

Algorithms for massive datasets project

Giuseppe Carugno, Giuseppe Lonardoni

July 4, 2022

Contents

0 Disclaimer	1
1 Introduction	2
2 Preprocessing	2
3 Embedding neural network	3
3.1 Input layer	3
3.2 Text vectorization layer	3
3.3 Embedding layer	4
3.4 Output layer	4
4 Labels decomposition	5
5 Forecasting neural network	6
5.1 Input layer	6
5.2 Hidden layers	6
5.3 Output layer	6
6 Results	7
6.1 Neural networks models	8
6.1.1 Embedding neural network	8
6.1.2 Forecasting neural network	9
6.2 Performance	10
6.2.1 Training and validation	10
6.2.2 Testing	10

0 Disclaimer

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

1 Introduction

The aim of the project is to develop an hashtags predictor using neural networks and deep learning. The original dataset repository is available on Kaggle and refers to the "Ukraine Conflict Twitter" dataset. It can be downloaded directly through the Kaggle API. The version used is the update of the 23rd of May. The main tools used are Google Colab and TensorFlow.

2 Preprocessing

The zipped dataset, once downloaded, has been decompressed, its CSVs rows filtered, using the language column to retain only the English ones, and then appended together using PySpark to create a dataframe. Only the column containing the text of the tweets is kept.

Each text is lowercased and split into tokens, using whitespace as separator. Tokens containing words are collected, with stop words and punctuation removed, to avoid storing useless informations, and concatenated together to form a new text string. These strings are the values of the first column of the new dataset. From all the tokens containing hashtags a list of the most frequent tokens is collected. These are the hashtags to forecast.

Once we have a list of frequent hashtags to predict, we scan the tokens of the hashtags retrieved from each text to check whether there are some that are also present in the list of frequent hashtags. If there's none all tokens are discarded and an empty value is left, otherwise a new string is composed with the integers representing the indexes of those hashtags in the frequent hashtags list, with a whitespace as a separator between each integer. The results of this operations are the values of the second column of the new dataset. Storing for each row just the indexes of the present frequent hashtags reduces the size necessary to store the new dataset.

After preprocessing sizes of the compressed and uncompressed data of the old dataset are reduced from 10 GB and 26 GB to the 640 MB and 2.7 GB respectively for the new one. While the old dataset is dropped the new one is stored and, since it contains 27 millions of rows, its size is feasible for deep learning.

3 Embedding neural network

The first relevant design choice is to build an embedding neural network to apply to the first column of the dataset. Since it contains categorical values the embedding neural network in charge of applying word embedding to it, to translate the text of the tweets into numbers to be fed as input data to the forecasting neural network.

The decision to split word embedding and forecasting reduces the complexity of both neural network models, allows to develop, train, test and fine tune both more easily and accurately and also, since we are dealing with massive datasets, provides us with greater flexibility.

3.1 Input layer

The input layer has only one input neuron. It receives one string at a time from the values of the first column of the dataset, string which is actually words from a text of a tweet.

3.2 Text vectorization layer

The first hidden layer is a Text Vectorization layer. It receives a string from the input layer and is in charge of creating tokens from the string received. Generated tokens are formatted according to the configurations parameters of the layer.

Since we want to retrieve words one by one, we set the layer to lowercase the input string and remove its punctuation. Then the string is split into tokens using the whitespace character as separator. Each token is mapped to its integer and the final output is a list of integer token indices, padded to an output length of a predefined size.

The mapping between tokens and integers has to be either provided through a dictionary or learned by the layer. To learn the mapping the adapt method of the layer is used on the list of texts of the first column. This way a dictionary is autonomously built by searching and storing the most frequent tokens, up to a number equal to a provided dictionary size. Then a unique integer is assigned to each token present in the dictionary according to its frequency.

Two special tokens are reserved in the dictionary: the empty token, which is an empty string and it's useful to be used as padding, and the 'UNK' token, which is used instead of a token with unknown characters or symbols or when the original token is not present in the dictionary, so it's automatically converted to this special default token.

The size of the dictionary for the tokens of the words is fine tuned, according to the quantity of information to store from the words, to have as much predictive power as possible. A starting size of 10k is used.

The number of tokens we set to extract from a text is 50. If we get less we add empty tokens as padding. The decision to use this value as a starting point comes from different data sources. The maximum tweet length, according to the official documentation, is 280 characters. According to Google's Peter Norvig article the average English word length is 4.79. Rounding this number up to 5 and adding 1, which is the whitespace character between each word, results in the length of 6. We divide 280 by 6 and get 46.7, which we round up to 50. Out of 50 words is reasonable to assume that the majority of words in a tweet are actual words while only a small number are hashtags. Then 50 should be enough to extract all the possible words from a text without its hashtags. This value will be eventually fine tuned later.

This layer outputs a list containing the tokens integers.

3.3 Embedding layer

The second hidden layer is the embedding layer. This layer handles the conversion of each integer of the list incoming from the text vectorization layer to a dense float list.

The new lists are lists of float values generated from each integer of the first list, whose sizes are fixed and equal to the chosen embedding dimensions.

The critical parameter to define for this layer is the number of embedding dimensions. It has to be both large enough to preserve all the necessary informations and small enough to keep in check "curse of dimensionality". The TensorFlow Team suggests as a rule of thumb to use as number of dimensions the fourth root of the size of the dictionary containing the tokens to embed. This number will be eventually fine tuned later.

The output of this layer is a list of lists containing the embedding of the integers coming from the text vectorization layer.

3.4 Output layer

This layer receives a list of float lists and outputs the numerical values to use for supervised learning on the forecasting neural network.

The most simple solution lies in using a flattening layer after the embedding layer. Flattening causes the output layer to have a number of neurons equal to the product of the values of the dimensions of the input list. This strategy however has the drawback of producing a large number of output values, meaning that the input and the output layer of the forecasting neural network will have a large number of neurons and the dataset to be fed to this second network becomes extremely large.

An alternative is using a pooling layer to reduce the 2-dimensional list to just a 1-dimensional one. Pooling layers are of two kinds: max pooling layers and average pooling layers. Since the range of values of the embedding goes from +1 to -1, using max pooling means a loss of informations since from the values to be pooled only the maximum ones are kept, clipping the negative range of values and reducing the effectiveness of the embedding. Average pooling instead outputs the average of the values to pool, smoothing the results and preserving informations. It also reduces the size of the output to just the second dimension of the input 2-dimensional list, which is equal to the embedding dimensions of the previous layer.

The choice of the layer to use will be made according to training and testing results since the output of this layer is the input of the forecasting neural network.

4 Labels decomposition

Since the forecasting neural network is trained using supervised training we must provide the expected output labels. Output labels are decompressed from the strings of the second column of the dataset.

A function reads each string containing the indexes of the hashtags present in the corresponding tweet that are also in the frequent hashtags list. It retrieves the integer values of the indexes from the string and then uses them to build an integer binary list, whose length is equal to the one of the frequent hashtags list, which contains 0's where there are no corresponding indices in the retrieved ones and 1's where they're are present instead. Then this list is returned for each string and each value expanded into a new column, generating from a single column multiple columns whose total number is equal to the length of the frequent hashtags list.

Each value of a new column, either 1 or 0, shows where the corresponding hashtag in the frequent hashtags list is present or not in a tweet. The output layer of the forecasting neural network receives a neuron for each hashtag to predict and is in charge to decide which ones of the frequent hashtags are in the tweets, while the corresponding columns are the expected results.

5 Forecasting neural network

This neural network's aim is to forecast if a tweet may have one or more frequent hashtags just from its words.

The rationale behind the first neural network handling the embedding is having a simplified architecture for this neural network. The simpler architecture provides many benefits: faster development, training and testing over massive datasets, reduced risk of overfitting and no vanishing gradient. The forecasting neural network is a feed forward neural network.

5.1 Input layer

The input layer has a number of neurons equal to the output of the embedding neural network. It receives a float list containing the results of the embedding of the words of a tweet. Each neuron handles one value of the list.

5.2 Hidden layers

The exact number of hidden layers and their numbers of neurons will be fine tuned according to training and testing results. However they're all simple dense layers consisting of multiple neurons.

As starting point just two hidden layers are used. The reason behind starting from two instead of one is the simple property that while one hidden layer can model only continuous functions, two hidden layers can model any Riemann-integrable one. Using also just two hidden layers and not more reduces risks of overfitting and vanishing gradient. The exact number of neurons for each layer will be fine tuned according to the result obtained during training and testing. Rule of thumb is their halving going deep.

5.3 Output layer

It is a multilabel classification problem because each input can be assigned to multiple classes. The output layer has a number of neurons equal to the length of the frequent hashtags list and each neuron is in charge of predicting if the current tweet has or not the corresponding hashtag in it.

The activation function for each output neuron must be a sigmoid function. Each sigmoid function in the output layer returns a probability between 0 and 1 that the corresponding frequent hashtag is present in the input tweet. Since now each label has a probability between two and only two binary values, 0 and 1, the loss function used by the model must be a binary cross-entropy function.

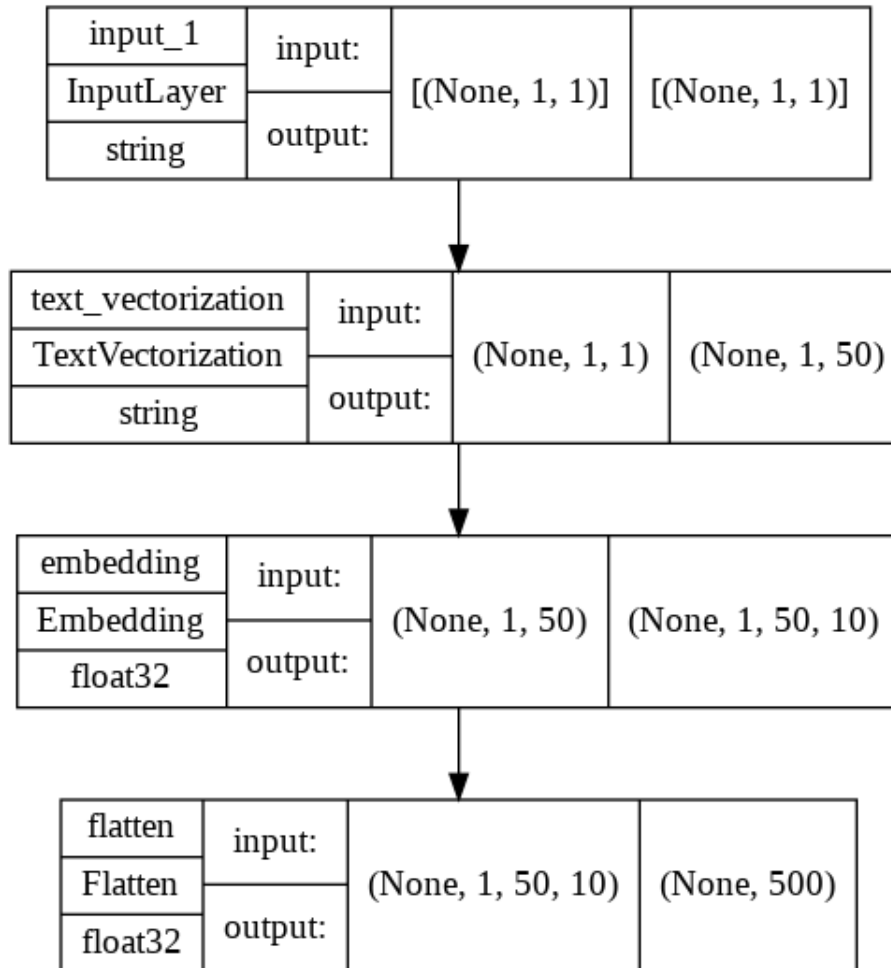
6 Results

Before fine tuning the final forecasting model some parameters still need be fixed. They're adjusted by trial and error and are useful to verify the correctness of our previous assumptions aside from the fixed 100 hashtags to predict.

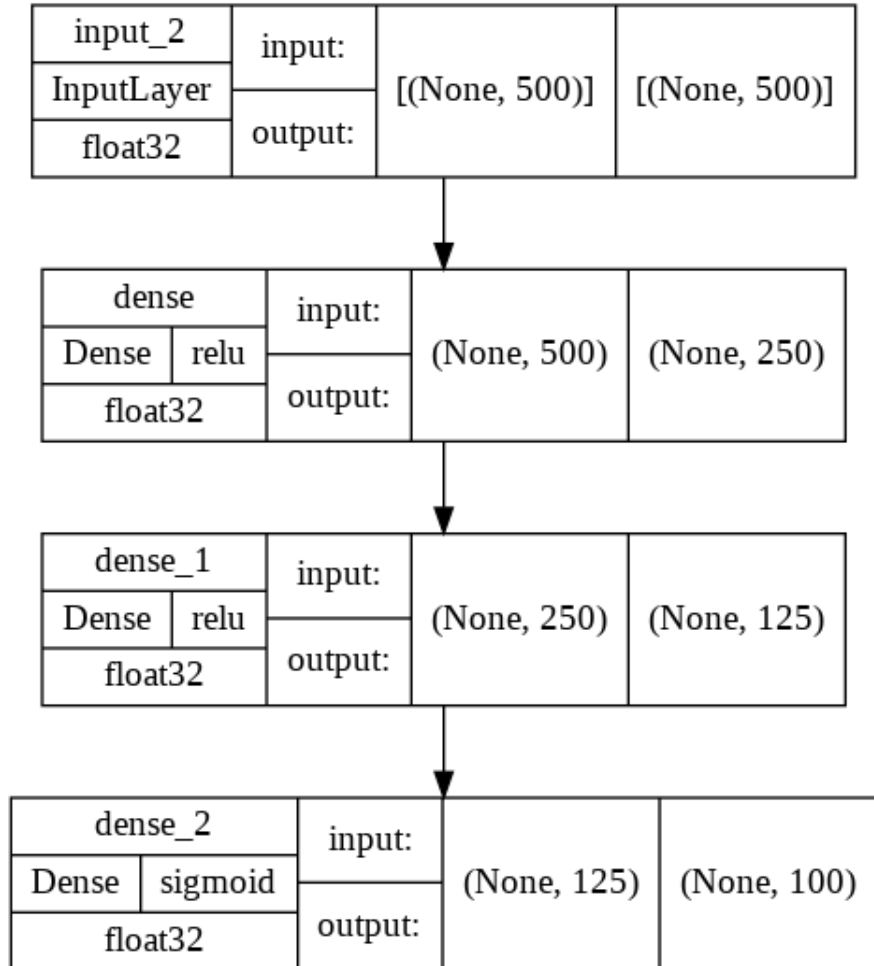
- Using 50 as the number of tokens to extract from a tweet ensures getting all the words available, even in the case of a tweet of maximum length without any hashtag.
- Over a dictionary of 10k words a flattening layer, as output layer of the forecasting neural network, results in an accuracy of 99% while a 2-dimensional pooling layer yields an accuracy of 98%.
- Lowering the dictionary size to 1k or increasing it to 100k doesn't change the results significantly so a size of 10k is used.
- Both kind of layers can be feasible but, even if the pooling layer simplifies the input of the second neural network, the flattening layer is used to reach the maximum possible accuracy.
- The rule of thumb used in the embedding layer to decide the number of embedding dimensions works well with all the different dictionary sizes. The resulting embedding float values are not too close to each other, so enough informations about the words present in the dictionary is stored.
- Using the flattening layer explains the rationale behind assuming a decreasing number of neurons in the hidden layers of the forecasting neural network. Since we're dealing with massive datasets, after the flattening operation made by the embedding neural network, the number of neurons needed to handle the size of the input is considerably higher than the 100 hashtags, each one having an assigned neuron in the output layer of the forecasting neural network. Neurons in each hidden layer are half the ones in the previous layer.

6.1 Neural networks models

6.1.1 Embedding neural network



6.1.2 Forecasting neural network

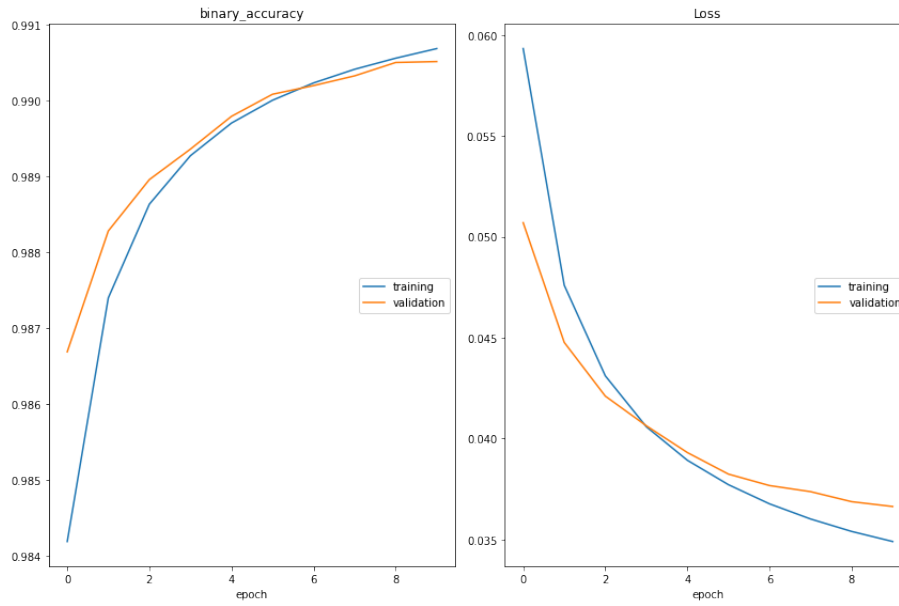


6.2 Performance

A big sample of data is retrieved from the stored dataset and preprocessed to be fed to the forecasting neural network. Then its rows are shuffled and split into training and test data, using a ratio of 70% training data and 30% test data.

6.2.1 Training and validation

Training data is shuffled again and split into an 80% of actual training data and a 20% of validation data. Training is performed in 10 epochs.



Binary accuracy and validation binary accuracy achieved are 99.07% and 99.05%. Binary loss and validation binary loss are of 0.035 and 0.037 respectively with strictly decreasing values so no overfitting occurs.

6.2.2 Testing

On the test set binary accuracy and binary loss are 99% and 0.037 so the model is both accurate and well fitted.