Local Classification with Recurrent Neural Network for Profiling Hate Speech Spreaders on Twitter. Notebook for PAN at CLEF 2021

Pablo Pallarés [1] and Carlos Herrero [1]

[1] *Universitat Politècnica de València, Camino de Vera, s/n, Valencia, 46022, Spain*

Abstract

The general purpose of this work is the study, implementation, and development of a system that, given a Twitter feed, determines if its author spreads Hate Speech (HS). The corpus used was provided in the 9th International Author Profiling Contest. This data set consists of 200 Twitter profiles: 100 in Spanish and 100 in English. There is one XML file per author with 200 tweets. The name of the XML file corresponds to the unique identification of the author. Finally, there is a truth.txt file with the list of authors (author id) and the ground truth (Class to which the author [0 or 1] belongs). First, we will partition the data set into Train, Development and Test, to carry out a supervised training of a classifier (x = sample, y = label) with the training set. Second, we proceed to make a parameter adjustment using the development set to minimize the classification error. This work intends to approach this task from a Deep Learning (DL) perspective, introducing some peculiarity that allows us to obtain good classification results. The developed system must input the absolute path to a decompressed data set and generate an XML file for each document in the data set, as described in the task. In the last section we will evaluate the accuracy obtained.

Keywords

Author profiling, hate speech, twitter, tweet preprocessing, profiling haters.

1. Introduction

Hate Speech (HS) is commonly defined as any communication that discredits a person or group based on some characteristic such as race, colour, ethnicity, gender, sexual orientation, national origin, religion or other characteristics. Given the enormous amount of content generated by users on Twitter, the problem of detecting and possibly contrasting HS spread becomes fundamental, for example, to fight against misogyny and xenophobia. To this end, this task aims to identify potential propagators of hate speech on Twitter as a first step in preventing hate speech from spreading among online users. It is about investigating whether it is possible to discriminate authors who have shared hate speech in the past from those who have never done so, as far as we know.

[1]CLEF 2021 – Conference and Labs of the Evaluation Forum, September 21–24, 2021, Bucharest, Romania
EMAIL: pabpalfo@upv.es (A. 1); carherb2@upv.es (A. 2)
ORCID: 0000-0002-0120-3251 (A. 1); 0000-0003-1985-3452 (A. 2)

This report includes all the development of the HS system, the results obtained with the classifier used, and the strategy followed from a Deep Learning approach. As we mentioned, we partition the data set into Train, Development and Test to evaluate the model. Finally, if we achieve results in test similar to those of development, we will carry out the training with the entire data set. In addition, we used a series of resources obtained from the network (Dictionary of Emojis, Word Embbedings [Glove_twitter_es / en]).

From here, the report is structured in the following sections: in the second section, we will expose the theoretical foundations of the proposed solution to tackle this task. The third section details the process of obtaining and pre-processing the data from the files provided. The fourth describes the processing and experimentation based on deep machine learning and an analysis of the results obtained to adjust parameters. We briefly review tokenization, data preparation, and generation of a model based on Recurrent Neural Networks. This section uses a Long Short-Term Memory (LSTM) [6] as a Decoder. Finally, the best model obtained to generate the results file with the test set is determined, and we conclude the study carried out in this task.

We made the different implementations of this work using Python language. On the one hand, we used the open-source library TensorFlow to train deep machine learning models and the high-level framework for deep learning, Keras. On the other hand, we also used the open-source library for machine learning sklearn.

2. Theoretical fundaments

As previously stated, we would try to introduce some innovative element to approach this task. The first idea is to treat each tweet as a local feature of the author, and instead of coding the author's 200 tweets as a single sample, we aim to obtain a tweet-level classifier. In the training phase of this local classifier, we labelled each tweet with the associated tag of the author:

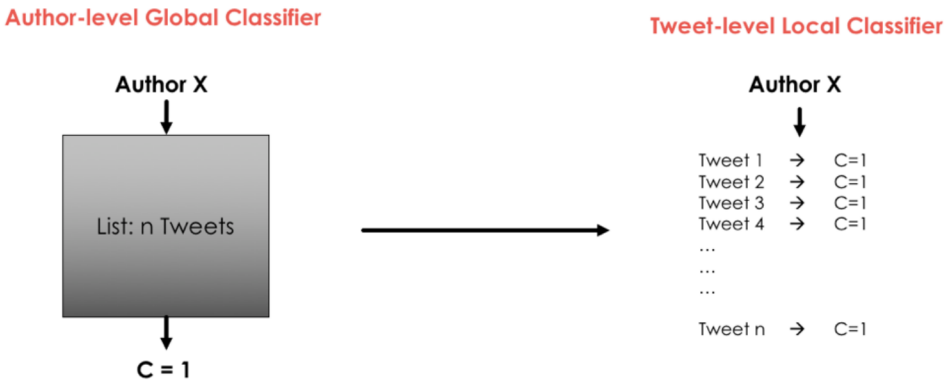

Figure1: Transformation of the data for training the local classifier

This can be a problem because all the author's tweets labelled as class "1" (which means this author disseminates hate speech) are not disseminating hate speech. However, we start from the premise that every author labelled as class "1" has a list of tweets where a certain percentage of them disseminate hate speech. Based on this approach, in the inference phase, we established a voting system, where each Tweet (local characteristic) in the list of tweets of an author (global sample) votes for a class. Finally, we tagged the author's list of tweets and the author's profile with the most voted class label. See the theoretical reasoning:

With this reasoning we assume that we have local class decisions:

$$\hat{c} = \underset{1 \le ci \le C}{argmax}\, p(C \mid tweet_i) \qquad C\{0,1\}$$

A summary of the reasoning of the probabilistic model followed is the following: we can estimate the posterior probability of class C given the entire set of tweets of an author. We can formulate it as follows, making the appropriate marginalizations of the class tags associated with each Tweet:

$$p(C|tweets_{author}) = \sum_{C_1=1}^{C} \sum_{C_2=1}^{C} \dots \sum_{C_n=1}^{C} p(C,\ c_1, c_2, \dots\ , c_n|tweets_{author}) \qquad (1)$$

Regarding the internal probability of the marginalization in expression (1), we can decompose it into the product of 2 terms: the probability that given the list of an author's tweets gets the classification of each tweet, and the posterior of class C given the class of each tweet and the list of tweets of the author to determine its label:

$$p(C, c_1, c_2, \dots\ , c_n|Ts_{autor}) =$$

$$= p(c_1, c_2, \dots\ , c_n|Ts_{autor}) * p(C|Ts_{autor}, c_1, c_2, \dots\ , c_{n)} \qquad (2)$$

Here, we made a series of approximations to simplify the calculations, approximating the two terms obtained in expression (2):

- **<u>Approaches:</u>**

1. For the first term of expression (2), we can assume that the author's Tweets ranking is statistically independent of the ranking of the other author's Tweets. On the one hand, this assumption is consistent because not all of an author's Tweets have to spread hate speech or vice versa, but, on the other hand, we can consider that there is some relationship between an author's different Tweets. Once this consideration has been made at the conceptual level, this facilitates operability, we also made the Naive Bayes assumption incorporating the statistical independence between Tweets:

$$p(c_1, c_2, \dots\ , c_n|Ts_{author}) \approx \prod_{i=1}^{n(n^{o}\, Tweets_{author})} p(ci|Tweet_i) \qquad (3)$$

2. In the second term of expression (2), we can make two approximations. First, we assume the statistical independence of class C concerning the author's list of Tweets, given the class labels of each of the tweets in the list. Conceptually, if one has the classification of each tweet (local classification), the author's list of tweets is not required to determine the posterior probability of the C class:

$$p(C|Ts_{author}, c_1, c_2, \dots\ , c_{n)} \approx p(C|c_1, c_2, \dots\ , c_n) \qquad (4)$$

In the second place, we can approximate the probability that the author belongs to class C by counting many times we classified a Tweet as class C, divided by the total number of Tweets:

$$p(C|c_1, c_2, \dots\ , c_n) = \frac{1}{n\ (n^{o}\, Tweets_{author})} * \sum_{i=0}^{n} \delta(\ c_i, C) \qquad (5)$$

Finally, substituting expressions (3) and (5) with the approximations, in the original expression (1), we have:

$$p(C|tweets_{author}) \approx \sum_{C_1=1}^{C}\sum_{C_2=1}^{C}\dots\sum_{C_n=1}^{C}\left(\prod_{i'=1}^{n} p(ci'|Tweet_{i'})\right) * \frac{1}{n}\sum_{i=0}^{n}\delta(c_i, C) =$$

$$= \frac{1}{n}\sum_{i=0}^{n}\sum_{C_i=1}^{C} p(c_i|T_i)\ \delta(c_i, C)\boxed{\sum_{C_1=1}^{C}\dots\sum_{C_n=1}^{C}\prod_{i'=1, i'\neq i}^{n} p\left(ci'|T^{[i']}\right)} = \frac{1}{n}\sum_{i=1}^{n} p(c|T_i) \quad (6)$$

$$=1$$

Therefore, we approximate the posterior probability of class C given the list of tweets of an author, as the average of the posterior probabilities of the class C, given each of the tweets in the list. Moreover, we estimated the a posteriori probability of class C given a Tweet through the best model obtained in the experimentation phase: $P(C|Tweet\_author_i)$

$$p(C|tweets_{author}) \approx \frac{1}{n}\sum_{i=1}^{n} p(c|T_i) \quad (7)$$

Finally, we select the most voted class $\hat{c}$ at the tweet level from the same author. The voting system consists of inferring with the local classifier trained on each of the Tweets to obtain a list with the class labels of each Tweet. Finally, we classify the author with the most voted class of the 200 Tweets that the author has. This justifies that this type of technique is known as a "voting scheme". Therefore, once we have the list with all the class labels associated with each of the author's tweets (list of 200 labels in this particular task), we select $\hat{c}$:

$$\hat{c} = \underset{1\leq ci\leq C}{\operatorname{argmax}}\, p(C \mid tweets_{author}) = \underset{1\leq ci\leq C}{\operatorname{argmax}}\, \frac{1}{n}\sum_{i=0}^{n}\delta(c_i, C) \quad (8)$$

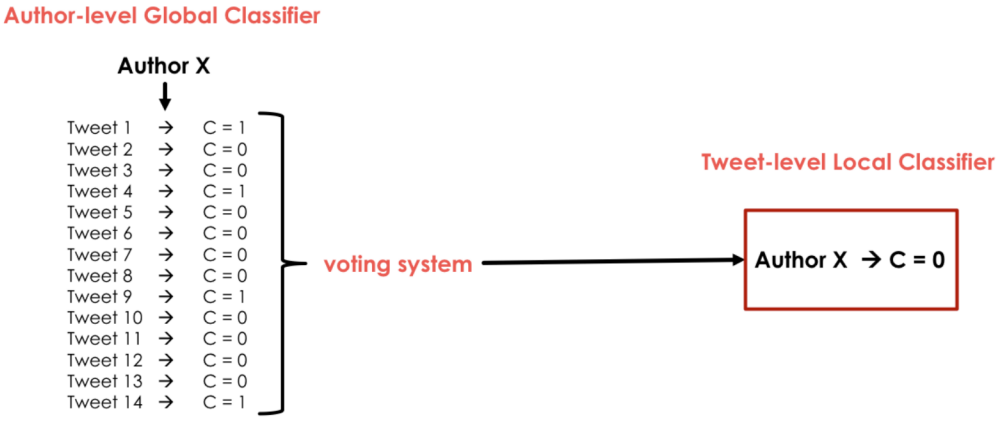

Figure 2: Example voting system in the inference phase

This approach ignores the posterior probabilities of the classes for each tweet and only considers the class label. We could set up a more complex system in several ways:

- accumulating the posterior probabilities of each tweet and obtaining the average
- or establishing a second classifier with the class labels obtained and the samples to have a different weight and importance.

However, from experience in other tasks, we decided to simplify as much as possible.

3. Data processing

Once we have theoretically contextualized the procedure to follow, the first step is to tackle the formatting problem to convert the files (.xml) to plain text and select the relevant information to the task. Each author file has different fields such as the language ("es" or "en"), but for this training and development phase, we are only interested in the 200 tweets of content and the label contained in the truth.txt file, which as we recall contains a list of authors and the associated class label (0 or 1).

With this first proposal, we went from having 100 samples of Spanish and 100 samples of English to having 10,000 samples per language, which already gives us more guarantees to tackle the task with Deep Learning.

The first step is to design a Python function, **open_data()**, which receives the absolute path to an uncompressed dataset with the XML files and the Spanish and English truth.txt files and generates a Python dictionary with the following structure:

{“es”: “[{“id”: “id”, “value”: “class_label”, “data”: “tweets”} … {}]”,
“en”: “[{“id”: “id”, “value”: “class_label”, “data”: “tweets”} … {}]”}

Figure 3: function open_data()

The next step is to pre-process the tweets. To do this, on the one hand, we use an external resource. An Emoji dictionary, found on the web, which we converted it into a Python dictionary with the following format → {"Key = Emoji": "Value = word/set_of_words", ...}. See an example of performing an Emoji à Word/Word_Set substitution process:

-Example Dictionary Emojis:

I won 🥇 in 🏏 → I won 1st_place_medal in cricket

In addition, we generated a tokenizer function to:

- separate punctuation symbols
- handle all mentions to users of type @user as a single token: <user>.
- handle all hashtags of type #hashtag as a single token: <hashtag>
- handle all urls in different formats as a single token: <url>
- keep all emails as a single token.
- keep English words ending with “'s” as a single token.
- keep all dates in different formats as a single token.
- keep all times in different formats as a single token.
- keep all whole numbers and decimals as a single token.
- keep all acronyms together as one token.

Using these two tools: tokenizer and Emoji dictionary, plus the **lower()** method to convert everything to lowercase, we also implement a Python function, **define_dataset()**. This function receives the list of data of a language (internal lists to the data dictionary: data["es"], and

data["en"]), and will return two lists: a list of Strings with the tokenized and cleaned tweets with the Emojis replaced by their associated word, a list with the class label of each tweet, and another list with the author of each tweet. For each of the tweets from the same author, we repeated the same class label and the same author 200 times.

Figure 4: function define_dataset()

Of the 100 English authors (10,000 tweets) and the 100 Spanish authors (10,000 tweets), we reserve 10 English authors 10% (1,000 tweets) and 10 Spanish authors 10% (1,000 tweets) for the Test. To partition the data set, we must do it at the author level, leaving all the tweets of the selected authors in the test set. In this way, we inferred all their tweets, and through the established voting system (the most voted class), determined the author's class label. If we had trained on any of the author's tweets, the evaluation would be unrealistic.

In this way, we again extended the training data, in this case, by twice as much. Therefore, we unified the lists of tweets, labels, and authors separated by language into a single list and shuffled the entire corpus.

4. Experimentation Based on Deep Learning

The next task is to perform the Text2Sequences conversion using some tools from the keras processing package:

1. We applied the Keras tokenizer on the list of tweets.

2. We performed the Text2Sequences conversion by applying padding afterwards to generate 82 values (length of the most extended tweet). In these lists, we replaced the words by their index in the vocabulary generated with the tokenizer. In this way, the list of tweets becomes a list of sequences, and each of these sublists is the input parameter of our neural network.

3. We loaded the embeddings obtained from the network: "glove.twitter.27B.100d.txt". These are 100-dimensional embeddings pre-trained with a Twitter task in English and Spanish [7].

4. With these Embeddings and the tokenizer, we generated an embeddings matrix, with the words of our vocabulary, which we loaded later as weights of our model in the embeddings layer.

Next, we need to adapt the class labels to work with our neural model. Recall that our set of labels was C = {0, 1}. Considering that our model had as output layer a dense layer of 2 neurons and a softmax as an activation function, the interesting thing is to make a new conversion with keras.utils.to_categorical. In this way, we represented our set of labels as follows:

- 0 = [1 0]
- 1 = [0 1]

Model based on Recurrent Neural Networks (1 input). This model has as a single input the sequence (coded tweet). This sequence enters element by element in an LSTM [6] to generate the activation with a ReLU as an activation function connected to a dense layer with four neurons and a softmax as activation.

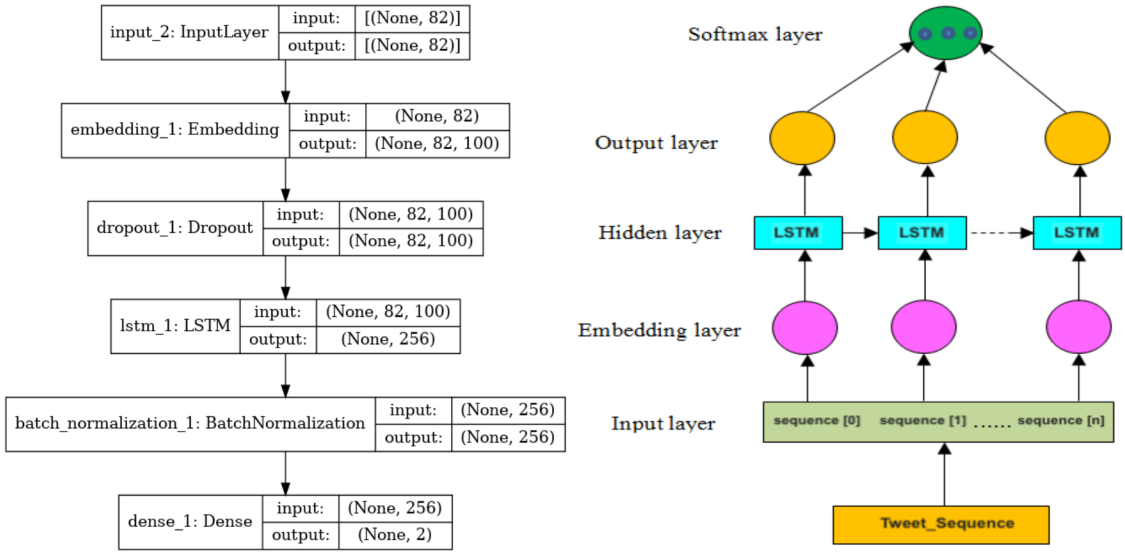

Figure 5: RNN based Model

We earlier experimented with initial network topology and modified using some techniques to avoid, as far as possible, overfitting using regularizers.l1_l2 (regularization L1, L2), BN (Batch Norm), and dropout. Let us take a closer look at them:

- **Regularization L1, L2:** Regularization consists of applying a penalty to the optimization criterion used to avoid extreme values of the network's weights to obtain high probabilities with moderate values of the weights (overfitting correction). L1 and L2 are the most common, and we can obtain them by calculating the norm in r of the vector of weights. It consists of raising to a term r (in this case 1 and 2) the absolute value of each of the weights, and apply the square root of r of the sum of all these terms. In this way, L1 (r=1) will be the sum of the absolute values of the weights, and L2 (r=2) the square root of the sum of the square of the weights. These terms (L1, L2) are multiplied by a scaling factor with values in the order of 10-2 to 10-5. We apply the regularization on the dense layers:

  model.add(Dense(1024, kernel_regularizer= regularizers.l1_l2(l1=1e-5, l2=1e-4))) [8]

- **Batch Norm** [9]**:** It is one of the solutions to solve the different problems presented by networks. It manages to improve the results by increasing the convergence capacity and providing numerical stability. It is not easily classifiable; it is halfway between optimization and generalization. Example in keras:

  model.add(BatchNormalization()) [10]

  Each neuron follows a normal distribution mean 0 standard deviation 1 for the input Batch population. It does not have a single activation, and if the Batch is 128, 128 values

follow this distribution. In addition, it can be scaled and shifted. It does not have to be centered at the origin.

$$y = \boldsymbol{\gamma}\ x + \beta$$

As we have not modified any parameter, by default, it initializes the displacement and scaling to 0 and 1 and learns them with Back-Propagation. For this reason, if placing a dense layer before the batch normalization, it would be ideal not to put Bias because batch normalization already introduces Bias and, in this way, we save calculations.

- **Dropout** [11]**:** is another possible solution to achieve better generalization. It eliminates neurons randomly, forcing the neurons that have not been eliminated to have the same representation capacity as all of them would have. Similar to Sparsity but less aggressive, in this case, the deactivated neurons are around 50%.

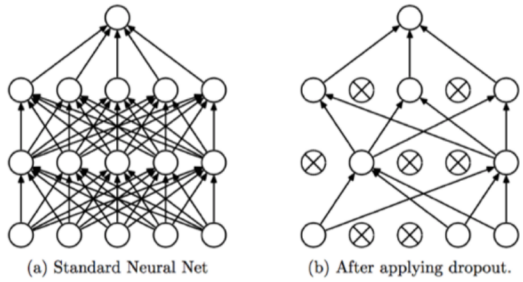

Figure 6: Dropout *[11]*

We can train it using different combinations, but since there is an infinite number of combinations, once we trained the network, in the inference process, the original network is used (all active) with the corresponding weights learned by turning off neurons. We multiply all the weights by [1-p] [probability of having survived] in the activation phase.

model.add(Dropout(0.5)) [12]

Once we defined the model, we proceed to the experimentation and development phase. During the development, we carried out a total of 3 experiments to test the influence of the different parameters studied. The first experiment consisted of testing with different sizes of the LSTM: 32, 64, 128, 256 and 512. In the second experiment, we experimented with different optimization algorithms like Adam, Adadelta and Adagrad. Finally, we loaded the Embeddings from an external file and performed a Fine-tuning of its weights' matrix. With this, we intended to adapt the Embeddings of the words to the specific task. Notice that we calculated the average results obtained from 3 different executions to obtain each of the values. For each of the runs, we shuffled and randomly partitioned the data set reserved for training and Development, always reserving 15% of this set (2700 tweets out of 18,000) for Development. In this way, the tests are more realistic, as we extracted the development set randomly for each training. For this purpose, we used the resource train_test_split from the sklearn.model_selection package. [13].

4.1. Experiment 1: RNN (LSTM) size

The first proposed exercise consisted of modifying the number of neurons present in the LSTMs [2] of the Encoder and Decoder. We tested its effect with 32, 64, 128, and 256 neurons. The results obtained on the validation set are as follows:

Table 1:
Size Comparison LSTM

WE	RNN size	Opt. Algorithm	Val_Accuracy
frozen	32	Adam	0.6439
frozen	64	Adam	0.6492
frozen	128	Adam	0.6489
frozen	256	Adam	0.6495
frozen	512	Adam	0.6487

In the table, we can see that using the Adam optimizer, varying the size of the LSTM with froze pre-loaded Embeddings. We obtained the best results with an LSTM size of 256 and 64. Interestingly, we obtained better results in those than with 128 neurons which is the intermediate value. This is probably because the number of experiments should be higher. For that reason, we decided to use the LSTM with 256 neurons in the following experiments so the model has a greater representation capacity.

4.2. Experiment 2: Optimization Algorithms

The second experiment consists of modifying the optimization algorithm. Initially we have worked with *Adam* [14], to later test with *Adagrad* [14] and *Adadelta* [14]. In the following paragraphs we will see a description of the 3 optimization algorithms.

All 3 are gradient descent optimization algorithms and adjust the learning rate as a function of the gradient so that if the gradients are substantial, the learning factor decreases and in areas of low gradient, valleys where the gradient is low, they adjust the gradient so that the weights are more significant to transmit the improvement without dissipating.

In Adagrad (Adaptative Gradient), the variation of the weights depends on the gradient, but unlike SGD, it is not a scalar product of the learning factor and the gradient but an element-wise product. It is a helpful algorithm for the treatment of sparse data because, for the adaptation of the learning rate discussed above, it is divided by $\sqrt{(m_i(l) + e)}$. This component accumulates the gradients squared to eliminate the symbol so that if they are huge (parameters with more frequent features), the learning factor decreases, and if it is minimal (parameters with infrequently associated features), the learning factor will be more prominent.

Adadelta, is an extension of the previous one, but storing a fixed-size window of accumulated gradients (previous) that allows having a view of how it evolves (the increments of the weights of a specific range) so that if the increments are reasonable the learning factor increases (it has a certain resemblance to momentum). This avoids that the reduction of the learning rate is monotonous.

Lastly, and despite being used in the first exercise, Adam (Adaptative Moment Estimation) is the most complex of the three methods. It combines all of the above and adds the previous weight to the gradient so that, if the previous weight were relevant, it would have more influence on the new one. See the results obtained on the validation in Table 2.

Table 2:
Comparison of Optimization Algorithms

Learning Rate	WE	RNN size	Opt. Algorithm	Val_Accuracy
0.002	frozen	64	Adam	0.659
0.1	frozen	64	Adagrad	0.564
1	frozen	64	Adadelta	0.502

First of all, note that this experiment has taken somewhat longer because it has been necessary to adjust each algorithm's most appropriate learning rate. After adjusting the parameters for each model, and as shown in Table 2, the best results have been obtained with Adam, and a learning rate of 0.002 has reached an Accuracy = 67.9%. Adam usually obtains better results, but it is necessary to carry out tests to certify it.

4.3. Experiment 3: Fine Tuning

The initial idea of this experiment was to perform a re-training starting from the weights of the best model obtained in the experimental phase. This re-training consists of performing a fine-tuning by unfreezing the weights of the embeddings' matrix and reducing the learning rate of the optimizer. In this way, we intend to adapt the weights of the embeddings of the words, which we have loaded and pre-trained from a Twitter task in English and Spanish, to the shared task, without altering the network's parameters. However, the improvements obtained were much lower than those achieved in a second experiment. For this one, again, the embedding layer is unfrozen from the beginning of the training. We saw an improvement of 2 points with the parameters. A learning rate scheduler has been introduced for the two experiments, applying annealing of this learning rate, with some reheating.

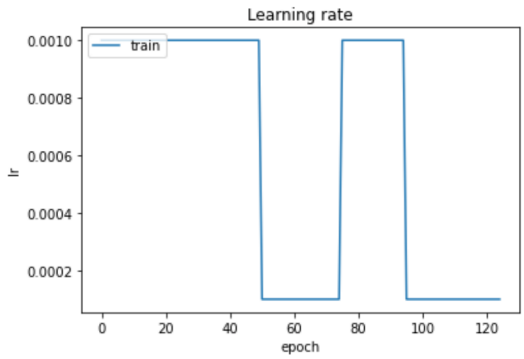

Figure 7: Learning rate Annealing with reheating

Table 3:
Comparison with unfrozen WE weights

WE	RNN size	Opt. Algorithm	Val_Accuracy
unfrozen	32	Adam	0.667
unfrozen	64	Adam	0.6732
unfrozen	128	Adam	0.6727
unfrozen	256	Adam	0.6725
unfrozen	512	Adam	0.67249

From Table 3, we can see that the results have improved considerably by training from scratch with the weights of the unfrozen embeddings. Furthermore, we found that, in this case, the results

with 64 neurons in the LSTM are slightly better than the results with 128 and 256 neurons. Unfortunately, we performed this experiment with the 256-neuron model before delivering the competition results due to lack of time and because it was the best model so far at this time. Therefore, we obtained the results used in the competition with a model trained from scratch with a 256-neuron LSTM and using the Adam optimizer with the learning rate scheduler described above. The truth is that the results are alike with the different LSTM sizes, but the main difference lies in the training and inference time. The model with 64 neurons in the LSTM is approximately four times faster than the model with 256 neurons.

5. Results Analysis

The comparative results obtained throughout the experimentation phase are at the tweet level (local), training with the train set, and validating with the development set. However, the competition proposes the presentation of ranking results at the author level (global). For this purpose, we established a voting system, as described in Section 2 of this report. Essentially, it assigns the author the most voted class label during the inference phase of his set of tweets, with the neural classifier obtained at tweet level.

In this section, two results are going to be analyzed. On the one hand, those obtained in the shared task, which is previous to this last comparison of experiment number 4 and, therefore, were obtained with a recurrent neural model with 256 neurons in the output layer of the LSTM and using Adam as optimizer. We trained this model with the whole dataset provided in the task, shuffling all the authors' tweets in Spanish and English and applying some training techniques, such as those already mentioned in Section 4 and others, such as others learning rate annealing with reheating.

Table 4:
Competition Results

RNN size	Epochs	Aut. ES accuracy	Aut. EN accuracy	Average Accuracy
256	125	0.78	0.58	0.68
256	250	0.80	0.57	0.685

We presented the results obtained with two different training of the same model. The results are indeed very similar. In both cases, the accuracy in Spanish is well above the accuracy in English, despite having a well-balanced task at the language level 10,000 Tweets in Spanish and 10,000 tweets in English. The local classifier developed was bilingual, but the results tend to be better in one language than the other for similar competitions. The result of the average of both languages is very similar to that obtained in the tests before the presentation of the model.

Secondly, we analyze the results obtained after this second part of experiment 4, where improvements have been achieved at tweet level, unfreezing the embeddings matrix, but with an LSTM of 64 neurons. At the same time, the test and inference phase are considerably accelerated. Notice that, in this case, the models have been trained, with 90% of the data provided (90% of authors), belonging to the train and development set, and have been evaluated with the Tweets of 10% of the authors reserved for testing. Therefore, we compared both models: LSTM-256 (presented in the competition) and LSTM-64 (best model obtained at Tweet level) under the same circumstances:

Table 5:
Post-competition comparison

Model	WE	RNN size	Opt. Algorithm	Test Accuracy
1	descongelados	64	Adam	0.64
1	descongelados	256	Adam	0.62

Table 5 shows the results obtained in the test evaluated with 200 tweets from 10 authors in Spanish and 200 tweets from 10 authors in English (4,000 tweets in total). We can see that the results have improved with an LSTM of 64 neurons achieving an accuracy improvement of almost 2 points and speed improvements of x4. The 256 LSTM model does not have the same weights as the model used to obtain the competition results because we trained the competition model with 100% of the data provided and the one used in this test with 90%. However, we used the same parameters and training strategies. In both models, we obtain an accuracy near 68.5% with the Tweet level validation set.

6. Conclusions

During the development of this work, we addressed the tasks of reformatting, filtering and preprocessing important text, data preparation, vectorization, tokenization, model training and fine-tuning. In addition, we used deep learning at the tweet level (Local approach). Once we have done all the experimentation, we determined that the best classifier uses recurrent neural networks with an LSTM of 256 neurons, Adam optimizer and fine-tuning by unfreezing the WE matrix. With this model, we inferred the test set at the tweet level and set up the voting system to define the output set in the format given in the Author Profiling task [5]. However, we found that we can still propose different strategies to improve the speed and the accuracy of the classifier, e.g., reducing the LSTM to 64 Neurons. Some of the proposals we can think of for future work are analyzing the Tweets' size in the task using a histogram and adjusting the string length input parameter in the model. This will speed up the training, which is because LSTMs cannot parallelize and take advantage of GPU capacity. In addition, the longer the string length, the heavier it is. Another proposal would be introducing more state-of-the-art neural models, such as Transformers: Bert, with attention models capable of selecting the most relevant information. In short, we have achieved good results, taking into account state of art in similar tasks.

7. References

[1] J. Bevendorff, B. Chulvi, G. L. d. l. Peña, M. Kestemont, E. Manjavacas, I. Markov, M. Mayerl, M. Potthast, F. Rangel, P. Rosso, E. Stamatatos, B. Stein, M. Wiegmann and M. Wolska. "Overview of PAN 2021: Authorship Verification, Profiling Hate Speech Spreaders on Twitter, and Style Change Detection" in *12th International Conference of the CLEF Association (CLEF 2021)*, Bucharest, Romania, 2021.

[2] M. Potthast, T. Gollub, M. Wiegmann and B. Stein. "Information Retrieval Evaluation in a Changing World"; TIRA Integrated Research Architecture; series: The Information Retrieval Series; ids. stein:2019r; isbn. 978-3-030-22948-1, N. F. a. C. Peters, Ed., Berlin Heidelberg New York: Springer, sep. 2019. doi. 10.1007/978-3-030-22948-1_5

[3] F. Rangel, G. L. d. l. Peña, B. Chulvi, E. Fersini and P. Rosso. "Profiling Hate Speech Spreaders on Twitter Task at PAN 2021"; CLEF 2021 Labs and Workshops, Notebook Papers. In *Conference and Labs of the Evaluation Forum (CLEF 2021)*, 2021.

[4] F. Rangel, G. L. d. l. Peña, B. Chulvi, E. Fersini and P. Rosso. "Profiling Hate Speech Spreaders on Twitter"; In 9th International Competition on Author Profiling; Symanto, 2021. Url: https://pan.webis.de/clef21/pan21-web/author-profiling.html.

[5] "Profiling Hate Speech Spreaders on Twitter" Symanto, Mayo 2021. url: https://pan.webis.de/clef21/pan21-web/author-profiling.html.

[6] e. Hochreiter and J. Schmidhuber. "Long Short Term Memory" in *technical report FKI-207-95,* pp. 0-8, 21 Agosto 1995.

[7] R. S. C. D. M. Jeffrey Pennington. "GloVe: Global Vectors for Word Representation". Url: https://nlp.stanford.edu/projects/glove/.

[8] K. Api, "keras layers regularizers". Url: https://keras.io/api/layers/regularizers/.

[9] C. S. Sergey Ioffe, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *32nd International Conference on Machine Learning*, LILLE GRAND PALAIS, 2015.

[10] K. api, "Keras batch_normalization". Url: https://keras.io/api/layers/normalization_layers/batch_normalization/.

[11] G. H. A. K. I. S. R. S. Nitish Srivastava. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research 15 (2014) 1929-1958,* Mayo 2014.

[12] K. api, "keras dropout".

[13] "sklearn model_selection,". Url: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[14] S. Ruder. "An overview of gradient descent optimization algorithms", *insight Centre for Data Analytics, Nui Galway Aylien Ltd.,* 2016.