Twitter Sentiment Extraction

Sara Bertoldo, 588361, sarabert96@gmail.com

Pouria Faraji, 585089, p.faraji@studenti.unipi.it

Niloofar Yousefi, 584643, niloofar.yousefi8@gmail.com

University of Pisa, Pisa, Italy
Department of Computer Science

Abstract. Classical sentiment analysis tasks aim to label a sentence as positive or negative. Our project will go one step further trying to identify which parts of the sentence (that in our case are tweets) contribute in labeling it as neutral, positive or negative. To perform this job, we used a model composed of three layers. The first layer is a BERT transformer while the third layer is a Neural Network. The second layer has been tested with three possibilities: GRU, CNN and GRU + CNN. We will compare the models pointing out the best one, and trying to understand why it obtained the best values.

Keywords: Sentiment Analysis · BERT · GRU · CNN · Supervised Learning

Code The code is available in GitHub:
`https://github.com/Human-Language-technologies/HLT`.

1 Introduction

Traditional sentiment analysis tasks aim to identify if a sentence is *positive* or *negative*, trying to infer its purpose. For example, the sentence *"I just woke up. My teeth really hurt from my rubber bands, really hurt"* is labeled as *negative*, while *"I really really like the song Love Story by Taylor Swift"* as *positive*. This project stands out from the previous because aims to create a model that, knowing if a sentence is *negative, neutral* or *positive*, tries to figure out which words played a role in the marking. In our examples, the key words would be *really hurt* and *like*.

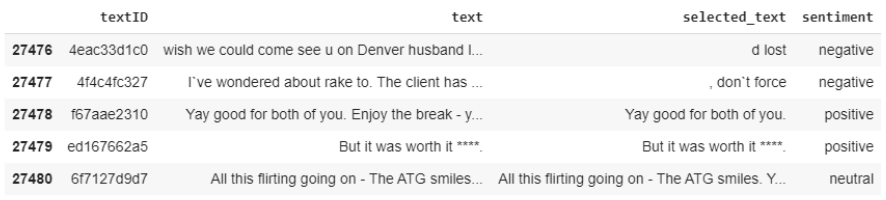
This paper will present the problem in detail, outlining how the structure of the model should be designed and proposing different approaches. We will use four different models: Bidirectional Encoder Representations from Transformers (BERT), Gated Recurrent Unit (GRU), Convolutional Neural Network (CNN) and Neural Network, joining them in different combinations. We will study the obtained results, identifying the best model for our purpose. In the end, we will explain why that model overtook the others, and we will propose further improvements for the task.

2 Problem Statement

In this paper we attempt to pick out the part of the tweet (word or phrase) that reflects the sentiment.

The guidelines for the task came from the Kaggle Competition: *"Twitter Sentiment Extraction - Extract support phrases for sentiment labels"*.[1] The dataset used is titled Sentiment Analysis: Emotion in Text tweets with existing sentiment labels, used under creative commons attribution 4.0. international licence.

Each row of the dataset of this project contains unique ID for each piece of text (*textID*), the text of the tweet (*text*), the general sentiment of the tweet (*sentiment*) and a sub-string of the tweet (a word or phrase) that represent the sentiment (*selected_text*). An example of the dataset is shown in Figure 1.



	textID	text	selected_text	sentiment
27476	4eac33d1c0	wish we could come see u on Denver husband l...	d lost	negative
27477	4f4c4fc327	I`ve wondered about rake to. The client has ...	, don`t force	negative
27478	f67aae2310	Yay good for both of you. Enjoy the break - y...	Yay good for both of you.	positive
27479	ed167662a5	But it was worth it ****.	But it was worth it ****.	positive
27480	6f7127d9d7	All this flirting going on - The ATG smiles...	All this flirting going on - The ATG smiles. Y...	neutral

Fig. 1: Dataset example

The metric used for evaluation is the word-level Jaccard score, in order to calculate the similarity of two strings.

$$J(A,B) = \frac{\mid A \cap B \mid}{\mid A \cup B \mid} \tag{1}$$

In the project we will use different combinations of models: BERT, GRU, CNN, comparing the results.

3 Model structure

In order to predict the selected text of a tweet which ascribes the sentiment, we need to define a proper input shape and structure for the model. Selected text is part of the main text, so to be identified it has starting index and ending index. Therefore, the outputs must represent starting and ending indexes which need to be predicted.

Figure 2 shows our model with input and output if we look at it as a black box.

Now let's dive into the details of our black box model. The model used in this project is composed of three main layers. The output of each layer is the input of the following one. The first layer is a BERT Transformer whose input has the shape: (Batch Size, Sentence Length). Each tweet needs to be tokenized, and we used *bert-base-uncased* version of BERT to do it. We designed the input structure as follows:

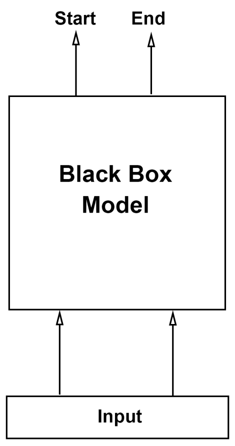


Fig. 2: Black Box Model

1. INIT token
2. ID of the sentiment
3. two padding tokens
4. token IDs of the main tweet
5. EOS token indicating end of sentence

Figure 3 shows an example of how we process a tweet.

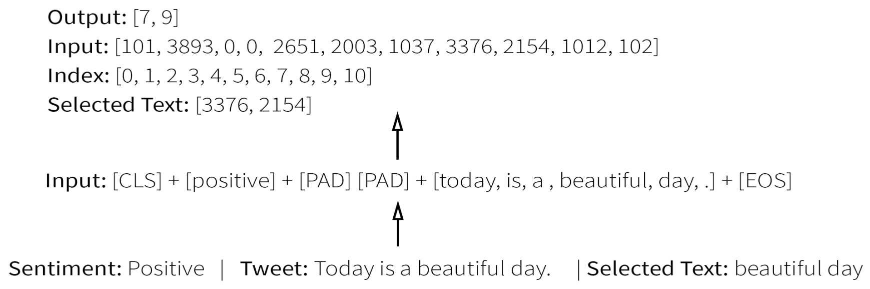

Output: [7, 9]
Input: [101, 3893, 0, 0, 2651, 2003, 1037, 3376, 2154, 1012, 102]
Index: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Selected Text: [3376, 2154]
Input: [CLS] + [positive] + [PAD] [PAD] + [today, is, a , beautiful, day, .] + [EOS]
Sentiment: Positive | Tweet: Today is a beautiful day. | Selected Text: beautiful day

Fig. 3: Process of transforming input

Default dimension of the encoder layers in BERT is 768. After processing the input, BERT generates results with the shape of (Batch Size, Sentence Length, Embedding Dimension = 768).

The second layer in our model takes the BERT outputs as input and process it. We used three different approaches as our second layer, which will be explained in the following sub-sections. As last layer we used a one-layer Neural Network, which as input takes the output of its previous layer and generates two outputs, one for staring position and the other for ending position of selected text, as mentioned before.

Since we have two numerical outputs, we chose Mean Square Error (MSE) to be our criterion as loss function. Moreover, Adam with default parameters is our chosen optimizer.

3.1 GRU

The first approach consisted in using a Gated Recurrent Unit in the second layer, as shown in figure 4. The hyper-parameters in this scenario are batch size,

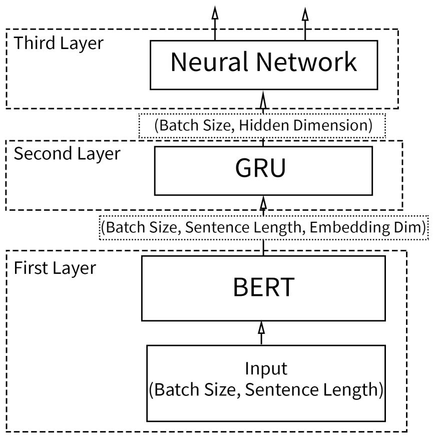

Third Layer
Neural Network
(Batch Size, Hidden Dimension)
Second Layer
GRU
(Batch Size, Sentence Length, Embedding Dim)
First Layer
BERT
Input
(Batch Size, Sentence Length)

Fig. 4: Proposed model with GRU

hidden dimension, number of layers, bidirectionality, and dropout value for the results of GRU. In case of bidirectional, we get two last hidden states of GRU and then concatenate them to be passed to the third layer which as anticipated is a one-layer Neural Network. If it is not bidirectional, only the last hidden state is passed.

Input shape of the GRU is (Batch Size, Sentence Length, Embedding Dimension = 768), and after processing, output shape of the GRU will be (Batch Size, Hidden Dimension).

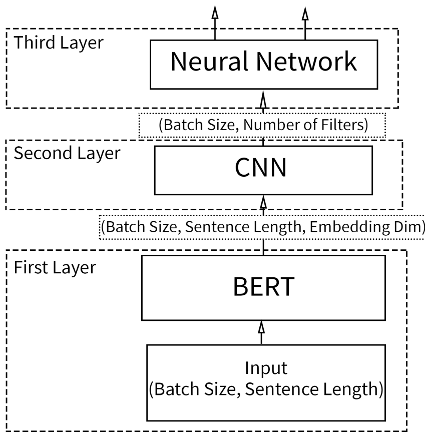
With only GRU, our proposed model has 112M parameters, and since BERT parameters are not updated during training, there will remain only 3M trainable parameters.

3.2 CNN

Our second approach has a Convolutional Neural Network in the second layer of the proposed model as shown in figure 5. In this case hyper-parameters are different from previous approach, which includes batch size, number of filters, kernel size, and dropout. Input channel for the CNN is 1, since we are dealing with a matrix of word embedding. However, kernel size has the following shape: (Size, Embedding Dimension). Embedding dimension here is for the encoder's layers of BERT which is 768.

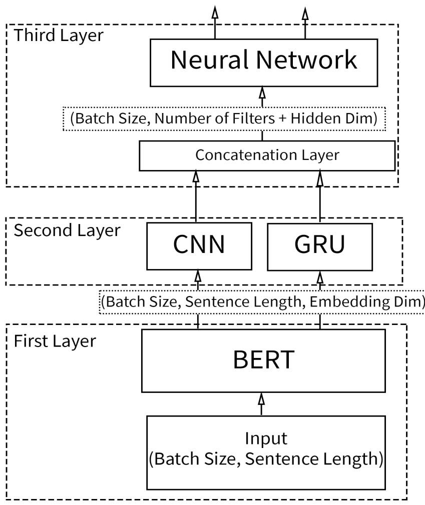
Input shape of the CNN is a 3D matrix of the shape (Batch Size, Sentence Length, Embedding Dimension) and after processing, output shape of CNN which is the input of the last neural network is (Batch Size, Number of filters)

On this one we have approximately 101M parameters, where only 230K of them are trainable. Again, BERT's parameters do not need to be updated.


Third Layer
Neural Network
(Batch Size, Number of Filters)
Second Layer
CNN
(Batch Size, Sentence Length, Embedding Dim)
First Layer
BERT
Input
(Batch Size, Sentence Length)
Fig. 5: Proposed model with CNN

3.3 GRU + CNN

In the third approach, we used a Gated Recurrent Unit joined to a Convolutional Neural Network as the second layer. In this case output of BERT will enter into the GRU and CNN at the same time. After processing, their outputs will be concatenated to go into the last layer. The proposed model is shown in figure 6. Hyper-parameters are more diverse in this approach, because we are using


Third Layer
Neural Network
(Batch Size, Number of Filters + Hidden Dim)
Concatenation Layer
Second Layer
CNN
GRU
(Batch Size, Sentence Length, Embedding Dim)
First Layer
BERT
Input
(Batch Size, Sentence Length)
Fig. 6: Proposed model with GRU and CNN

GRU and CNN at the same time. Hyper-parameters include batch size, hidden dimension, number of layers, bidirectionality of GRU, number of filters, and kernel size of CNN, besides dropout for the concatenation part.

In this case in total we have approximately 112M parameters in which only 3M are trainable and will be updated during the training.

4 Experiments

In order to find the best model in terms of lower loss value and higher Jaccard similarity, we divided our data set into three parts. The first part which is the train set (64%) is used to train our models. Then we have the validation set (16%) which is used to find the best model according to results of loss and Jaccard, and finally, we have the test set (20%) to evaluate our model on new data. Table 1 shows total number of tweets in each data set.

Table 1: Number of tweets in each data set

Train Set	Validation Set	Test Set
17588	4397	5496

Moreover, table 2 shows total number of unique tokens per each column for the whole data set.

Table 2: Number of unique tokens in each column

Text	Selected Text	Label
29435	19431	3

Figure 7 shows the frequency of each sentiment in the whole data set. As it shown in the figure, *neutral* is the most frequent sentiment in our data set.

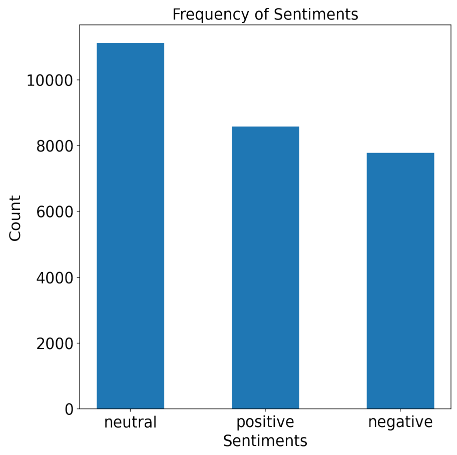


Fig. 7: Frequency of sentiments in the whole data set

For the three approaches proposed in model structure, we tried with different combination of hyper-parameter values. In all of the following experiments, we used 10 epochs. More specifically, with only GRU and only CNN each epoch took approximately 1 and half minute. Whereas, with GRU and CNN next to each other, it took 3 minutes for each epoch.

Table 3: GRU

Batch	Hidden	Layers	Bidi.	Dropout	TL	VL	TJ	VJ
32	256	2	True	0.25	21.672	31.609	43.41 %	43.70 %
128	256	2	True	0.25	24.947	32.230	42.88 %	44.32 %
32	256	1	False	0.25	30.229	31.048	39.59 %	41.75 %
32	256	2	True	0.5	29.464	30.740	41.15 %	44.11 %
32	512	2	True	0.5	24.250	31.189	42.02 %	43.70 %

Batch = batch size, Hidden = hidden dimension, Layers = number of layers, Bidi. = bidirectional, Dropout, TL = train loss, VL = validation loss, TJ = train Jaccard, VJ = validation Jaccard. Output dimension (not showed) is always 2.

Table 3 shows the results of experiments with different GRU hyper-parameters discussed in model structure. As it is evident in the table, having two bidirectional layers of GRU is much better than one uni-directional layer in terms of loss and Jaccard value. Moreover, using lower batch size will lead to lower train and validation loss and higher train and validation Jaccard similarity.

Regarding dropout, we can tell that having higher value of dropout will have better results in terms of loss for the validation set.

Table 4: CNN

Batch	Filters	Filter Size	Dropout	TL	VL	TJ	VJ
32	100	3	0.5	41.290	34.850	33.80 %	38.50 %
32	100	3	0.25	33.226	34.724	36.30 %	38.57 %
32	50	3	0.25	36.825	34.946	35.48 %	37.87 %
128	100	3	0.5	44.537	34.283	33.64 %	36.55 %
32	100	5	0.5	38.261	33.796	33.98 %	38.55 %

Batch = batch size, Filters = number of filters, Filter Size, Dropout, TL = train loss, VL = validation loss, TJ = train Jaccard, VJ = validation Jaccard. Output dimension (not showed) is always 2.

Table 4 shows the results for train and validation set with their loss and Jaccard values, using different combinations of hyper-parameters discussed in CNN subsection of model structure. Having lower number of filters in CNN might not be a good idea, because loss is higher and Jaccard is lower. Moreover, increasing the batch size did not improve the results.

Table 5 shows the results for our third approach which consists of GRU next to a CNN. Here we fixed number of layers of GRU to 2 and set as bidirectional. Different combinations of other hyper-parameters were experimented.

Table 5: GRU+CNN

Batch	Hidden	Dropout	Filters	Filter Size	TL	VL	TJ	VJ
128	256	0.5	100	3	27.723	30.779	40.45 %	43.35 %
128	256	0.25	100	3	22.871	31.798	43.06 %	44.01 %
32	256	0.25	100	3	27.633	30.191	41.42 %	43.82 %
32	512	0.5	100	3	28.003	30.069	39.58 %	43.31 %
128	256	0.5	50	3	27.623	31.473	40.40 %	43.92 %
128	512	0.5	100	5	29.002	30.543	39.97 %	43.52 %

Batch = batch size, Hidden = hidden dimension, Dropout, Filters= number of filters, Filter Size, TL = train loss, VL = validation loss, TJ = train Jaccard, VJ = validation Jaccard. Output dimension and number of layers (not showed) are always 2. Bidirectional is always True.

5 Results

Comparing the tables obtained from the hyper-parameter tuning, we can see that in each model the values are stable within a very narrow range.

The best results for validation loss and Jaccard score from each model are listed in table 6

Table 6: Best results for each model

Model	Val. Loss	Val. Jaccard	Baseline Loss	Baseline Jaccard
GRU	30.740	44.11 %	90.930	16.14 %
CNN	33.796	38.55 %	90.690	16.73 %
GRU + CNN	30.069	43.31 %	89.089	16.38 %

Comparing the losses and Jaccard scores with the random baselines, our results are far better. Even if we did not obtain very small losses on one hand and very high scores on the other, we can be satisfied. Although the three models obtained similar values, we can notice how the model which performs better in this task is GRU. This outcome is confirmed by the plots of figure 8 and 9 which show the learning curve of the models.

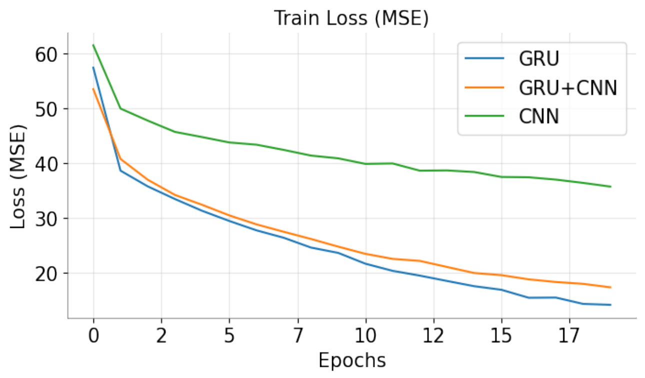


Fig. 8: Training Loss curves

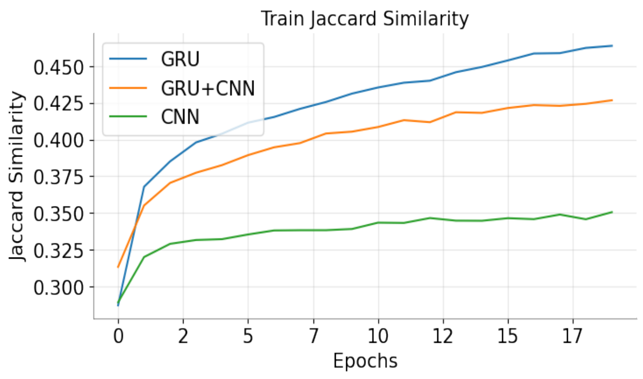


Fig. 9: Training Jaccard curves

Generally speaking, CNNs are hierarchical and RNNs sequential architectures. Which architecture performs better depends on how important it is to semantically understand the whole sequence. An architecture like GRU handles long sequences correctly but can lose information on short key parts. In addition, as GRU chooses the last hidden state to represent the sentence, if the meaning of the last part of the sentence contrast with the rest of it, this might result in the wrong prediction. The state-of-the-art research tell us that GRU and CNN are comparable when lengths are small, but GRU gets increasing advantage over CNN when meet longer sentences [2].

Our task would have prompted us to think that the best model to use was CNN, since GRU neglect the meaning of single words. The experiment conversely showed us that this is not the case, and underlined how the whole sentence is important in understanding the selected text. Moreover, the inputs fed to BERT contains a lot of zeros after EOS token, therefore, CNN tries to process all of these extra noise which may affect its performance. One thing that could improve CNN performance was using an attention mask to remove the effect of those excess zeros.

6 Conclusion

Comparing the results obtained, we can notice that they are all similar. There is not a model which outperforms the others, however the model composed by BERT, GRU and NN is the one which obtained the best outcomes. Looking at the values with respect to baselines, we can be satisfied by the scores.

There are a few improvements that can be done to improve the performance of the proposed model. First, as it is evident from the training curves, having more epochs leads to lower loss value and higher Jaccard similarity. However, we should take into account the fact that having lower loss value in training doesn't necessarily mean better results on validation and test set, and probably it is an over-fitting situation.

Next thing we can do is to consider only the relevant part of the input by applying an attention mask. In the current version of our model, all inputs in the batch have the same length, which is the one of the longest sentence in the batch. Shorter inputs are forced to be zero padded, thus, adding extra useless information to each sentence. Using an attention mask would remove this effect leading to better results.

Using BERT is not the only option. There are plenty of other Transformer models which are great for Natural Language Processing purposes[3]. One noteworthy is RoBERTa, which had great performance from notebooks of other users in the Kaggle competition.

Another helpful task that can be done is data cleaning. One issue here is that sentiments are not binary but rather there are three classes and the majority of them are *neutral*. We guess if we had only *positive* and *negative*, performance would have been better. Moreover, being this a data set of tweets, it has plenty of words written not in standard way but rather in a very informal way (for example *B4* to say *before* or *thanx* to say *thanks*), which makes process of sentences more difficult. Also, some selected texts in data set had problems. They either contained the whole text (the complete tweet) or they contained some meaningless sub-string of the tweet, which again makes the process more cumbersome.

References

1. Twitter Sentiment Extraction - Extract support phrases for sentiment labels, `https://www.kaggle.com/c/tweet-sentiment-extraction/overview`. Last access: July, 2020.
2. Yin, Wenpeng Kann, Katharina Yu, Mo Schütze, Hinrich. (2017). Comparative Study of CNN and RNN for Natural Language Processing. `https://www.researchgate.net/publication/313443663_Comparative_Study_of_CNN_and_RNN_for_Natural_Language_Processing`. Last access: July, 2020.
3. Transformers `https://huggingface.co/transformers/`. Last access: July, 2020.