

TOXIC-SENTIMENT ANALYSIS FOR LIVE CHAT

Victoria Grosu
881177

Chiara Pesce
882078

Francesco Perencin
880106

Tommaso Talamo
875496

**DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING
(CM90) - a.a. 2023-24**



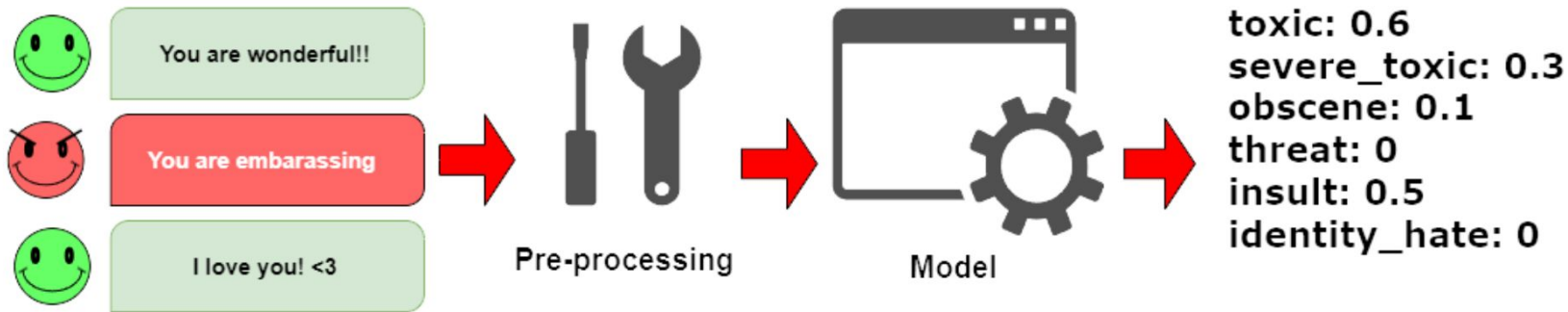


Introduction





HIGH-LEVEL PIPELINE



The aim is to identify and classify toxic comments in real-time conversations.



DATASET

Dataset from Toxic Comment Classification Challenge designed for multi label task.
It is a dataset of comments from Wikipedia's talk page edits and it is divided into:

- **train.csv**: the training set, contains comments with their binary labels
- **test.csv**: the test set, you must predict the toxicity probabilities for these comments. To deter hand labeling, the test set contains some comments which are not included in scoring.
- **test_labels.csv**: labels for the test data; value of -1 indicates it was not used for scoring.

The types of toxicity are:

- toxic
- severe toxic
- obscene
- threat
- insult
- identity hate



PRE-PROCESSING

The pre-processing is composed by different functions:

- **precleaning**: Initial cleanup of text data, usually involving lowercasing and removing non-essential characters.
- **clean hashtags**: Remove hashtags from the text.
- **filter chars**: Remove unwanted characters, such as emojis or special symbols.
- **truncate text**: Shorten text to a specified length.
- **remove url shorteners**: Remove URLs, including shorteners, from the text.
- **perform stemming**: Reduce words to their root form using stemming.
- **perform lemmatization**: Reduce words to their base form using lemmatization.
- **remove repeated punctuation**: Remove repeated punctuation characters.



TOOLS

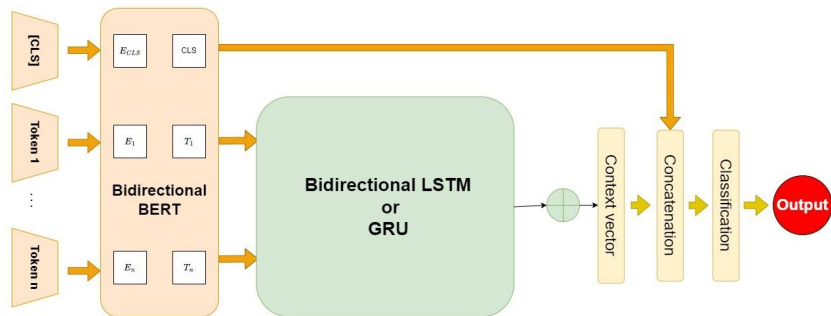
The development was carried out on [Google Colab](#), utilizing the powerful capabilities of PyTorch. The main tools and libraries used are:

- Pandas
- NumPy
- NLTK
- Hugging Face
- Scikit-Learn
- Torch



what's our idea ?

with CLS TOKEN



without CLS TOKEN





APPROACHES

Four different approaches:

1. BERT + BiLSTM + Bahdanau attention + [CLS] Token
2. BERT + BiLSTM + Bahdanau attention (Without [CLS] Token)
3. BERT + BiGRU + Bahdanau attention + [CLS] Token
4. BERT + BiGRU + Bahdanau attention (Without [CLS] Token)

Models



BERT + FINE TUNE

We use '**bert-base-uncased**' pre-trained model that is a variant of the BERT.

- **Uncased Model:** This means that the text is converted to lowercase before being processed.
- **Base Version:** The base model has:
 - 12 layers (transformer blocks)
 - 768 hidden units per layer
 - 12 attention heads
 - Total of 110 million parameter

Fine-tuning Phase:

We fine-tuned the bert-base uncased for sentiment analysis classification using our dataset.

The final model is: '**bert_sequence_classification_trained**'.



LSTM and GRU

We utilized two different types of Recurrent Neural Networks (RNNs): Gated Recurrent Units (GRU) and Long Short-Term Memory (LSTM) networks.

For both GRU and LSTM, we used the following parameters:

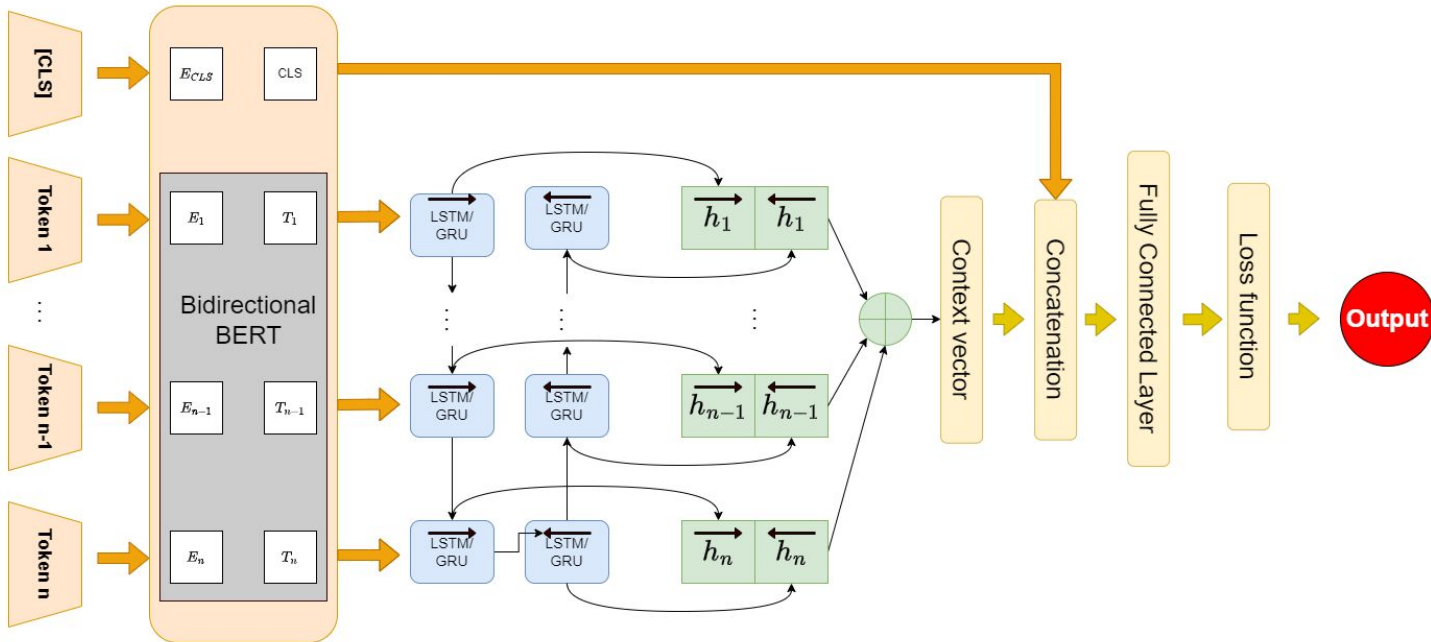
- embedding dim = 768,
- hidden dim = 256,
- output dim = 6

To train and evaluate the models, we used two different **loss functions**:

- BCEWithLogitsLoss
- MultiLabelSoftMarginLoss

We train our model using 30 epochs.

Architecture pipeline with CLS





BCEWithLogitsLoss

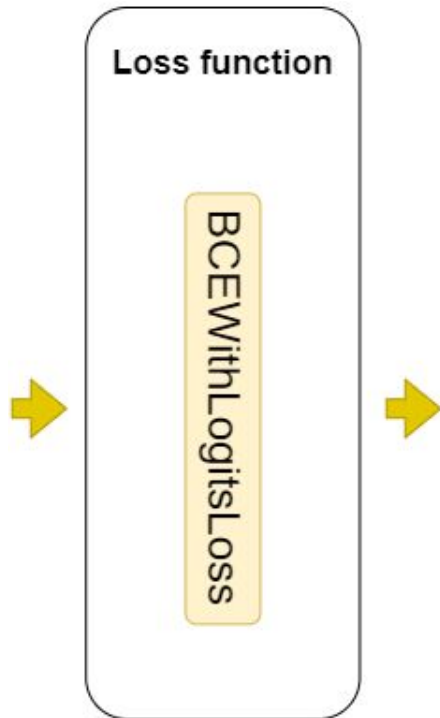
Combines the sigmoid activation function and BCE loss in a single layer.

$$\text{BCEWithLogitsLoss}(x, y) = \text{BCE}(\sigma(x), y)$$

Which is equivalent to:

$$\text{BCEWithLogitsLoss}(x, y) = -[y \log(\sigma(x)) + (1 - y) \log(1 - \sigma(x))]$$

In general, this is a numerically more stable solution than a plain Sigmoid function followed by a Binary Cross Entropy loss.

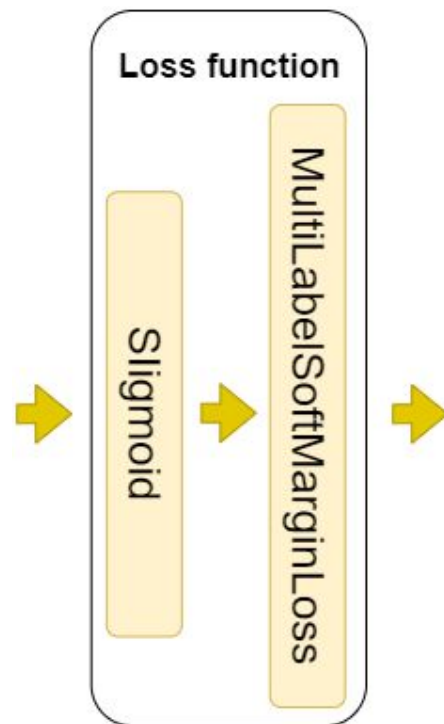


MultiLabelSoftMarginLoss

Given the number of instances N and the number of labels C , we compute the sigmoid of the input and then process it with the loss function:

$$\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^C -[y_{ij} \log(\sigma(x_{ij})) + (1 - y_{ij}) \log(1 - \sigma(x_{ij}))]$$

Unlike BCELoss, this function takes into account the presence of multiple classes that are not mutually exclusive.



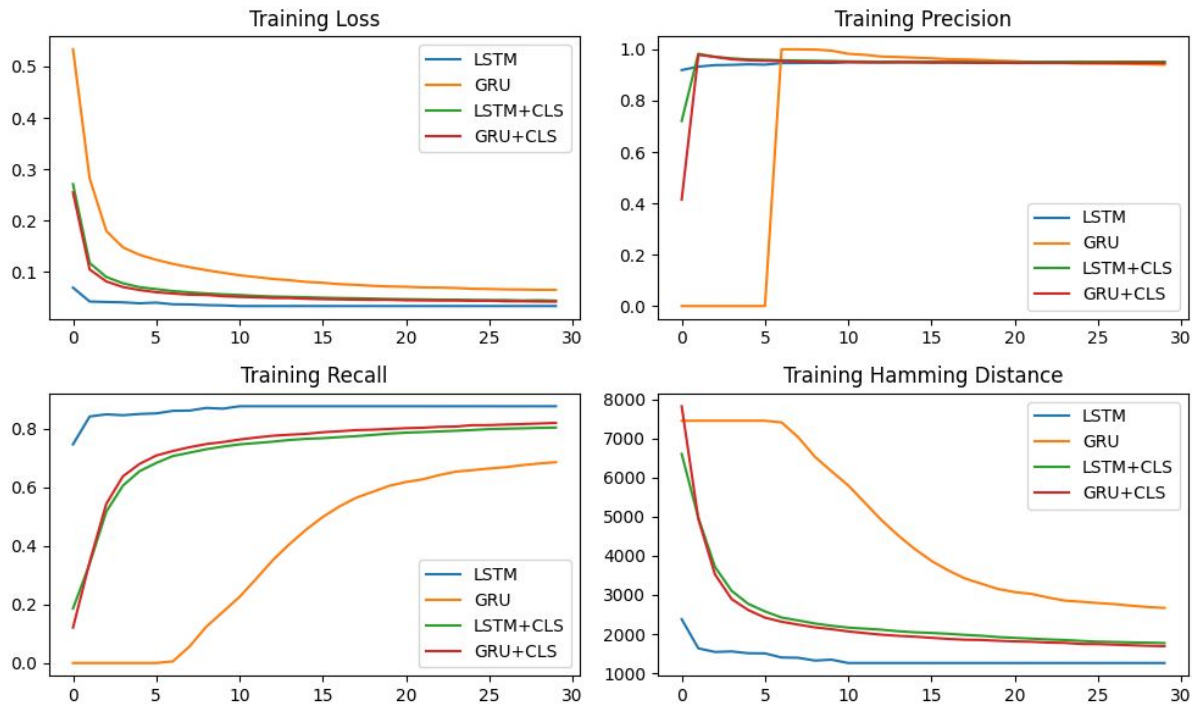


Results training



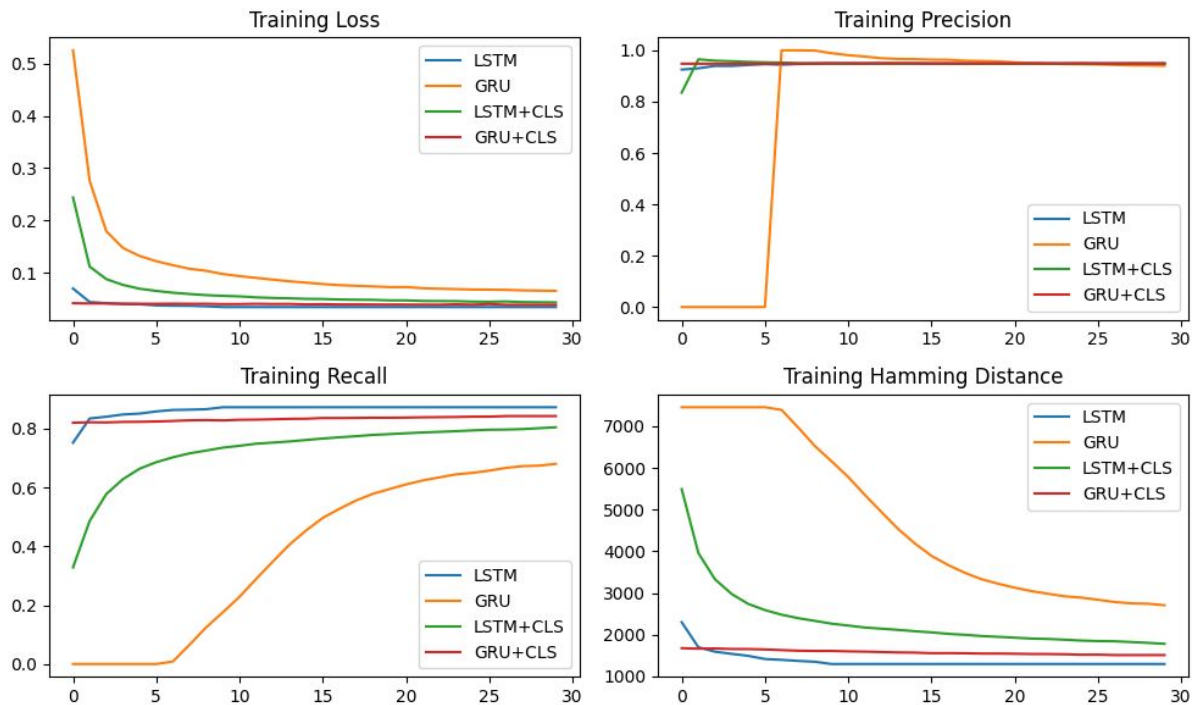


Training with BCEWithLogitsLoss





Training with MultiLabelSoftMarginLoss





Results test





Result with MultiLabelSoftMarginLoss

	LSTM	GRU	LSTM+CLS	GRU+CLS
Accuracy	0.8849	0.8703	0.8855	0.8876
Precision	0.8610	0.9172	0.8984	0.8866
Recall	0.7345	0.6236	0.6968	0.7168
F1-score	0.7927	0.7424	0.7849	0.7927
Hamming Distance	5494	6189	5463	5362
Time (seconds)	24.751	21.618	25.370	21.241
Average Time (ms)	3.02	2.64	3.10	2.60



Result with BCEWithLogitsLoss

	LSTM	GRU	LSTM+CLS	GRU+CLS
Accuracy	0.8832	0.8704	0.8861	0.8879
Precision	0.8725	0.9168	0.9022	0.8963
Recall	0.7146	0.6245	0.6953	0.7078
F1-score	0.7857	0.7429	0.7854	0.7910
Hamming Distance	5575	6182	5436	5350
Time (seconds)	25.004	21.252	24.993	20.749
Average Time (ms)	3.06	2.60	3.05	2.54



Results comparison

Accuracy: The best accuracy is achieved by the GRU + CLS model with BCEWithLogitsLoss, but it is also very similar for the other models. Adding CLS to all models increases accuracy.

Precision: The highest precision is obtained with the GRU model using MultilabelSoftMarginLoss. Precision is worse in LSTM models.

Recall: The best recall is with LSTM models, especially with MultilabelSoftMarginLoss.

F1 Score: It is fairly good for all models, but there is an increase in F1 score in GRU models when CLS is added.

Hamming Distance: The best model consistently has been GRU + CLS. Adding CLS to all models decreases the hamming distance.

Fastest Model: The GRU + CLS model with BCEWithLogitsLoss is the fastest both in terms of total execution time and average time.



Best and Worst models

Best Overall Model: GRU + CLS with BCEWithLogitsLoss.

Reason: It achieves the highest accuracy, very high precision, a good F1 score, and the lowest Hamming distance. Additionally, it is the fastest model in terms of both total execution time and average time.

Worst Overall Model: GRU with MultiLabelSoftMarginLoss

Reason: has the worst accuracy, recall, and F1-score among all models. The Hamming distance is the highest, indicating greater dissimilarity between predicted and actual labels. Although its total execution time and average time are not the worst, its performance metrics are significantly lower compared to other models.



Result competitor

	CNN1D	CNN-V	CNN-I	BiLSTM	BiGRU
Accuracy	0.90	0.91	0.89	0.90	0.90
Precision	0.68	0.73	0.63	0.72	0.68
Recall	0.18	0.3	0.69	0.6	0.68
F1-score	0.28	0.41	0.64	0.64	0.67



Comparison with competitor

Accuracy Comparison:

While their accuracy is marginally better, the difference is slight.

Precision Comparison:

Our models consistently outperform theirs in precision, achieving approximately 0.90 across all models, whereas their best precision reaches only 0.74.

Recall Comparison:

Nearly all our recall values surpass their best recall performance.

F1 Score Comparison:

Across the board, our models demonstrate superior F1 scores compared to theirs.

Hamming distance Comparison:

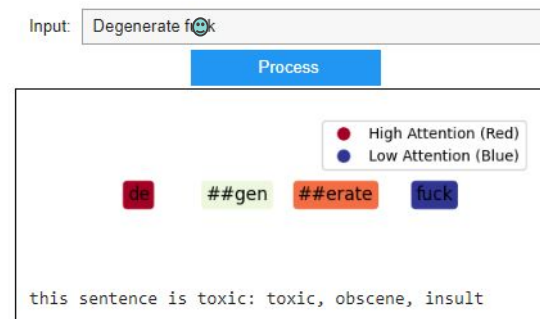
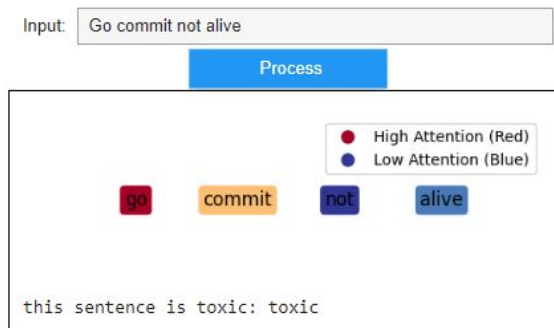
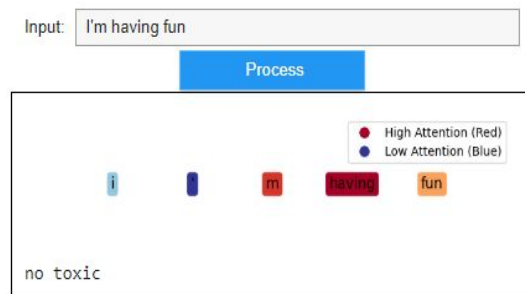
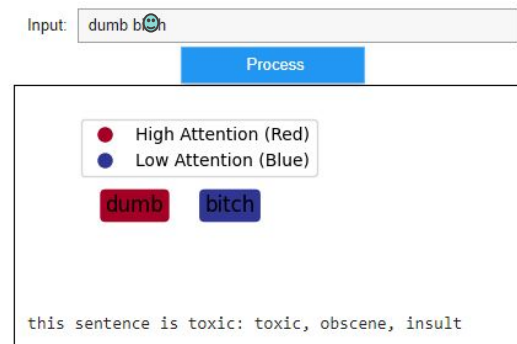
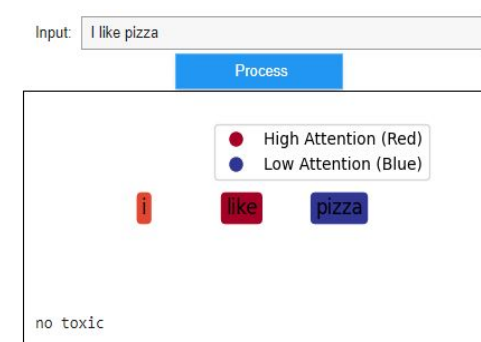
In the competitor's paper, there is no mention of the Hamming distance.

Time Comparison:

In our comparison, it's important to highlight that we couldn't assess their execution time directly because it wasn't detailed in their paper.

Competitor: Overlapping Toxic Sentiment Classification using Deep Neural Architectures

Some examples done right!



Some examples done wrong...

Input: sjfrndkienAS😊OLEfndjfnend

Process

no toxic



Input: I hate pizza

Process

no toxic




this sentence is toxic: toxic

Input: I like met

Process

no toxic





Potential developments





Potential Developments

- more data
- data augmentation
- advanced features extraction
- develop other models
- making models robust to sarcasm and masked words

Thank You!

