PROJECT WORK IN BIG DATA - KNOWLEDGE ENGINEERING

A COMPARISON OF NEURAL NETWORKS ARCHITECTURES FFNS, LSTMS AND GRUS WITH STATIC GLOVE AND CONTEXTUAL BERT EMBEDDING FOR NLP SENTIMENT ANALYSIS TASK.

Dario Cioni, Mohamed Salah Jebali

Matricola: 7073911, 7078487

School of Engineering - Master of Engineering Degree in Artificial Intelligence

Università degli Studi di Firenze

Luglio 2022

Contents

1 Introduction **1**

2 Data set description **2**

2.1 Disaster tweets data set 2

2.2 Trip Advisor Hotel reviews data set 2

2.3 News headlines data set 3

3 Neural Networks involved: a brief theory description **4**

3.1 LSTM 4

3.2 GRU 5

3.3 Bidirectional RNNs 6

4 Design of the used Neural Networks **7**

4.1 Feed-forward Network 7

4.2 LSTM 8

4.3 GRU 9

4.4 Transformers 10

5 Static Embedding: GloVe **11**

5.1 GloVe static embedding 11

6 Contextual Embedding: BERT **13**

6.1 BERT contextual embedding 13

7 Experiments **16**

7.1 Code 16

7.2 Data pre-processing 16

7.3 Loss function 17

7.4 Experimental setting 18

7.5 Results 19

7.5.1 Experiment 1 results: best model 19

7.5.2 Experiment 2 results: best static embedding paradigm 21

8 Conclusions **21**

ABSTRACT

In this project, we modeled different neural networks (NNs) architectures to solve the sentiment analysis task on different data sets.We modeled 3 different types of NNs with different embedding architectures: FFNs, LSTMs and GRUs. We adopted 3 different embedding paradigms: an embedding trained on the original corpus of the application data set, a static pre-trained embedding (GloVe), and a contextual pre-trained embedding (BERT). We conducted then several experiments to compare the models' performance and the best embedding paradigm. We then came to the results that BERT was the best model for the NLP sentiment analysis for 2 out of 3 data sets, while for the other data set the best models were LSTM and FFN.

Keywords Natural Language Processing · NLP · Sentiment Analysis · FFN · LSTM · GRU · GloVe · BERT · embeddings · Python · Keras · Tensorflow

1 Introduction

In this work, we aimed to solve the **sentiment analysis** task by modeling different neural networks with different embeddings architectures.

Sentiment analysis is the use of natural language processing, text analysis, computational linguistics and biometrics to systematically identify, extract, quantify and study affective states and subjective information.

To solve the task, we modeled several artificial neural networks with different architectures: Feed Forward Network (FFN), Artificial Neural Network with Short-Term Memory (LSTMs) and Gated Recurrent Units (GRUs).
Each network was characterized by 3 different embeddings architectures:

- One embedding layer trained on the corpus of the data set;
- A static embedding layer pre - trained on different corpus. In this case we used **GloVe**.
- A contextual embedding layer pre - trained on different corpus and fine tuned on the actual corpus of the data set. In this case we used **BERT**.

The **goal** of the project was to solve the sentiment analysis task and analyze the effect of the embedding layer in the performance of the artificial neural networks used.

To model the networks, we used the **Python** programming language paired with the **TensorFlow** and **Keras** libraries. The code is available here.

We conducted several tests on 3 different data sets to evaluate the performance of neural networks:

- Disaster tweets data set.
- Trip Advisor Hotel reviews data set.
- News headlines data set for sarcasm detection.

Outline The report is divided as follows: in the Section 2 we give a description of the data set used for the experiments; in the Section 3 and 4 we describe the architecture and the design of the artificial neural networks that we used; in the Section 5 and 6 we define what Static Embeddings and Contextual Embeddings are, while in the in Section 7 we give a description of the experiment set up and the results obtained.

2 Data set description

2.1 Disaster tweets data set

Twitter has become an important means of communication, allowing people to be able to announce and share emergency situations in real time. For this reason, it is interesting to be able to create an automated system that can analyze tweets in real time and determine their sentiment.

This data set contains Tweets that may or may not refer to disasters. The goal of nlp models in this case is to predict whether the Tweets are about a disaster or not.
The data set has the following characteristics:

- **Language:** English.
- Each sample in the data set has the following information:
  - The **text** of a Tweet;
  - A **keyword** from that tweet (although this may be blank!);
  - The **location** the tweet was sent from (may also be blank).
- **Number of samples:** 7613.
- **Slightly Unbalanced:** 3271 samples belongs to class 1 (disaster) while 4342 belongs to class 0.
- **Max number of character per sample:** 157.
- **Other:** about 100 samples contain dirty characters that don't belong to the ASCII format.

Acknowledgments This data set was created by the company figure-eight and originally shared on their website.

2.2 Trip Advisor Hotel reviews data set

Trip Advisor has undoubtedly become one of the major means of being able to review businesses and services around the world.
It is crucial for businesses to have the opinion of customers, whether they are positive or negative.
Therefore, a sentiment analysis system that can determine whether a review is negative or positive can be of great use to businesses.

This data set contains hotel reviews written on Trip Advisor and has the following characteristics:

- **Language:** English.
- Each sample in the data set has the following information:
  - The **number** of a review;
  - The **text** of the review;
  - The **rating** of the review that goes from 1 to 5.
- **Number of samples:** 20491.
- **Very Unbalanced:** 15093 samples belongs to class 1 (positive) while 5398 belongs to class 0.
- **Max number of character per sample:** 12738.
- **Dichotomized**: since the reviews' rates go from 1 to 5 but we are doing binary classification for the sentiment analysis task, we've dichotomized the data set in two classes: ratings that are **greater** than 3 are classified as **positive** and belongs to class 1; otherwise, they're classified as **negative** and then belongs to class 0.
- **Other:** It doesn't contain *html* tags or characters that don't belong to ASCII format.

Acknowledgments Alam, M. H., Ryu, W.-J., Lee, S., 2016. Joint multi-grain topic sentiment: modeling semantic aspects for online reviews. Information Sci- ences 339, 206–223.
It can be found here.

2.3 News headlines data set

Hyphenating a sentence of a sarcastic nature is an arduous task. Moreover, if sarcasm is embedded within news and news articles, figuring out whether these articles are fake news or real news becomes even more complicated. The challenge is to model a neural network that can determine whether a news item is sarcastic or not based on its headline.

The data set used contains news headlines.
To overcome the limitations related to noise in Twitter data set, this **News Headlines data set for Sarcasm Detection** is collected from two news website: **TheOnion** and the **HuffPost**.
TheOnion aims at producing sarcastic versions of current events and they collected all the headlines from News in Brief and News in Photos categories (which are sarcastic). They also collected real (and non-sarcastic) news headlines from HuffPost

This data set has the following advantages over the existing Twitter data set:

- Since news headlines are written by professionals in a formal manner, there are no spelling mistakes and informal usage. This reduces the sparsity and also increases the chance of finding pre-trained embeddings.
- Furthermore, since the sole purpose of TheOnion is to publish sarcastic news, we get high-quality labels with much less noise as compared to Twitter data set.
- Unlike tweets which are replies to other tweets, the news headlines we obtained are self-contained. This would help us in teasing apart the real sarcastic elements.

The data set has the following characteristics:

- **Language:** English.
- Each sample in the data set has the following information:
  - The **sarcasm:** 1 if the record is sarcastic, 0 otherwise;
  - The **text** of the news' headline;
  - The **article link:** link to the original news article. Useful in collecting supplementary data.
- **Number of samples:** 26709.
- **Slightly Unbalanced:** 11724 samples belongs to class 1 (sarcastic) while 14985 belongs to class 0.
- **Max number of character per sample:** 254.
- **Other:** It doesn't contain *html* tags or characters that don't belong to ASCII format.

Acknowledgments The data set is from Misra and Arora (2019) and for more information visit the following link.

3 Neural Networks involved: a brief theory description

In this report we used three types of units in our models: feed forward units, recurrent units (LSTM and GRU) and transformers. We provide a brief theoretical overview of the recurrent units

3.1 LSTM

The LSTM unit divides the context management into two sub-problems: removing information no longer needed and adding information likely to be needed for later decision making.

LSTM adds an explicit context layer to the architecture and uses specialized neural units that use *gates* to control the flow of information into and out of the units. All the gates in LSTM consist of a feed-forward layer followed by a sigmoid activation function, followed by a point wise multiplication of the layer being gated. It is possible to observe a scheme of an LSTM unit at Figure 1.

The *forget gate* deletes information from the context that is no longer needed

$$\Gamma_f = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{1}$$

The *update gate* selects the information to add to the current context

$$\Gamma_u = \sigma(W_u[h_{t-1}, x_t] + b_u) \tag{2}$$

The *output gate* is used to decide what information is required for the current hidden state

$$\Gamma_o = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3}$$

To compute the information to be extracted from the previous hidden state and the current inputs, which can be referred as a new *candidate cell state*, a simple linear combination followed by an activation function ($\tanh$) is performed.

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c) \tag{4}$$

The new state is calculated as a combination of the candidate state and the previous cell state, each one gated respectively by the update gate and the forget gate

$$c_t = \Gamma_u * \tilde{c}_t + \Gamma_f * c_{t-1} \tag{5}$$

The information required for the output hidden state is calculated as the new cell state, gated by the output gate.

$$h_t = \Gamma_o * c_t \tag{6}$$

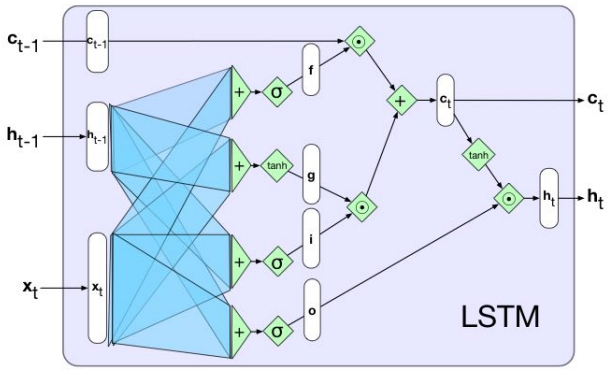


Figure 1: A single LSTM unit displayed as a computation graph.

3.2 GRU

At each time step we consider a new candidate state $\tilde{c}_t$ which can substitute the cell state c_t, calculated as an activation function ($\tanh$) over a combination of the previous cell state and the input with a gate Γ_r which gives a level of relevance of the new information

$$\tilde{c}_t = \tanh(W_c[\Gamma_r * c_{t-1}, x_t] + b_c) \tag{7}$$

$$\Gamma_r = \sigma(W_r[c_{t-1}, x_t] + b_r) \tag{8}$$

The update gate is computed as a sigmoid function over a linear combination of the previous cell state and the input.

$$\Gamma_u = \sigma(W_u[c_{t-1}, x_t] + b_u) \tag{9}$$

This is crucial for the cell state change: the update gate decides wether to update the state

$$c_t = \Gamma_u * \tilde{c}_t + (1 - \Gamma_u) * c_{t-1} \tag{10}$$

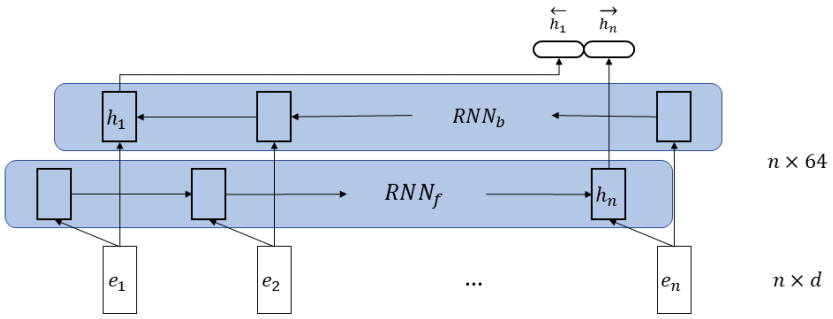
The output hidden state of the GRU is equal to the cell state c_t

$$h_t = c_t \tag{11}$$

3.3 Bidirectional RNNs

Causal RNNs use information from the left (prior) context to make the predictions at time t, but in applications where we have access to the entire input sequence we would like to use words also from the context to the right.

Bidirectional RNNs are composed of two separeate RNNs, going respectively left-to-right and right-to-left and concatenate their representations.



$$h_t^f = RNN_{\text{forward}}(x_1, \ldots, x_t) \tag{12}$$

$$h_t^b = RNN_{\text{backward}}(x_1, \ldots, x_t) \tag{13}$$

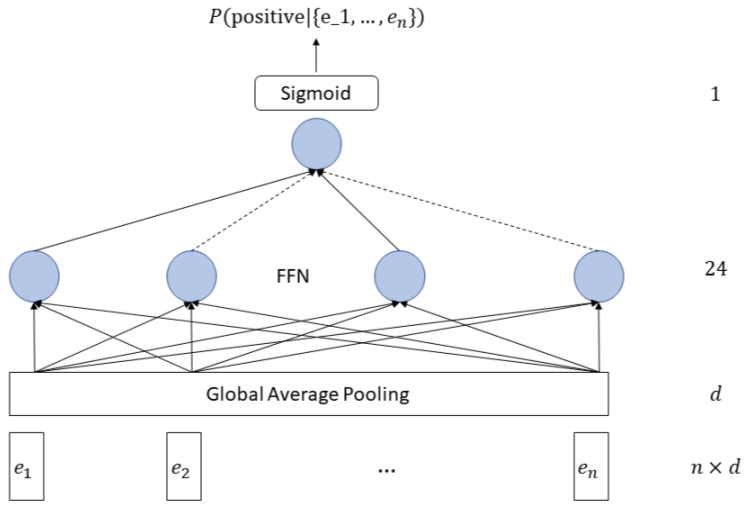
$$h_t = h_t^f \oplus h_t^b \tag{14}$$

4 Design of the used Neural Networks

We provide a description of the models we have evaluated in the experiments and a snippet of the code that produced the models. The models are then evaluated in Section 7

4.1 Feed-forward Network

The first model we designed is a simple Feed-Forward Network, composed by two feed forward layers. In this model the input embeddings are passed to a Global Average Pooling layer, which produces an average on each embedding dimension of the input embedding. This is then followed by a Fully Connected Layer with a ReLu activation and a single output layer with a sigmoid activation, which finally predicts a value between 0 and 1, which is used as the sentiment prediction of the input.

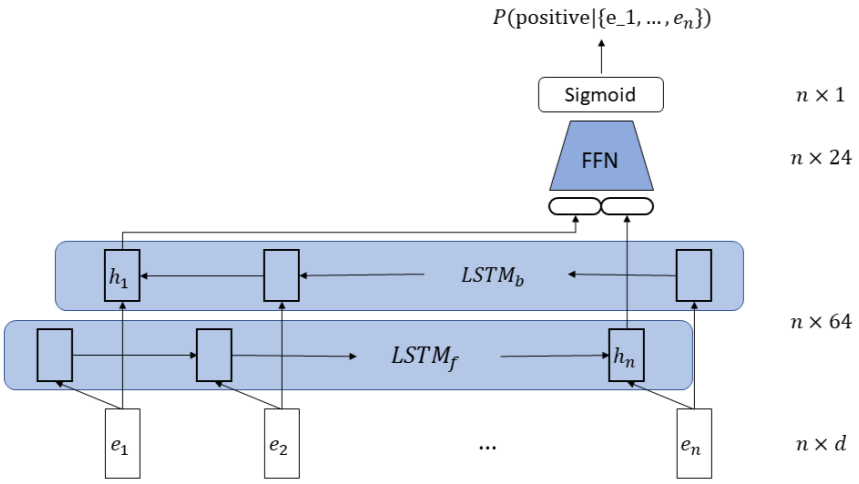


We used embeddings of length $d = 300$, 24 units for the hidden layer and 1 unit for the prediction.

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=['accuracy'])
model.summary()
```

4.2 LSTM

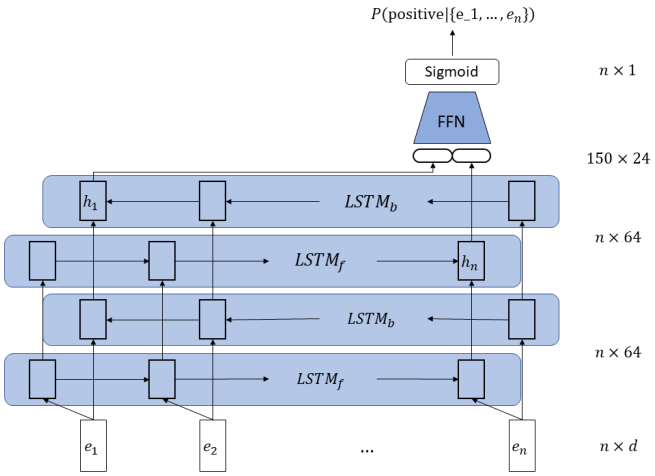
The second class of models are based on LSTM units. The input embeddings are passed to a bidirectional LSTM layer, which will process the input sequence one token at a time, updating its context and producing an output at each step. The LSTM layer is followed by a Fully Connected Layer with a ReLu activation and a single output layer with a sigmoid activation. This network will generate a prediction for each time step, but only the last prediction will be used for evaluation.



This model uses embeddings of size d which was chosen between 200 and 300, a bidirectional LSTM layer with 32 units, a fully connected layer with 24 units and 1 output unit for the prediction.

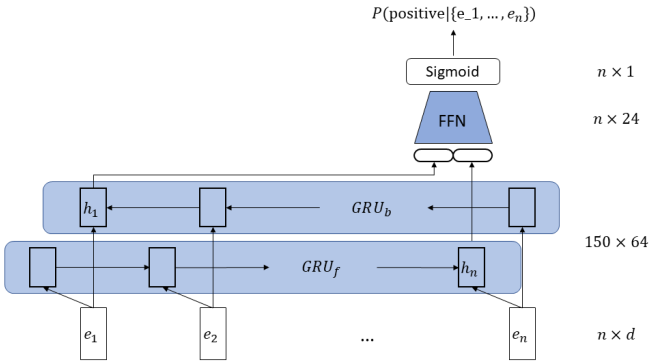
```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, return_sequences=True,
    input_shape=(max_length, embedding_dim),dropout=0.5)),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=['accuracy'])
model.summary()
```

A multilayer version, obtained by stacking more LSTM layers one over the other was also created: this model uses 2 layers of 32 LSTM units for each direction (64 in total), 24 fully connected units with a ReLu activation and a single unit for the prediction.



4.3 GRU

The GRU network shares most details with the LSTM network previously described, while using the GRU unit for the recurrent layer. The embeddings are passed to a bidirectional GRU layer, followed by a Fully Connected Layer with a ReLu activation and a single output layer with a sigmoid activation.



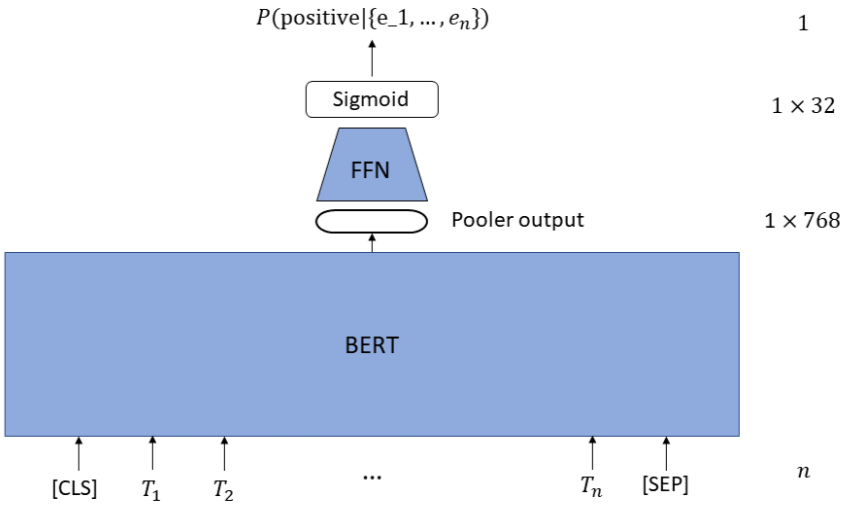

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.GRU(32, return_sequences=True,
    input_shape=(max_length, embedding_dim),dropout=0.5)),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy',optimizer=optimizer,metrics=['accuracy'])
model.summary()
```

4.4 Transformers

The last model we evaluated is based on Transformers and uses a pretrained version of BERT (*bert-base-uncased*), obtained from Huggingface and pretrained on BookCorpus and English Wikipedia. The architecture of the pretrained BERT model is described in depth in Section 6.

To be able to perform the encoding, BERT needs a sequence in the format [CLS] sequence [SEP]

To achieve that, a pretrained BertTokenizer from HuggingFace was used. The BertTokenizer takes as input a text sequence and produced in output an encoded sequence of tokens with the added special tokens and padded to match the defined max length of the sequence, and an attention mask which avoids to perform attention on padding token indices.



To perform the classification task, the last layer hidden state of the first token of the sequence is then passed to a linear layer with a $\tanh$ activation, which produces an output of the dimension of the hidden layer, in our case 768.

The output is then passed to a Feed Forward Network composed by a fully connected layer with 32 units and an output layer with one unit that finally performs the prediction.

5 Static Embedding: GloVe

Embeddings definition Vector semantics is the standard way to represent word meaning in NLP, helping us model many of the aspects of word meaning and solving several tasks that involve semantic interpretation of the words, such as *sentiment analysis*.

The idea was born in the 1950s and evolved until the point that to define the meaning of a word by its **distribution** in the language use (its neighboring words or grammatical environments).
In other words, the main idea was that two words that occur in very similar distributions (whose neighboring words are similar) have similar meanings.
So, vector semantics consists in representing a word as a **point** in a multidimensional semantic space that is derived from the distribution of word neighbors.

Embeddings are the vectors for representing words and often correspond to a **projection** from an higher to a lower dimensional space.
Embeddings are useful because in a Neural Networks architecture that analyzes texts, words are represented by their embeddings. By representing words in this way, we have that words with similar meanings have similar (or close) embeddings.
Moreover, often, embeddings are **short** and **dense** vectors with semantic properties.

Static Embeddings There are two kinds of embeddings: **static embeddings** and **contextual embeddings**.
Static embeddings means that the method learns one fixed embedding for each word in the vocabulary and it never changes after being learnt.
In our project we trained a static embedding layer over the vocabulary of each data set and used a pre trained static embedding called **GloVe** to carry out the experiments.

5.1 GloVe static embedding

Introduction GloVe stands for **Global Vectors for Word Representation** and is an unsupervised learning algorithm for obtaining vector representation for words proposed by *Stanford* in Pennington et al. (2014).

Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representation showcase interesting linear substructures of the word vector space.

Pre-trained word vectors In our project we made use of the GloVe pre-trained embeddings trained on 2 different corpus:

- Trained on *Wikipedia 2014* and *Gigaword 5* with 6B tokens, 400k vocabs and 300 dimension vectors.
- Trained on *Twitter* corpus composed by 2B tweets. It contains 27B tokens, 1.2M vocabs and 200 dimension vectors. This embedding layer was used to predict the sentiment in the **disaster tweets data set**.

Training The GloVe model is trained on the non-zero entries of a global word-word co-occurrence matrix, which tabulates how frequently words co-occur with one another in a given corpus.
Populating this matrix requires a single pass trough the entire corpus to collect that statistics.
For large corpora, this pass can be computationally expensive, but it is one-time up-front cost. Subsequent training iterations are much faster because the number of non-zero matrix entries is typically much smaller than the total number of word in the corpus.

Model Overview GloVe is essentially a log-bilinear model with a weighted least-squares objective. The main intuition underlying the model is the observation that ratios of word-word co-occurrence probabilities have the potential of encoding some form of meaning.
Using the matrix we can compute the ratio of probabilities between any two pairs of words as follows:

$$F(w_i, w_j, \tilde{w_k}) = \frac{P_{ik}}{P_{jk}} \tag{15}$$

For example. consider the co-occurrence matrix probabilities for **target** words *ice* and *steam* with various trial words from the vocabulary.

Table 1: Co - occurrence probabilities matrix

Probability and Ratio	k = solid	k = gas	k = water	k = fashion
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

As we might expect, *ice* co-occurs more frequently with *solid* than it does with *gas*, whereas *steam* co-occurs more often with *gas* than it does with *solid*. Only in the ratio of probabilities does noise from non-discriminative words like water and fashion cancel out, so that large values (much greater than 1) correlate well with properties specific to ice, and small values (much less than 1) correlate well with properties specific of steam. In this way, the ratio of probabilities encodes some crude form of meaning associated with the abstract concept of thermodynamic phase.

The training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. Owing to the fact that the logarithm of a ratio equals the difference of logarithms, this objective associates (the logarithm of) ratios of co-occurrence probabilities with vector differences in the word vector space. Because these ratios can encode some form of meaning, this information gets encoded as vector differences as well.

Acknowledgment You can find more details about static embeddings and GloVe in Speech and Language Processing online book and here respectively.

6 Contextual Embedding: BERT

In the previous Section we've talked about embeddings, more precisely about static embeddings, and we gave a description of GloVe, the static embedding that we used in our experiments.
In addition to static embeddings, we also used **contextual embeddings** for our experiments, so it's incumbent upon us to give a definition of contextual embeddings and a description of **BERT**, the model we used for our embedding layer.

Contextual Embeddings Unlike static embeddings, **contextual embeddings** are characterized by the fact that the semantic vector for each word is different in different contexts, so it's dynamic.
Contextual embeddings can be used as representations of word meaning in context for any task that might require a model of word meaning. More precisely, contextual embeddings represent the meaning of word *tokens*: instances of a particular word type in a particular context.
Summing up, contextual embeddings are thus vectors representing some aspect of the meaning of a token in context.
For example, given a sequence of input tokens $x_1 \dots x_n$, we can use the output vector y_i from the final layer of the model as a representation of the meaning of the token x_i in the context of sentence $x_1 \dots x_n$. Or, it is common to compute the representation for x_i by making a *pooling* of the output tokens y_i of the **last four** layers of the model.

6.1 BERT contextual embedding

BERT stands for **Bidirectional Encoder Representations from Transformer** and is a *transformer-based* deep learning technique used in NLP tasks for training contextual embeddings. It is first introduced by Google in Devlin et al. (2018) and exploits **bidirectional transformers** encoder and the method of **masked language modeling** to allow the language model to see the entire texts at a time, including both the right and the left context. This makes BERT different from the **causal** or left-to-right paradigm we've seen earlier. Indeed, bidirectional encoders overcome causal paradigm's limitation by allowing the **self-attention** mechanism to range over entire input. Therefore, they use self-attention to map sequences of input embeddings $(x_1 \dots x_n)$ to sequences of output embeddings the same length $(y_1 \dots y_n)$, where the output vectors have been **contextualized** using the information from the entire input sequence.
The contextualization is accomplished through the use of the same self-attention mechanism of the left-to-right models, with the difference that bidirectional encoders skip the mask, allowing the model to contextualize each token using information from the entire input.
All the other element of the transformer architecture remain the same for bidirectional encoder models.

The original BERT model The original bidirectional transformer encoder model BERT had the following characteristics:

- A subword vocabulary consisting of 30000 tokens generated using the *Word-Piece* algorithm,
- Hidden layers of size of 768,
- 12 layers of transformers blocks, with 12 multihead attention layers each.

It resulted in a model with more than 100M parameters.
Finally, a fundamental issue with transformers is that size of the input layer dictates the complexity of model. For BERT, a fixed input size of 512 subword tokens was used.

Training Bidirectional Encoders The original approach to training bidirectional encoders is called **Masked Language Modeling (MLM)** and it uses unannotated text from a large corpus. Here, the model is presented with a series of sentences from the training corpus where a random sample of tokens from each training sequence is selected for use in the learning task. Once chosen, a token is used in one of three ways:

- It is replaced with the unique vocabulary token [MASK].
- It is replaced with another token from the vocabulary, randomly sampled based on token unigram probabilities.
- It is left unchanged.

In BERT, 15% of the input tokens in a training sequence are sampled for learning. Of these, 80% are replaced with [MASK], 10% are replaced with randomly selected tokens, and the remaining 10% are left unchanged as we can observe in Figure 2.

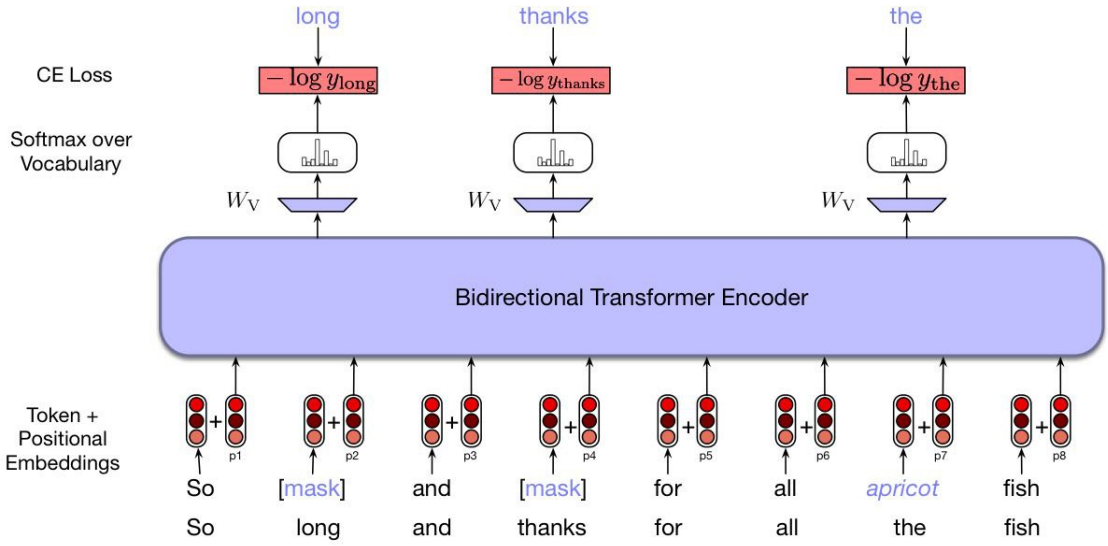


Figure 2: Masked language model training. Three of the input tokens are selected, two of which are masked and the third is replaced with an unrelated word.

The MLM training objective is to predict the original inputs for each of the masked tokens using a bidirectional encoder of the kind described in the last section.
The cross-entropy loss from these predictions drives the training process for all the parameters in the model. Note that all of the input tokens play a role in the self-attention process, but only the sampled tokens are used for learning.
More specifically, the original input sequence is first tokenized using a subword model. The sampled items which drive the learning process are chosen from among the set of tokenized inputs. Word embeddings for all of the tokens in the input are retrieved from the word embedding matrix and then combined with positional embeddings to form the input to the transformer.

Fine-Tuning The power of pretrained language models lies in their ability to extract generalizations from large amounts of text—generalizations that are useful for myriad downstream applications.
To make practical use of these generalizations, we need to create interfaces from these models to downstream applications through a process fine-tuning called **fine-tuning**. Fine-tuning facilitates the creation of applications on top of pretrained models through the addition of a small set of application-specific parameters.
The fine-tuning process consists of using labeled data from the application to train these additional application-specific parameters. Typically, this training will either freeze or make only minimal adjustments to the pretrained language model parameters.

HuggingFace pre-trained BERT For our experiments, we used the pre-trained BERT model provided by HuggingFace (Wolf et al. (2019)) available at the following link

Acknowledgement You can find more details about *contextual embeddings* and BERT in Speech and Language Processing online book and in Devlin et al. (2018) respectively, as well as many of the notions described ahead.
Moreover Figure 1 from Section 3 and Figure 2 from Section 6 are taken from Speech and Language Processing online book.

7 Experiments

Our goal was to solve the sentiment analysis task using NLP techniques. To do this, we modeled different neural networks with different architectures and embeddings with different characteristics.
We created models based on different units: FFN, LSTM AND GRU, using 3 different embedding paradigms: trained on the data set corpus, static pre-trained GloVe, and contextual pre-trained BERT.

We therefore conducted **two experiments** on 3 data sets belonging to different content domains to test the generated models and embeddings used.
The **first experiment** aimed to find the best performing model, while the **second experiment** aimed to answer the following questions:

- What is the best static embedding architecture for the LSTMs and GRUs neural networks models?

The following section is organized as follows: first we will give a high-level overview of the implemented code; after that we will deal with the text pre-processing part and then go on to describe the 2 experiments conducted in detail.

7.1 Code

The models were implemented using the **Python 3.7** programming language with **TensorFlow** and **Keras** as libraries, as well as **Pandas** and **Numpy**.

Experiments were conducted on a remote computational environment using Kaggle, which provides the following technical features:

- 12 hours of execution time for CPU and GPU notebook sessions.
- 20 Gigabytes of auto-saved disk space.
- **CPU** specifications:
  - 4 CPU cores
  - 16 Gigabytes of RAM
- **GPU** specifications: NVIDIA Tesla P100
  - 2 GPU cores
  - 13 Gigabytes of RAM

The detailed code implementation is available on **GitHub**at the following link

7.2 Data pre-processing

Each data set has been pre-processed before it was given as input to the neural networks.
By pre-processing we mean cleaning the data set from dirty characters and pre-process the text to generate the *corpus*.

Dirty characters removal Some data sets can contain samples with dirty characters such as:

- HTML tags
- Links
- Hashtags
- Emoticons [both in single character :smile: or with composed characters :)]
- Non-ascii characters

These characters have been removed prior to text pre-processing in order to improve the quality of the data set and obtain overall better results: if it was not done, the removal of punctuation would have resulted in the production of several nonsense words, caused by links, hashtags or tags. For example, in the **Disaster tweets data set** almost 100 samples contained some dirty characters mentioned above.

Text pre-processing Another fundamental practice before starting analyzing a text is to pre-process it. Pre-process a text means that it goes through a pipeline made of the following operations to build a *corpus*:

1. **Punctuation and numbers removal**
2. **Tokenization**.
3. **Case Folding** to lower case.
4. **Stop-Words removal**.
5. **Stemming**.

To tokenize the text we used the TensorFlow Tokenizer to create the **vocabulary** using the train data set, and then proceed to tokenize the train and test data set.
There are a few parameters to be set:

- Vocabulary size
- Dimension of the embedding: usually between 100 and 300
- Maximum length in characters
- Padding settings

We set 1000 as vocabulary size: larger vocabularies were tried but did not bring to a substantial increment in performance. We also chose a max length of characters of each sample, and samples longer than the max length have been truncated, while shorter have been padded to reach tensors of equal size for each batch.

The last operation of the text pre-processing pipeline was **stemming** that was done with *PorterStemmer* from the NLTK library.
However, it is important to specify that **stemming was not applied to input texts that were given to networks with pre-trained embedding layers such as GloVe or BERT** because these were trained on corpora with words that did not undergo the stemming process and this created a conflict.

7.3 Loss function

Each model was trained using the same loss function and optimizer:

- Loss function: Binary Cross Entropy.
- Optimizer: Adam which is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.
- The learning rate is constant and set to 10^{-4} except for BERT where the learning rates is set to 10^{-5}.

7.4 Experimental setting

All models were trained for 40 epochs with **early stopping**, setting 5 epochs of patience and using 20% of the training data for validation.
Each model was then evaluated on a separate held-out set, composed by 20% of the data of each data set, and we evaluated for each model using the classification_report tool from *scikit-learn* library:

- Accuracy
- Precision
- Recall
- F1-score

Specifically, to measure the performance of the models on the Trip Advisor data set, we used the F1-score because the data set is highly unbalanced, as can be seen in Figure 3.

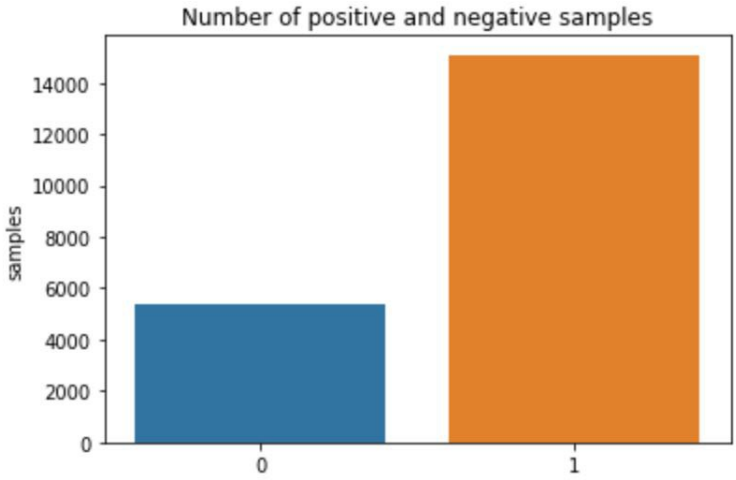


Figure 3: Number of positive and negative samples in the Trip Advisor Hotel Reviews data set that shows that the data set is unbalanced.

We evaluated the four classes of models on the Disaster Tweets data set (DT), News Headlines Sarcasm Detection(NH) and Trip Advisor Reviews (TR) and for each data set we set the *max_length* hyper parameter that we discussed in the **data pre-proccesing section** as follows:

Table 2: max_length for each data set

data set	max_length value
DT	150 words
NH	25 words
TR	500 words

7.5 Results

In this section we're giving a brief commentary of the results obtained from the two experiment. The results are reported in the Table 3, where in each rows there are the model architecture paired with different embedding paradigm, while in the columns we have the name of the data set which the models were tested on. In the last column we reported the F1-score too because the Trip Advisor data set is unbalanced.

	DT	NH	TR	
Model	Acc	Acc	Acc	F1
FFN	0.63	0.75	**0.89**	**0.86**
LSTM(w/o pt embs)	0.79	0.75	0.88	0.85
LSTM(w/o pt embs) - 2 layers	0.76		0.84	0.81
LSTM (GloVe Wiki+GW5B)	0.79	0.76	0.87	0.81
LSTM (GloVe Twitter Embs)	0.77	0.76	0.83	0.81
GRU (w/o pt embs)	0.79	0.75	0.88	0.84
GRU (GloVe Wiki+GW5B)	0.78	0.76	0.86	0.83
GRU (GloVe Twitter Embs)	0.78	0.76	0.88	0.86
BERT classifier	**0.93**	**0.83**		

Table 3: Comparison of models trained on the Disaster Tweets data set (DT), News Headlines Sarcasm Detection(NH) and TripAdvisor Reviews (TR)

7.5.1 Experiment 1 results: best model

The following is a brief commentary of the results obtained from the experiments shown in Table 3, broken down by data set.
In general it came out that BERT is the best model paradigm for the task and that it benefits largely from the removal of the dirty characters: it was observed that training without performing this operation led to a rapid overfitting of the model, while this behaviour was not present after the removal of the characters.

Disaster Tweets (DT) The best model in terms of accuracy for the Disaster Tweets data set is **BERT** with an accuracy of 0.93.
Its accuracy outclassed the other models' accuracy, especially for the FFN that reached only a 0.63 result.

News Headlines Sarcasm Detection (NH) Also for the News Headlines Sarcasm Detection the best model is **BERT** with an accuracy of 0.83.
Unlike the previous data set, the gap between BERT's accuracy and the other models' one is lower, with a minum of 0.75 from FFN.

Trip Advisor Hotel Reviews (TR) For this data set BERT was not included in the tests for computational reasons, since the hardware at our disposal did not allow us to perform the training process properly. We could have also reduced the learning rate but we would have definitely directed the model to overfit.

In terms of F1-score (since the data set is unbalanced) it turned out that **FFN** is the best among the other models in absolute terms, with 0.86 value. However, the difference in accuracy and F1-score between models is really low, so discriminating a better model is not easy.
But if we had to choose between all the models we would select the FFN model because it is simpler, easier to train in terms of computational resources and its learning curve is smooth and doesn't present any overfitting as we can see in Figure 4.

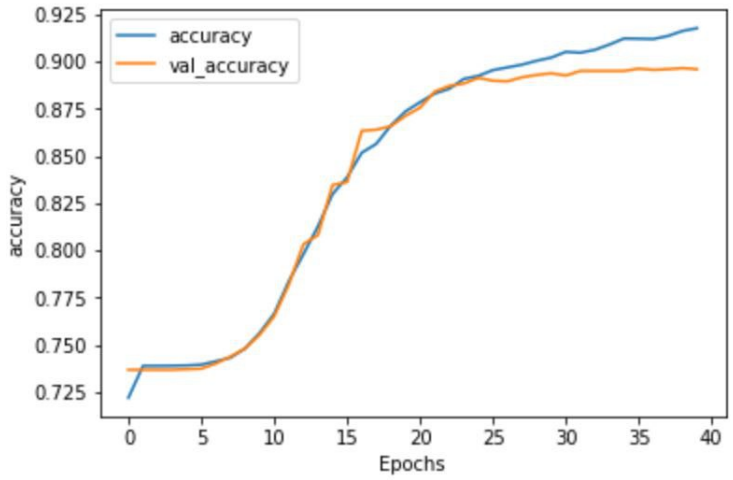


Figure 4: Training curve of the FFN model on TR data set by accuracy

General comments BERT is the best model but it tends to overfit as we can see in Figure 5. A possible solution to avoid this behaviour could be to decrease the learning rate that was set to 10^{-5} but it would enormously increase the computational cost for the training and with out hardware setting it couldn't be possible to conclude the training.

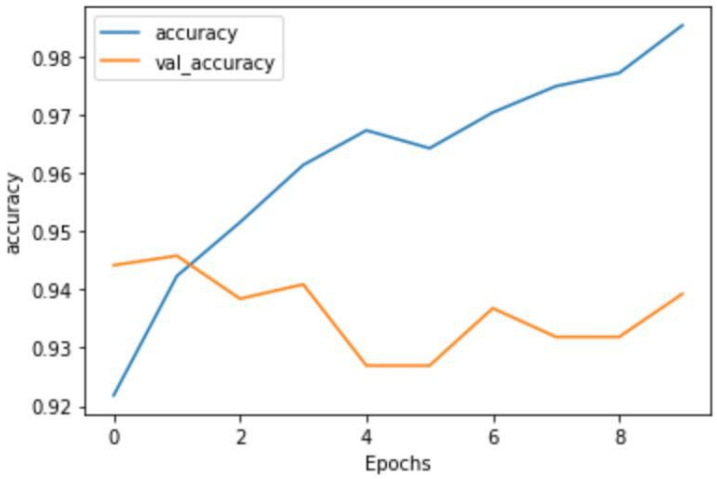


Figure 5: BERT training curve on the Disaster Tweets data set

Another interesting observation is that every test the LSTM models and the GRU ones behave very similarly.

7.5.2 Experiment 2 results: best static embedding paradigm

To select the best static embedding paradigm we tested 3 different settings paired with LSTMs and GRUs. It turned out that the best embedding layer is the one **without pre-training** and trained on the current *corpus* of the actual data set except for the News Headline Sarcasm data set, where the best models are the one pre-trained on the Wiki+GW5B or Twitter data set.

We expected that, at least for the Disaster Tweets data set, the embedding pre-trained on the twitter data set could make a difference but it even performed worse than the embedding pre-trained on the Wiki + GW5B data set.

8 Conclusions

In this paper we aimed to solve the nlp task the sentiment analysis by modeling different models based on neural networks. Specifically, we modeled several neural networks based on 3 different architectures: FFNs, LSTMs, and GRUs. In addition, we integrated the models with 3 different types of embedding: one trained on the data set corpus, one pre-trained statistic (GloVe), and finally one pre-trained contextual (BERT).

The models were implemented in Python language through the use of several libraries including TensorFlow and Keras, pandas, numpy and scikitlearn.

Our goal was to evaluate the best model among those created and to find the best static embedding architecture. To do so, we performed 2 different experiments on 3 different data sets belonging to different domains: Disaster Tweets data set, News Headline Sarcasm Detection data set and Trip Advisor Hotel Reviews data set.

The test results show that for the Disaster Tweets and News Headline data sets the best model is the one using BERT contextual embedding while for the Trip Advisor data set the best model is the simple FFN, although the other models do not differ much and BERT was not tested on this data set.

Instead, regarding the best static embedding, the best paradigm is the one that does not use a pre-trained embedding but instead creates one by training it on the corpus of the current data set, except for the News Headline Sarcasm data set, where the best models are the one pre-trained on the Wiki+GW5B or Twitter data set.

References

Rishabh Misra and Prahal Arora. Sarcasm detection using hybrid neural network. *arXiv preprint arXiv:1908.07414*, 2019.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2019. URL https://arxiv.org/abs/1910.03771.

Guy Hadash, Einat Kermany, Boaz Carmeli, Ofer Lavi, George Kour, and Alon Jacovi. Estimate and replace: A novel approach to integrating deep neural networks with existing applications. *arXiv preprint arXiv:1804.09028*, 2018.

Rishabh Misra and Jigyasa Grover. *Sculpting Data for ML: The first act of Machine Learning*. 01 2021. ISBN 978-0-578-83125-1.

Dan Jurafsky and James H Martin. Speech and language processing. vol. 3. *US: Prentice Hall*, 2014.