MACHINE LEARNING AND BIG DATA PROCESSING

PROJECT REPORT

Recurrent Neural Networks for rumor detection on social media

Professors:
Prof. Adrian MUNTEANU
Prof. Nikolaos DELIGIANNIS

Authors :
Jolan HUYVAERT
Théo LISART
Logan SIEBERT

Academic year 2019-2020

Contents

List of Figures

1 Introduction

The spread of misinformation on social media can lead to harmful situations, people manipulation and social unrest. With the constantly increasing number of users on social media, fake news propagation becomes more recurrent and faster. This naturally led to a great amount of work in this field. The aims of this project is to study and implement rumor detection on social media. One can usually describe a rumor as an event associated with a given set of posts from different users such that the textual features can be extracted from the concatenation of these posts. It is worth noting that other features than the textual ones can be used for a better model accuracy. For instance, by including user related features to account for the source reliability. The challenging problem of early automated rumor detection turned out to be well suited to deep neural networks as they are able to model the complex data dependencies in social media events streams. In this project, we will classify if an event is a rumor or not based on their textual features and by means of recurrent neural networks.

2 State-of-the-Art

2.1 Using Recurrent Neural Networks

State-of-the-art rumor detection algorithms exploits the advantages provided by neural networks to model the deep data dependencies. The possibility to early extract rumor post streams pattern trough a recurrent neural network is explored in [1]. They consider a post and all its re-posts as an event. Recurrent Neural networks being well suited to capture these dependencies in sequential data streams. The novel approach is based on the extraction of the statistical textual features used in natural language processing such as TF-IDF. By properly formatting labeled data from Twitter and Weibo, the authors solve the classification problem of an event (fake or not) by considering the evolution of an information signal as a varying length time series. This approach thus learns both temporal and textual representations of rumor events. Considering an event-related post, the users comments stream are fed-forward as temporal series. This article shows that RNNs are more precise and stable at recognizing rumors dynamics on microblogs than previous established learning algorithms.

2.2 Using hybrid deep learning networks

The first approach is mainly based on textual features but [2] shows that it is possible to simultaneously train a fully-connected neural network to account for the even-related post's source. These two model outputs are then combined for classifications. This

learning approach lead to a CSI (Capture, Score, integrate) model based on a combination of signals taken from the *text* and the general *response* stream as previously and combine this results with the learned *source* rating scores. This approach also exploits deep learned posts representation using *doc2vec* features extraction.

3 Implementation

The publicly available dataset[1] from Weibo and used by [1] is used throughout this project. The primary step would then be to properly work with the data before focusing on the textual features extraction from posts. Indeed the data fed to the RNN is firstly partitioned in order to reduce the computational cost as multiple posts can then rely on an event, that is fed to a RNN unit. This is explained in section 3.1. Then the feature extraction methods are briefly presented and the chosen method implementation is detailed in section 3.2. The classification problem based on these textual features and outputting the rumor label, is presented in section 3.3. Finally, section 3.5 presents the model training, including the learning parameters and the dataset presentation.

3.1 Time series construction

In order to reduce the computational cost of the back-propagation over large amount of time steps by treating each post as an input to a RNN unit, the posts are batched into time intervals. In addition, social media posts are usually very short and thus containing low information. As seen in section 3.2, the interval will be seen as a super document containing the information belonging to its time span. This partitioning should be able to capture properly the densely populated time spans of the events. Thus instead of using fixed length time series, a variable length time series construction is adopted as proposed in [1]. This works as follows:

- The entire timeline is initially divided in N time intervals spanning the same amount of time l.
- Look for the non-empty continuous interval covering the longer time span and discard the other intervals
- If the the number of intervals is less than N and more than for the previous iteration, reduce l by a factor $1/2$ and loop.

Dense time spans of each event are thus partitioned uniformly to approximate the RNN reference length N. Events with dense time spans will tend to have less intervals than

[1] `http://alt.qcri.org/~wgao/data/rumdect.zip`

N and sparse events will tend to have more intervals than N. The detailed algorithm is presented in appendix B. A value of N=10 was adopted in this project.

3.2 Textual features extraction

3.2.1 Statistical features

Here are presented the statistical text features derived from natural language processing. The idea of these methods is to extract the topic from a collection of text documents by weighting the words importance in a document. These techniques can be used standalone or combined with other kind of relevant features. Only textual features from the posts texts are considered in this project.

Natural language processing techniques can be used to extract features from a text . The Bag-of-word (BoW) model treats the importance of a word by counting its occurrence in each document in a collection of N documents. This results in a orderless bag-of-words representing the topic and used as features. Orderless means here that the representation is not dependent on the words order history. These results are then normalized to get the word's frequency of occurrence before being fed to a neural network [3]. This frequency of occurrence relates the importance of a word in a collection of documents.

The most frequent words are unlikely to be the most significant and rare words might yield great importance. Rare words can have a high significance related to the topic while very frequent words tends to have no importance. The term frequency–inverse document frequency (TF-IDF) take accounts for this offset. The TF for a collection of N documents can be computed as follows [4]:

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}} \tag{1}$$

where f_{ij} is the frequency of occurrence of a word i in document j. The frequency of occurrence is thus normalized with the maximum number of occurrences of any term in the same document. The most frequent term in a document has thus a maximum TF of 1. As the number of documents is increased an offset need to be adjusted to account for the frequent words increase. In this way, the significance of rare words is preserved. It is achieved trough the IDF for a collection of N documents. It is computed as follows [4]:

$$IDF_i = \log(N/n_i) \tag{2}$$

where n_i is the number of documents in which the word i occurs. The TF-IDF score is then computed with:

$$TF.IDF = TF_{ij} \times IDF_i \tag{3}$$

The TF-IDF method is computed for each event seen as a collection of documents. Each document being the time interval containing all the corresponding post texts. First the TF is computed for each interval and then multiplied by the IDF, computed for the whole vocabulary. Prior to this step, the text is pre-processed to remove numbers, emoticons, symbols , punctuation, and non Chinese alphabet words with the *re* package. An additional function *ChineseAnalyzer* from the *Chinese* package [2] is used to tokenize the Chinese words as they are usually not separated by spaces in a sentence. Then only the K most important TF-IDF values are kept for each interval resulting in a vector of dimension K to be fed to each RNN cell. As the number of interval is variable, all the events are padded with 0's to reach the pre-defined RNN reference length N=10. The TF-IDF extractor *TfidfVectorizer* [3] from the scikit-learn toolkit was used to check our implementation.

3.2.2 Deep learning based features

Besides the presented textual features extractor, deep learning approaches that outperforms simpler models for large datasets can be used to learn efficient words or documents vector representations.

For instance, the *word2vec* word embedding method introduced in [3] can account for different degrees of similarity between word by learning their context. This NN thus embeds closely related words in a similar way in the features vector space. The continuous bag-of-word model (CBOW) and the skip-gram model are the two architectures introduced in [3] and are based on a probabilistic neural network first introduced in [5] that learns the joint probability function of sequences of words. This methods is not really efficient for our problem as it increases drastically the feature sizes by treating each word in each interval as a vector. The *doc2vec* is an extension of *word2vec* allowing to construct paragraph vector representations from entire documents instead of individual words. This could be well suited for social media posts that can directly be embedded [6]. It is worth noting that this technique can be used to embed the event directly, which can be fed to a classic feed-forward neural network.

3.3 Recurrent Neural network

Recurrent Neural networks have shown to be be very efficient at solving the inference or classification problem for sequential data and is then the neural architecture of choice for this project. As stated previously rumor event streams exhibits some temporal patterns

[2] `https://pypi.org/project/chinese/`

[3] `https://github.com/scikit-learn/scikit-learn/blob/fd237278e/sklearn/feature_extraction`

with respect to their vocabulary and can be learned by a RNN. One has to choose a little more complex architecture than the standard structure to be able to avoid vanishing and/or exploding gradient and to deal with long term dependencies in sequences. The general structure of recurrent neural networks is presented as well as more elaborated structures such as *long-short term memory cells* (LSTM) and *gated recurrent units* (GRU).

Simple recurrent neural networks are computational directed graphs and differ from feed-forward neural networks by the fact that there exists at least one feedback connection between nodes[7]. These networks thus work on data sequences (time series). The RNN updates an internal state and can outputs something at each time step. The basic state and output update equations for a tanh activation are:

$$\begin{cases} h_t = \tanh(Ux_t + Vh_{t-1} + b) \\ o_t = Wh_t + c \end{cases} \tag{4}$$

where b and c are the bias and U, V, W are the weights learned by the network. The basic architecture for fully-recurrent networks works as follows:

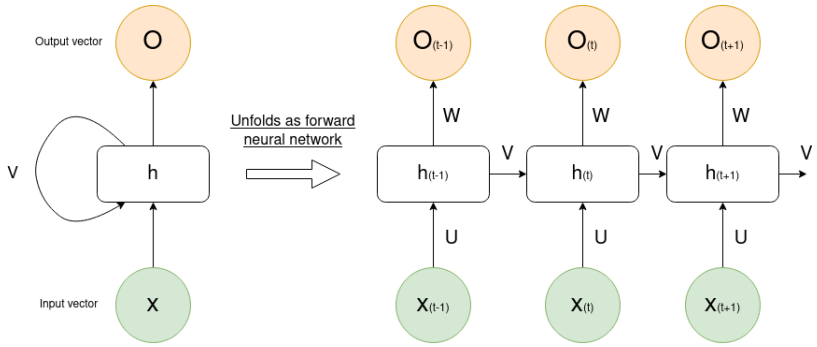

Figure 1: Unfolding basic uni-directional recurrent network

One can interpret the RNN by unfolding it at each time instance, corresponding to a feed-forward neural network trained with back-propagation through time. The two main problems of this architecture is the short-term memory effect in the forward pass, meaning that future states only depend on a small subset of previous states. This is due to the repetition of multiplication by the weight matrix V and tanh activation functions. This problem also shows up in the back-propagation pass showing either vanishing or exploding gradients when computing the loss function [8]. Nowadays a few solutions to that problem have been brought up, we will however focus on the two most important ones.

LSTM cells (Long Short-term Memory): To handle long-term dependencies, a more complex cell where the information flow can be regulated is needed. This includes additional gates controlling the amount of memory added or forgotten and how the input will

affect the memory. The information can now flow on the upper horizontal signal path on fig 2. The LSTM cell has the ability to interact with it trough activation functions and point-wise multiplication. This gate structure allow to regulate the information flow as the sigmoid σ output lies between 1 and 0, regulating the proportion of signal going through.

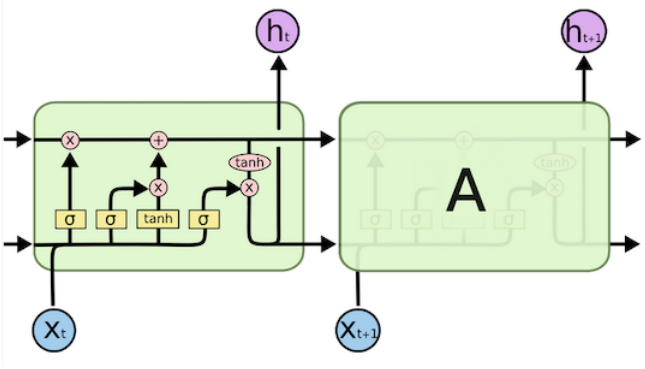

Figure 2: LSTM principle [9]

From left to right, starting from the input x_t from the cell, we have the *input gate layer*, the *forget gate layer*, a *tanh* function creating a new memory state $\widetilde{c_t}$. Which combined with the memory not forgotten, gives the memory state c_t. Then a sigmoid activation gives the output state o_t filtered with a tanh to get the new hidden state h_t. The LSTM internal states equations are [1]:

$$\begin{cases} i_t = \sigma(x_t W_i + h_{t-1} U_i + c_{t-1} V_i) \\ f_t = \sigma(x_t W_f + h_{t_1} U_f + c_{t-1} V_f) \\ \widetilde{c_t} = tanh(x_t W_c + h_{t-1} U_c) \\ c_t = f_t c_{t-1} + i_c \widetilde{c_t} \\ o_t = \sigma(x_t W_o + h_{t-1} U_o + c_t V_o) \\ h_t = o_t \tanh(c_t) \end{cases} \tag{5}$$

Gate units are effective because the trained network is able to choose what to keep in memory and what to forget but induce much more network parameters and thus more complexity.

GRU cells (Gated Recurrent Units): Research has been done to achieve higher performance of LSTMs such as the forget gate added by Gers et al.[10]. A GRU cell is similar to an LSTM cell, but has fewer gates and thus fewer parameters [11] . GRUs also do not have a cell state, but instead operate directly on their hidden state to transfer information. They have two gates, a reset gate and an update gate. The update gate acts similar

to the forget gate of an LSTM. The reset gate is used to decide how to combine the past information with the new input.

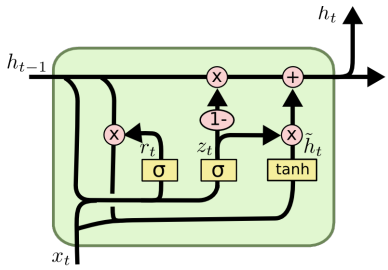

Figure 3: GRU principle [9]

The following equations are given for the gates and hidden state [1]:

$$\begin{cases} z_t = \sigma(x_t U_z + h_{t-1} W_z) \\ r_t = \sigma(x_t U_r + h_{t-1} W_r) \\ \widetilde{h_t} = \tanh(x_t U_h + (h_{t-1}.r_t) W_h) \\ h_t = (1 - z_t).h_{t-1} + z_t.\widetilde{h_t} \end{cases} \tag{6}$$

Where z_t is the update gate, r_t the reset gate and h_t the hidden state with candidate activation function $\widetilde{h_t}$.

3.4 RNN architectures

The different RNN architectures considered in this work come from [1] and are presented in Figure 4.

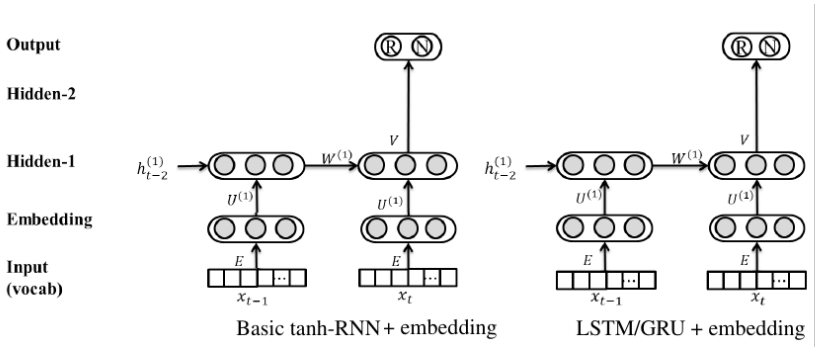

Figure 4: RNN configurations for the implementation [1]

Simple tanh RNN The first one is a simple tanh RNN with an embedding dense layer that has an empirically set fixed size of 100 units used to reduce the dimension of the sparse input TF-IDF scores vector and thus reduce the scale of the RNN parameters. The

output layer use a softmax ativation for the classification problem. Future work can be led to assess the influence of N on the model performance.

Gated units RNN The second architecture replaces the simple RNN layer with an LSTM layer or with a GRU layer. As the time series construction only contains a small amount of intervals (N=10), the two-layer architecture is not implemented. This latter architecture could be used to model more complex time dependencies for large time series by modeling the time dependencies between the outputs of the RNN cells from the previous layer.

3.5 Models training

The Weibo dataset statistics is presented in appendix C. As the Weibo dataset is well balanced, random shuffling of the entire dataset is carried out before the data splitting. These shuffled version of the dataset are saved in order to carry our experiments on the same data. The 4664 events and their corresponding labels are splitted into 70 % for the training set, 15 % for the validation set and 15 % for the test set. This latter set is used to assess the model performance on unseen events.

As it is usually done for classification problems we establish two outputs in the network where the ground-truth labels are described by one-hot encoded vectors, $[1, 0]$ for rumors and $[0, 1]$ for a non-rumor event. The softmax output layer outputs class probabilities such that the *categorical crossentropy* loss function from *Keras* can be used. As there are only two classes, the loss function reduces to:

$$\mathcal{L}(\mathbf{y}, \widetilde{\mathbf{y}}, \boldsymbol{\theta}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=0}^{1} y_j^{(i)} . \log(\widetilde{y}_j^{(i)}) \tag{7}$$

where i is the training example index and j is the index in the 1x2 output class probabilities vector. For each example, the goal is to fit the parameters to the training data such that we minimize this total loss function over the entire dataset. A L-2 norm penalty can be used on the network weights $\boldsymbol{\theta}$ as a regularization technique to prevent overfitting. An other technique to prevent overfitting is to add a dropout rate for the layers weights. To assess the model performance we use the accuracy metric which represents the number of well classified events over the entire dataset.

For the optimizer we chose to use Adam, since it combines the advantages of adaptive learning rate from AdaGrad and the momentum from RMSProp to update the network weights. This optimization algorithm gives faster and safer convergence for our models.

4 Results

The learning rate, vocabulary size K and the regularization parameter were empirically determined to get the best performance and these values are used in the comparison between the different studied RNN architectures. These values are given in table 1.

Vocabulary size (K)	2500
Embedding size	100
Regularization L-2 (λ)	0.01
Learning rate (α)	0.001
Batch size	64
Number of epochs	100

Table 1: Training hyper-parameters values

Without regularization, the RNN overfits on the training data reaching 80 % accuracy while the validation accuracy doesn't improve anymore and start to slightly decrease. A value of λ =0.01 was chosen in order to prevent overfitting. By doing so, the validation follows the training accuracy but doesn't reach 80 % anymore. It appears that the network can't generalize more, probably due the low informational content of the TF-IDF. A more complete text pre-processing with for example word stemming based on their syntax could be carried to get better training accuracies with better generalization. Another way to extract more meaningful information from the data would be to use the *doc2vec* embedding.

The evolution of the training and validation loss over the number of epochs as well as the training loss are presented in fig. 5 for two learning rates and the simple RNN architecture. A learning rate of 0.01 seems to gives oscillations in the validation accuracy while a learning rate of 0.001 gives better convergence. The latter is chosen for the next RNN architectures comparison.

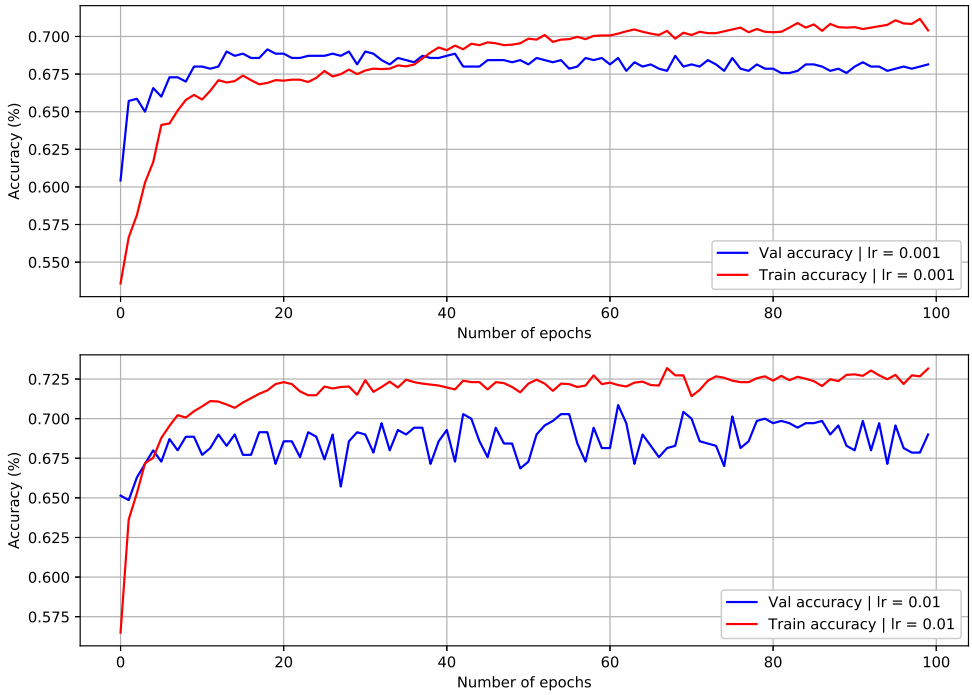

Figure 5: Training and validation accuracies for two learning rate

The three different RNN architectures are then compared with the hyper-parameters values given in table 1 and presented in fig. 6. The mean accuracies are computed over 5 experiments and showing low variability between experiments.

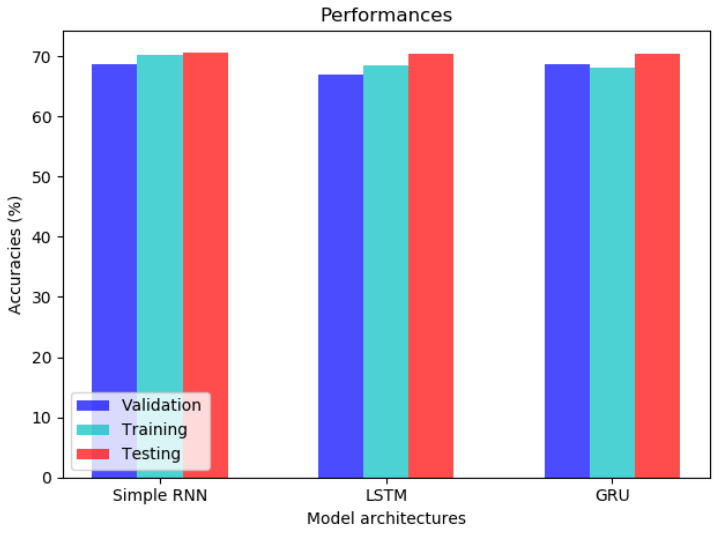

Figure 6: Comparison of the performance of the RNN architectures

We can see that our testing accuracies on unseen events are greater than the training

ones for the tree architectures. The model does not really overfit but it can't learn more. The three architectures gives converge to similar validation and training accuracies. The gated units seems to be not necessary for our short time series.

	Validation accuracy	Train accuracy	Testing accuracy
Simple RNN	68,6 %	70,194 %	**70,632** %
LSTM	67,029 %	68,451 %	70,411 %
GRU	68,571 %	68,177 %	70,386 %

Table 2: Final mean accuracies for different RNN architectures

The final results are summarized in table 2. The simple RNN gives slightly better results than the other ones.

5 Conclusion

We reach the conclusion that the quality of the preprocessing of the data is key in obtaining good results from the different network architectures. With more in-depth textual processing more meaningful information could be extracted. On the other hand deep learned textual features could be an improvement to this task. We are able to obtain good accuracies for the tested architectures, but we are convinced that it is possible to improve on these further. Especially the LSTM and GRU architectures. By considering a larger number of interval capturing more precise temporal pattern, the LSTM and GRU architectures could learn the long-term dependencies that would exist in these longer time series. Moreover more experimental data and tests should be executed to fine tune the various training values. Even though we found some relatively optimal ones, it is quite clear that it can always be improved by empirical work.

References

[1] Jing Ma et al. "Detecting Rumors from Microblogs with Recurrent Neural Networks". In: *AAAI Press* (July 2016).

[2] Natali Ruchansky, Sungyong Seo, and Yan Liu. "CSI: A Hybrid Deep Model for Fake News". In: *CoRR* abs/1703.06959 (2017). arXiv: 1703.06959. URL: http://arxiv.org/abs/1703.06959.

[3] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *CoRR* abs/1301.3781 (2013), pp. 746–751. arXiv: 1301.3781. URL: https://arxiv.org/abs/1301.3781.

[4] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011. URL: http://i.stanford.edu/~ullman/mmds/ch1.pdf.

[5] Yoshua Bengio et al. "A neural probabilistic language model". In: *Journal of machine learning research* 3.Feb (2003), pp. 1137–1155.

[6] Quoc Le and Tomas Mikolov. "Distributed representations of sentences and documents". In: *International conference on machine learning*. 2014, pp. 1188–1196. arXiv: 1405.4053. URL: https://arxiv.org/abs/1405.4053.

[7] K.-L Du and M.N.s Swamy. "Recurrent Neural Networks". In: Dec. 2014, pp. 337–353. ISBN: 978-1-4471-5570-6. DOI: 10.1007/978-1-4471-5571-3_11.

[8] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[9] Christopher Olah. *Understanding LSTM Networks*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[10] F.A. Gers, J. Schmidhuber, and F. Cummins. "Learning to forget: continual prediction with LSTM". In: *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*. Vol. 2. ISSN: 0537-9989. Sept. 1999, 850–855 vol.2. DOI: 10.1049/cp:19991218.

[11] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *arXiv:1406.1078 [cs, stat]* (Sept. 2014). arXiv: 1406.1078 version: 3. URL: http://arxiv.org/abs/1406.1078 (visited on 05/29/2020).

[12] Yoav Goldberg and Omer Levy. "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method". In: *arXiv preprint* (2014). arXiv: 1402.3722. URL: https://arxiv.org/abs/1402.3722.

[13] Andrej Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks*. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[14] Jason Brownlee. *How to Clean Text for Machine Learning with Python*. URL: https://machinelearningmastery.com/clean-text-machine-learning-python/.

[15] Deepak Mishra. *DOC2VEC gensim tutorial*. URL: https://medium.com/@mishra.thedeepak/doc2vec-simple-implementation-example-df2afbbfbad5.

[16] Rageeni Sah. *Text Data Cleaning - tweets analysis*. URL: https://www.kaggle.com/ragnisah/text-data-cleaning-tweets-analysis.

[17] Mohammed Al-Sarem et al. "Deep Learning Based Rumor Detection on Microblogging Platforms: A Systematic Review". In: *IEEE Access* PP (Oct. 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2947855.

Appendices

A Group members contributions

- Logan: Dataset extraction and processing, time series construction and textual features extraction.
- Jolan: RNN data processing and RNN architectures implementation and training
- Théo: RNN architectures implementation, simulations and results gathering.

B Variable-length time series construction

Input : Relevant posts of $E_i = \{(m_{i,j}, t_{i,j})\}_{j=1}^{n_i}$,
Reference length of RNN N
Output: Time intervals $I = \{I_1, I_2, \ldots\}$

/* Initialization */
$L(i) = t_{i,n_i} - t_{i,1}$; $\ell = \frac{L(i)}{N}$; $k = 0$;
while *true* **do**
  k ++;
  $U_k \leftarrow Equipartition(L(i), \ell)$;
  $U_0 \leftarrow \{\text{empty intervals}\} \subseteq U_k$;
  $U'_k \leftarrow U_k - U_0$;
  Find $\bar{U}_k \subseteq U'_k$ such that $\bar{U}_k$ contains continuous intervals that cover the longest time span;
  if $|\bar{U}_k| < N$ && $|\bar{U}_k| > |\bar{U}_{k-1}|$ **then**
    /* Shorten the intervals */
    $\ell = 0.5 \cdot \ell$;
  else
    /* Generate output */
    $I = \{I_o \in \bar{U}_k | I_1, \ldots, I_{|\bar{U}_k|}\}$;
    return I;
  end
end
return I;

Figure 7: Variable-length time series construction given the set of relevant posts of an event E_i and the reference length of RNN N

C Weibo dataset statistics

Statistics of the dataset

Statistic	Weibo
Users #	2,746,818
Posts #	3,805,656
Events #	4,664
Rumors #	2,313
Non-Rumors #	2,351
Avg. time length / event	2,460.7 Hours
Avg. # of posts / event	816
Max # of posts / event	59,318
Min # of posts / event	10

Figure 8: Weibo dataset statistics