ECOLE POLYTECHNIQUE DE BRUXELLES

Machine Learning and Big Data Processing

ELEC-Y-591

Rumor detection on social media, using recurrent neural network

Authors:

Dumont Jules

Rosso Félix

Simon Antoine

Teachers:

Adrian Munteanu

Nicolaos Deligiannis

Leandro Di Bella

Academic Year 2024-2025

Contents

1 Introduction

In an era where social media platforms have become the primary source of information for millions of people, the rapid spread of false content poses a serious threat to public opinion and trust. Indeed, fake news can have social, political, and even economic consequences. As such, the development of rumor detection systems has emerged as an area of research within the field of machine learning.

This project explores the task of rumor detection on Twitter using deep learning techniques. We focus on the implementation and evaluation of the CSI model [**CSI**] (Capture, Score, Integrate), a hybrid neural network architecture that uses both the temporal dynamics of tweet propagation and user credibility features. Through the integration of text characteristics, user behavior, and temporal patterns, the model aims to accurately classify Twitter threads as rumors or non-rumors.

Our work includes a review of the current state-of-the-art in rumor detection, the collection and processing the dataset, the implementation of the CSI model as well as a simplified baseline, and an evaluation of their performances on real-world data.

2 Scientific Literature Review

In this chapter, we review three significant contributions to this field: (1) use temporal patterns characterization of rumors propagation, (2) integrate a user score module, and (3) a summary of the existing datasets and the different methodologies.

1 Information Credibility on Twitter

The first study [**rumor3**] presented in this report, published in 2011, focused on determining the credibility of information shared on Twitter. The authors employed machine learning techniques, namely decision trees, to classify the credibility of tweets. They extracted the features from the textual content of tweets : the text characteristic like sentiment indicators, the presence of URLs and hashtags as well as user-based features like the number of followers. Their findings demonstrated that these features could effectively distinguish between credible and non-credible information.
This paper is among the earliest to apply machine learning to the problem of information credibility on social media. While this paper did not focus exclusively on fake news propagation, it started the groundwork for future research by highlighting the potential of textual patterns recognition in fake news detection.

2 RNN-Based Detection from Microblogs

This paper [**rumor**], published in 2016, proposed to use recurrent neural networks (RNN) for detecting rumors on microblog platforms. The step forward presented in this study is to also characterize temporal patterns of rumors propagation, in addition to textual patterns. Therefore, the model processes a sequence of answers to a post related to a particular subject and learns to distinguish rumors.
In practice : we model the events as variable-length time series, where each step adds the pub-

lished posts over a short time intervals. Then, the features are inputted in a RNN like a Long Short Term Memory (LSTM) or a Gated Recurrent Unit (GRU) neural network.
This model outperformed traditional only text feature approaches and excels at early detection, often within a few hours of a rumor's emergence.
This paper marked one of the earliest uses of deep learning for rumor detection, using temporal modeling of misinformation propagation.

3 CSI: A Hybrid Deep Model for Fake News Detection

This paper [**CSI**], published in 2017, digged more into the idea of characterizing the credibility of the users. It is well known that some users are more likely to spread misinformation than other and sometimes presents group behavior which we can use to recognize rumors. In practice, we integrate a new module with the previous one : the user score module. Therefore, the model consists of three modules :

- **Capture:** This module, based on the previous paper [**rumor**], employs a RNN Long Short-Term Memory (LSTM) network to model the temporal patterns of user interactions with news articles, incorporating both engagement timing and associated textual content.
- **Score:** Here, a neural network assigns a suspicion score to users based on their behavior, representing their tendency to participate in the propagation of fake information.
- **Integrate:** The outputs from the Capture and Score modules are combined to make a final classification of the article's veracity.

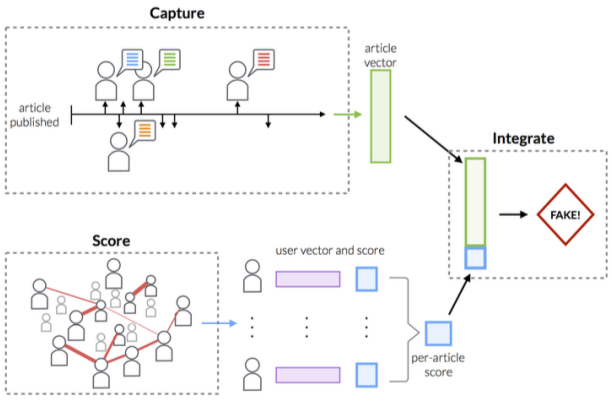

Figure 2.1: Intuition behind CSI.

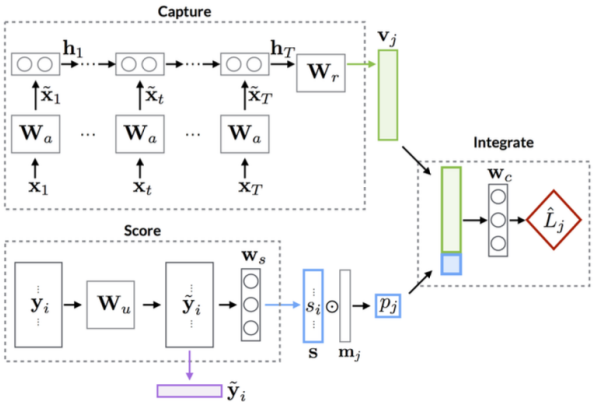

Figure 2.2: Representation of the NNs used.

The strength of the CSI model lies in its integration of information, allowing to avoid the use of social network graphs which are difficult to implement. It showed better performances on real-

world datasets from Twitter and Weibo, outperforming previous models which uses a single type of feature input.

3 Implemented Model

1 Neural Network Models Implementation

Detecting rumors on social media is a complex task that cannot rely solely on the textual content of posts. Instead, a more robust approach involves modeling the temporal patterns of information diffusion and incorporating user behavior and credibility This idea is at the core of the CSI model introduced in [**CSI**], which we implemented and studied in this project. The model decomposes rumor detection into three complementary sub-tasks: capturing article propagation, scoring users based on credibility, and integrating these signals to produce the final prediction.

We implemented and compared two architectures:

- The **CSI model (Capture-Score-Integrate)**, which explicitly models time-series propagation, user trust, and combines both.
- A **Simple Baseline model**, using only temporal features extracted from the article, we omit here the score module.

In what follows, we detail the components of the CSI architecture.

1.1 Capture Module: Modeling the Propagation Dynamics

Each article shared on a platform gives rise to a sequence of user engagements over time (engagements can be see as the number of comments/reply tweet). The Capture Module is designed to extract temporal patterns from this sequence, as well as its textual information (see chapter**??** explaining data extraction). To this end, it uses a neural architecture composed of:

- A linear embedding layer to project the raw engagement features at each time step into a latent space.

- A dropout layer to reduce overfitting during training.
- A recurrent LSTM layer to model temporal dependencies between user engagements. Its last hidden state is then inserted into the last layer.
- A fully connected layer to produce a fixed-size vector representation v_j of the article,itself passed into hyperbolic tangent activation.

This process can be interpreted as "listening to the pulse" of the article by observing how engagements evolves over time. For instance, a rumor may exhibit sharp spikes in sharing patterns or reach many low-credibility users quickly — patterns the LSTM can potentially detect.

```
def __init__(self, dim_x_t, dim_embedding_wa, dim_hidden, dim_v_j):
    super(CaptureModule, self).__init__()
    self.embedding_wa = nn.Linear(dim_x_t, dim_embedding_wa)
    self.dropout_wa = nn.Dropout(p=0.5)
    self.rnn = nn.LSTM(dim_embedding_wa, dim_hidden, batch_first=True)
    self.fc_wr = nn.Linear(dim_hidden, dim_v_j)
    self.dropout_wr = nn.Dropout(p=0.5)
```

1.2 Score Module: Evaluating User Credibility

The Score Module assigns a credibility score to each user based on their profile features (e.g., account age, follower count, average behavior or behavior in a group for our case). Intuitively, this reflects the idea that some users are more likely than others to propagate false information.

The module architecture is straightforward but effective:

- A first linear layer embeds the user feature vector into a latent representation and passes it into hyperbolic tangent activation.
- A second linear layer outputs a scalar score, later passed through sigmoid activation to obtain a probability-like credibility estimate.

This score is essential for understanding who are the users spreading the information, complementing the Capture Module's focus on how it spreads.

```
def __init__(self, dim_y_i, dim_embedding_wu):
    super(ScoreModule, self).__init__()
    self.user_fc = nn.Linear(dim_y_i, dim_embedding_wu)
    self.score_fc = nn.Linear(dim_embedding_wu, 1)
```

1.3 Integrate Module: Combining Source and Signal

Once both article-level (temporal) and user-level (credibility) features are extracted, the Integrate Module merges these signals to produce a final prediction.

- The credibility scores s_i of all users who engaged with the article are aggregated to get a global user trust score.
- This scalar is concatenated with the article representation v_j from the Capture Module.
- Finally, the linear layer maps this combined feature vector to a prediction $\hat{y}_j \in [0,1]$, indicating whether the article is a rumor.

```
def __init__(self, dim_v_j, user_scores_dim=1):
    super(IntegrateModule, self).__init__()
    self.fc = nn.Linear(dim_v_j + user_scores_dim, 1)
```

1.4 Complete CSI Model Architecture

The full CSI model simply instantiates and connects the three modules described above. Its forward pass follows this logic:

1. The sequence of article features x_t is passed through the Capture Module to obtain v_j.
2. User features y_i are passed through the Score Module to obtain scores s_i.
3. These scores are aggregated per article based on the engagement matrix m_j.
4. The aggregated user score is concatenated with v_j and passed through the Integrate Module to output the final prediction.

```
def __init__(self, dim_x_t, dim_embedding_wa, dim_hidden,
             dim_v_j, dim_y_i, dim_embedding_wu):
    super(CSI_model, self).__init__()
    self.capture_module = CaptureModule(dim_x_t, dim_embedding_wa,
                                        dim_hidden, dim_v_j)
    self.score_module = ScoreModule(dim_y_i, dim_embedding_wu)
    self.integrate_module = IntegrateModule(dim_v_j)
```

2 Training

The training of the CSI (Capture-Score-Integrate) model is managed by a dedicated `Trainer` class, which coordinates the entire learning workflow from data preprocessing to model evaluation. Preprocessing contains steps such as balancing classes in the dataset (equal amount of rumor/non-rumour threads) or standardization of the 2 first features η and Δt (described at **??**). The standardisation is in 2 steps. First we apply the $log1p$ function then we apply the z-score using means (μ) and standard deviation (σ) **of the training dataset** :

$$Fetaure \rightarrow \frac{log1p(Feature) - mean}{\sigma} \tag{3.1}$$

The training is performed over a fixed number of epochs (default set to 20). Each epoch consists of a training phase (80% of dataset) followed by a validation phase (20% of dataset):

- **Training Phase:** The model is set to training mode to enable gradient updates. For each batch:
  - A forward pass computes predictions as well as intermediate user and article representations.
  - The loss is calculated combining binary cross-entropy with L2 regularization.
  - Backpropagation is performed to compute gradients.
  - Gradient clipping with a maximum norm of 5.0 is applied to prevent exploding gradients. A `ReduceLROnPlateau` scheduler lowers the learning rate when the validation loss plateaus, enabling finer convergence.
  - The optimizer updates the model parameters using Adam with an adaptive learning rate.
- **Validation Phase:** The model is switched to evaluation mode, disabling gradient computation for efficiency. The validation loss is computed over the validation batches to monitor generalization and check that the model isn't overfitting.

4 Data Collection and Processing

We first used the PHEME rumor dataset [**pheme**], which contains Twitter threads labeled as either rumors or non-rumors. However, due to the limited amount data, the results were inconclusive, and we therefore used another similarly structured dataset: **Weibo Dataset**. It seems it was initially constructed by [**rumor**] and it can be found at [**dropurl**].

1 Dataset Structure

The dataset consists of tweet threads organized by events, where each thread[1] contains a source tweet and multiple reaction tweets. In the dataset, each thread is labeled as either rumor (1) or non-rumor (0). Our implementation encapsulates this processing in the `DataBase` class, which performs the following operations:

1. Parse all JSON files containing tweet data
2. Extract temporal information (get the time stamp associated to the tweets, create the bins and sorting tweets in them)
3. Compute text embeddings for all tweets
4. Build user representation vectors
5. Construct temporal sequences for each thread

2 Text Representation

For text representation, we employ a pre-trained sentence transformer model (`all-MiniLM-L6-v2` or `paraphrase-multilingual-MiniLM-L12-v2`) to embed tweet content into 384-dimensional

[1] Note that sometimes threads are called articles as it is the case in [**CSI**]

vectors. To reduce computational complexity while preserving semantic information, these high-dimensional embeddings are compressed using Gaussian random projection.

3 User Representation

User Representation via Matrix Factorization

Our approach leverages matrix factorization techniques to create two complementary user representations:

Global User Vectors ($\mathbf{x}_u$) We construct these vectors from a binary user-thread incidence matrix $M \in \mathbb{R}^{n_{users} \times n_{threads}}$:

$$M_{ij} = \begin{cases} 1 & \text{if user } i \text{ engaged with thread } j \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

In practice, we compute the truncated singular value decomposition (SVD), as it is done in [**CSI**], to extract the user vectors $x_u^{(i)}$.

Source-level User Vectors ($\mathbf{y}_i$) These vectors are derived from a user co-engagement graph with adjacency matrix $A \in \mathbb{R}^{n_{users} \times n_{users}}$:

$$A_{ij} = \text{number of threads both users } i \text{ and } j \text{ engaged with} \tag{4.2}$$

Similarly, we apply SVD and extract the user credibility vectors y_i.

4 Temporal Sequence Construction

For each thread, we construct a temporal sequence aligned with the CSI model's requirements:

1. Tweets are sorted chronologically and binned into hourly intervals relative to the source tweet
2. For each non-empty bin, we compute:
    - η: Number of reactions in the current bin
    - Δt: Time elapsed since the last non-empty bin

- $\mathbf{x}_u$: Average user vector of users engaging in the current bin
- $\mathbf{x}_\tau$: Average text embedding of tweets in the current bin

5 Dataset Preparation

The processed data is organized into a PyTorch dataset structure with the following components:

- Article features: Temporal sequences of $[\eta, \Delta t, \mathbf{x}_u, \mathbf{x}_\tau]$ for each thread
- User features: Matrix of user credibility vectors $\mathbf{y}_i$
- Labels: Binary indicators of rumor status

5 Results

This section presents the results we obtained during the training of models with differents configuration. The different hyperparameters choosed are the following:

- *SeqLen*: the length of the input sequence of the LSTM.
- *DimVJ*: the dimension of the vector v_j (output of the capture module).
- *DimH*: the dimension of the hidden parameter in the LSTM.
- *LR*: the learning rate at epoch 1.
- *RegAll*: the possibility to switch to regularisation to all the learning parameters if True.

The main objective was to evaluate the performance of the proposed CSI model and compare its effectiveness with a simplified version, the 'Simple_CSI_model' in our code (can also be seen as CI for capture and integrate), which omits the user score module. As mentioned in **??** the tests were done on 2 different datasets each different sizes that we report on the following table:

Table 5.1: Dataset Statistics

Metric	**Weibo**	**PHEME**
Number of Tweets	3,805,656	102,440
Number of Events/Threads	4,664	5,802
Number of Users	2,856,741	49,345

Our results are manly based on the accuracy with respect to the validation set recorded via TensorBoard. With the long time of training, only a few set of different configurations have been tested. Also it is noted that for the sake of readability and interpretability of the results, only the best shots were kept.

1 Results on PHEME Dataset

The analysis of logs allow us to identify the maximum accuracies achieved by the different model configurations on the validation set (PHEME). Table **??** summarizes the best performances for the full CSI model and the 'Simple_CSI_model' (CI), as well as the associated hyperparameters.

Table 5.2: Best Validation Accuracies and Associated Hyperparameters for PHEME Dataset

Model	Accuracy	SeqLen	DimVJ	DimH	LR	RegAll
PHEME CSI	78.20%	20	100	50	0.001	False
PHEME CSI	77.06%	20	100	50	0.001	True
PHEME CSI	76.93%	100	100	50	0.001	False
PHEME CSI	76.68%	15	100	50	0.001	False
PHEME CSI	76.30%	20	50	25	0.001	False
PHEME Simple CSI (CI)	75.16%	20	100	50	0.001	False

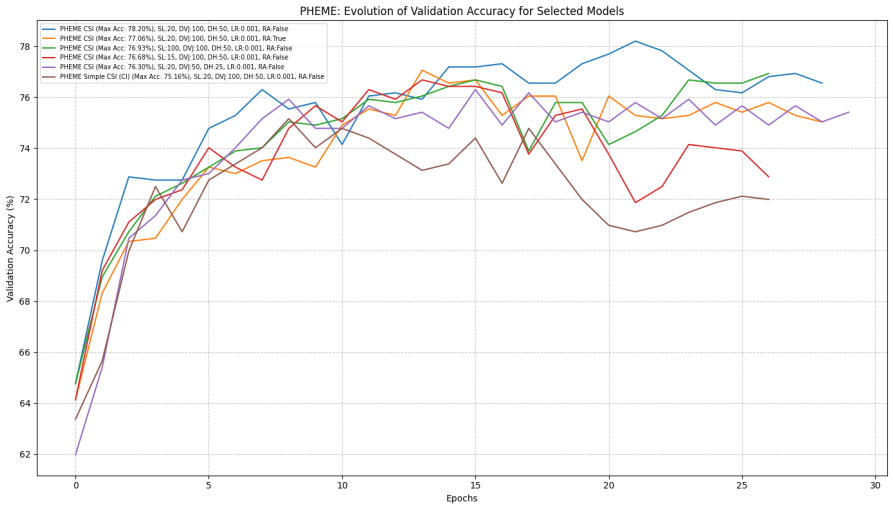

Figure 5.1: Evolution of validation Accuracy for Models trained on PHEME Dataset

As the table shows, the highest validation accuracy for the full CSI model is **78,2%**. We can note a slighlty better performance with the score module than without (CI peak at 75,1%) but as we will see in the next section it might be a coincidence. Furthermore, the final accuracy doesn't match at all the accuracy obtained by [**CSI**]. This is the reason why we tried a different approach using another dataset.

2 Results on Weibo Dataset

Table **??** summarizes the best performances for the full CSI model and the 'Simple_CSI_model' (CI) on the Weibo Dataset, as well as the associated hyperparameters.

Table 5.3: Best Validation Accuracies for Weibo Dataset

Model	Accuracy	SeqLen	DimVJ	DimH	LR	RegAll
Simple CSI (CI)	89.42%	100	100	50	0.001	False
CSI	89.20%	100	100	50	0.001	True
Simple CSI (CI)	88.55%	100	150	100	0.001	False
CSI	88.34%	100	100	50	0.0005	False
CSI	88.23%	100	100	50	0.0005	True

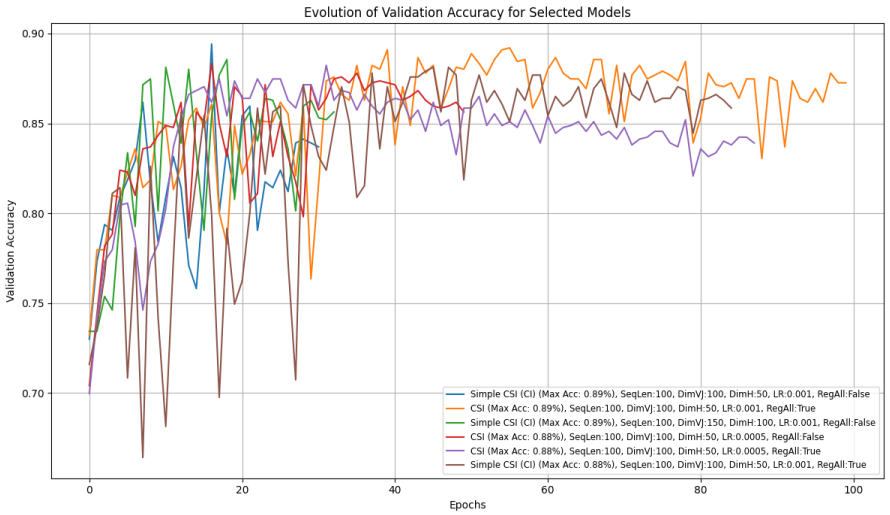

Figure 5.2: Evolution of Validation Accuracy for Models trained on Weibo Dataset

First we can clearly see that the performance are increasing of 10% in accuracy compared to PHEME Dataset, confirming it was a dataset problem. Secondly, we notice that the 'Simple_CSI_model' (CI), which does not take into account the user score module, achieves an equivalent accuracy than the full CSI model. It suggests that, in the tested configurations and implementation, the integration of the score module has not been succesfull, and didn't brought any significant advantage in terms of performances. This might be due to an unknown default in the implementation or the need of a different configuration. If it is the second case scenario, a solution could be to make the Score module to have its own training branch: in the current implementation the final score p_j is trained along with the vector v_j which dimensions are 100 times bigger. The backpropagation might favor the update of v_j only. If the score module had its own branch of training the model could learn more form the score module but it is again only an hypothesis

Figure **??** illustrates the evolution of validation accuracy over training epochs for the selected model configurations presented in Table **??**. This graph provides an overview of the convergence and stability of these models. We can see that our models seems to converge and stabilize after 40/60 epochs.

6 Conclusion

This project tested the potential of a hybrid deep learning architectures for fake news detection on social media. By implementing and training the CSI model, we showed how combining temporal patterns of information propagation with user credibility scoring can produce a good understanding of rumor dynamics. While the CSI model achieved promising results, our experiments revealed that the simplified version (excluding the user score module) performed similarly, suggesting limitations in the current implementation of the user score module.

These findings suggests several possible improvements for future work, such as refining the training strategy for the score module or testing on larger and more diverse datasets to better capture user interaction networks. Overall, this project provides a second "proof" of the accuracy of the CSI model, and shows there is still room for improvement.