testing Plan

To check out the system performance, we will run the program on some significant input:

Test No.	The Test subject	Expected result
1.	Wrong input (empty text file, image file)	The system will display an error message.
2.	Save the training output process as a text file	A text file will be created with the output text.
3.	Load a text file that contains a post from social media and press the button "Go."	The system will display "Rumor/ Not a rumor" on the screen
4.	Chose a language and then load a trained data that is not in the same language (Eng. & Heb.)	The system will display an error message.
5.	Press on the button, "Begin training."	The system will display the percentage of the training that completed, the epoch number, accuracy, and the loss.
6.	Press on the button "Advanced."	The system will display the accuracy and loss graphs.

7. 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] 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.

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

[10] S. Sharma, R. Kiros, and R. Salakhutdinov, "Action recognition using visual attention", *arXiv preprint* arXiv: 1511.04119, 2015.

[11] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate", *arXiv preprint* arXiv:1409.0473, 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.