Tweets Sentiment Analysis

Fabiana Monaco fabianamonaco96@gmail.com
Federico Puglisi federico.puglisi10@gmail.com

1. Model Description

We will use recurrent neural networks, and in particular LSTMs, to perform sentiment analysis which consists of considering a text to determine the sentiment behind it. Therefore, in this kind of application we have many sequences as input and we get one label as output. Moreover, in the RNNs, which are particularly useful when we have arbitrary lengths, we have a cell state which receives an input, provides a prediction but it also have a cycle (because the results changes in time).

With the RNNs we need a memory that at each step takes into account the decision of a previous step. In particular, at each time stamp we have a blackbox in order to pass into the other timestamp the information about decision taken in a state vector that encodes this information in a memory and pass it to the next step. If we unfold this vision, we get the RNNs and we unfold as many time as the points we have in our sequence.

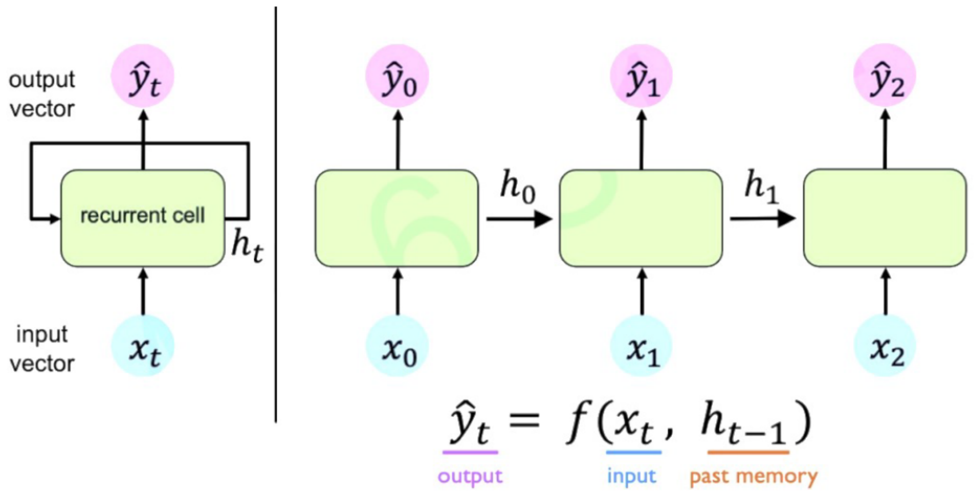

Figure 1: A representation of the Recurrent Neural Networks.

In the construction of the RNNs, it is necessary to pay attention into few design criteria:

- handle variable length sequences;
- track long term dependencies;
- maintain information about order for share parameters across the sequences.

Moreover, we have to find the proper representation for our words and, in particular, we have to convert them into words: this encoding process is called "Embedding" and it transforms indexes into a vector of fixed size.
In our model, we used LSTM which contains computation block that control information flow. In particular, information is added and removed through structures called gates which optionally let information through, for example via a sigmoid neural net layer and pointwise multiplication. It works through several steps:

1. Forget: forget irrelevant parts of the previous state;
2. Store: store relevant information into the cell state;
3. Update: selectively update cell state values;
4. Output: controls what information is sent to the next time step.

We decide to use it in our model because we have obtained more stable results rather than standard RNNs.

2. Dataset

The dataset is a collection of tweets of the famous application Twitter. It contains two files, one for the train set and one for the test set. Each of them contains two columns with tweets and labels, respectively. The train set consists of 73996 tweets, distributed under 4 categories (Positive, Neutral, Irrelevant and Negative).
Instead, distributed under the same categories, in the test set we have only 1000 tweets. Since the train set is much larger than the test set, we have decided to randomly transfer 6000 tweets with their respective labels from the first to the second.
Moreover, after the pre-processing of the data, that we will explain better in the training procedure section, we decide to build the validation set taking 10 percent of the tweets from the train set because the original dataset does not contain it.
After this procedure, in the train we have 61243 tweets, in the validation 6804 tweets, while in the test set we have 6949 tweets.

3. Training procedure

Initially, our dataset present data with special characters, punctuation and emoji, therefore, in the pre-processing phase, we cleaned our tweets and we split each words of them in order to create a unique vocabulary for the entire model.
Regarding labels, as we seen before, we have four sentiment target. For simplicity, the number of sentiment target has been reduced aggregating the "Irrelevant" labels into the "Neutral" ones.
Consequently, after the embedding process, we have associated to "Positive", "Neutral" and "Negative", the number 2,1,0, respectively.
Before the construction of our model, we have created the loaders using a batch size of 256. Then, for the initialization of the model, we set the size of each embedding vector equal to

1024 which is also the number of expected features in the input of the LSTMCell. Moreover, we set the the number of features in the hidden state h equal to 512.
Moreover, in our model we add a softmax layer to converts the output into a probability distribution.
In our training, we used, as optimizer, the Adam algorithm with learning rate set to 0.001 and, as loss function, the Cross Entropy.

4. Experimental Results

The model described in the previous section was trained for 100 epochs, obtaining the following results showed in Table 1:

Model LSTM	Train	Validation	Test
Accuracy	92.84%	83.54%	83.89%
Loss	62.88%	71.52%	71.15%

Table 1: Performance of the model.

Finally, we displayed our results using the confusion matrix:

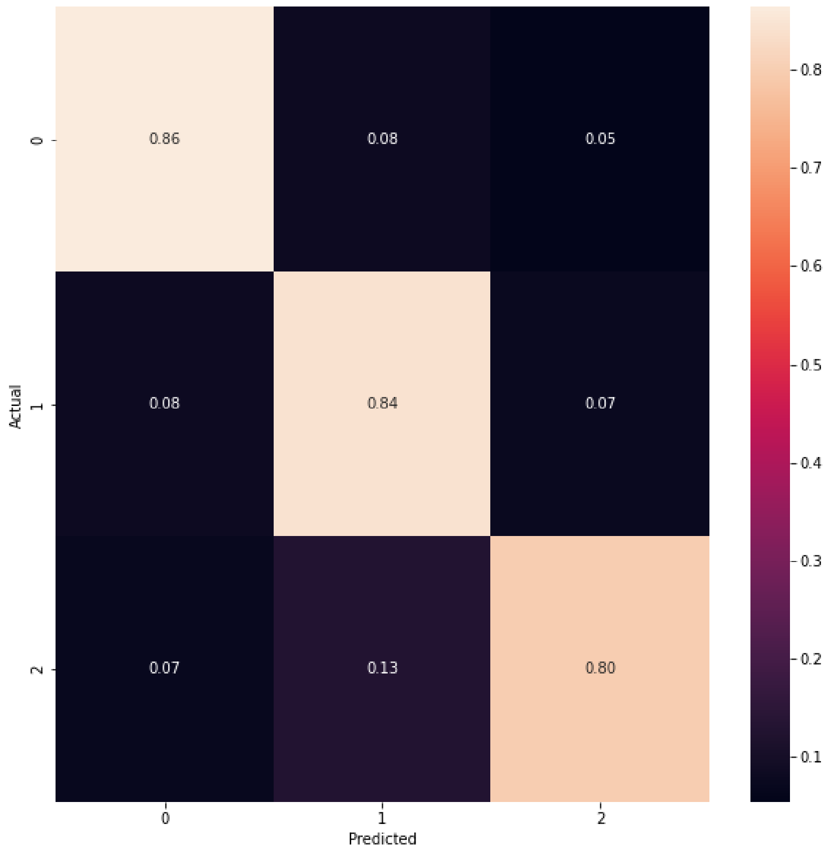

Figure 2: Confusion Matrix.