Aspect-based Sentiment Analysis at Evalita (ABSITA): comparing an attention enhanced LSTM and a fine-tuned BERT

Bersani Michele [*], *Lusiani Federico* [†]

Human Language Technologies, Academic Year: 2020/2021

5 July 2021

1 Abstract

In the field of Natural Language Processing, and more specifically Sentiment Analysis, transfer learning from large, pre-trained transformers models (such as BERT) have shown state of the art results. This work aims to apply these models to a dataset where understanding of both content and sentiment is required. Additionally, we compare these transfer learning models with LSTM-based models trained from scratch. Both classes of models have been augmented with an additional attention layer, following the work of Baziotis et al. [1]. The dataset used is the ABSITA dataset, containing about 6000 italian reviews extracted from *booking.com*.

2 Introduction

Performing Sentiment Analysis on reviews presents some difficulties due to high variance in the lengths of the sequences. Moreover, the sequences can include typing error or ironical expressions, which are known for increasing the difficulty of NLP tasks.

2.1 The task

The ABSITA task requires detecting the presence and the polarity of the following topics in a piece of review:

Cleanliness Comfort Amenities Staff Value Wi-Fi Location Other

[*]Università di Pisa, Master Degree in Computer Science, m.bersani@studenti.unipi.it

[†]Università di Pisa, Master Degree in Computer Science, f.lusiani@studenti.unipi.it

For every topic, there are three binary labels: one for the presence and two for the polarity of the information, positive or negative. Note that all the possible combination of the polarity are possible: a review can be neutral, positive, negative, but also both of them, in case it includes some positive comments, opposed to some more critical ones. The presence label works on Aspect Category Detection (ACD), whereas the polarity labels work on Aspect Category Polarity (ACP).

The top performing model on ABSITA challenge was ItaliaNLP 1, by Cimino et al. [3]. The model proposed in the paper for this task consists in a bi-directional LSTM. On top of it, the authors put a separate dense module for every binary label.

2.2 Text preprocessing

ABSITA is a clean dataset, containing only words and punctuation. The only preprocessing operation needed was lowercasing all the input strings, as required by the embedding layers we used.

2.3 The training dataset

The training dataset is made of 6337 labelled sentences of variable length. The length of the sentences is highly variable: most of them are below 10 words, but some others include more than 100 words. Regarding the semantics and the lexicon, most of the sentences include a simple structure with words related to travels and satisfaction level. The presence of irony is not so frequent as in other fields like tweets, but it is still present. Spelling errors are frequent and correctly represent a real-life application of NLP dealing with user text.

As observable in Table 1, the dataset is highly unbalanced, in fact all the boolean indicators are mostly negative. To perform model selection, a fixed 20% portion of the patterns is held out for validation. Both the Training Set and the Validation Set contain a frequencies similar to the full dataset (relative difference below 15%).

Topic	Presence	Positive	Negative
cleanliness	0.14	0.08	0.06
comfort	0.37	0.15	0.23
amenities	0.29	0.15	0.15
staff	0.19	0.15	0.05
value	0.07	0.03	0.04
wifi	0.02	0.01	0.01
location	0.21	0.19	0.03
other	0.12	0.05	0.07

Table 1: Frequencies of the classes in the dataset

3 Models

The models we compare are Italian versions of BERT and a Deep LSTM. For all the models, the binary cross entropy function is used as a loss function and Adam Stochastic Gradient Descent (Kingma and Ba [7]) is chosen as optimization method.

3.1 Deep LSTM

This model is inspired to the work by Baziotis et al. [1] which proposed a 2-layer bidirectional LSTM with attention pooling over the last layer and a fully-connected module on top of it. A summary of the model is observable in Figure 1.

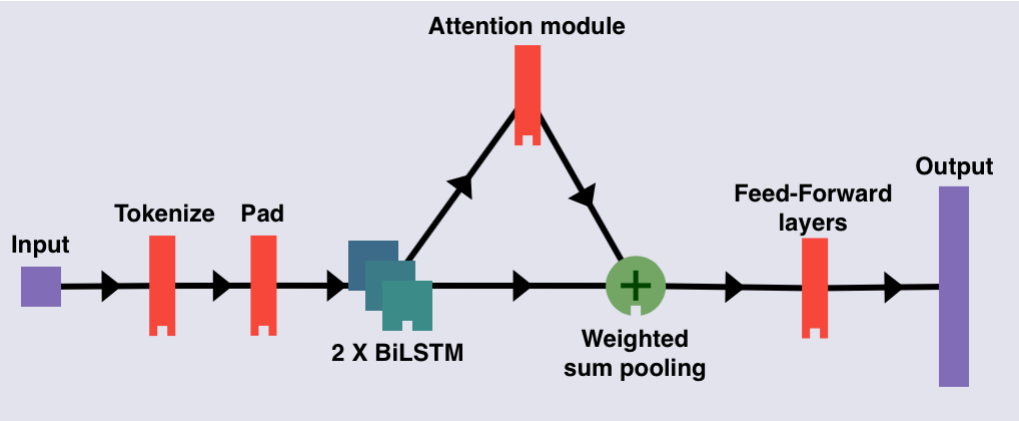

Figure 1: Sketch of the Deep LSTM architecture

3.1.1 Tokenization and Embedding

The tokenization is performed through the Gensim library, which provides maximal contiguous sequences of alphabetic characters (omitting digits)[1]. For the embedding, we use a pre-trained Word2Vec embedding made public by the work of Di Gennaro et al. [4]. This embedding has been trained on a large corpora of Italian texts (for a total of 2.6 GB of raw text). We expect these word-embeddings (vectors of 300 floats) to be semantically informative and to provide a good input for the model. However, it must be noted that Word2Vec is a lookup table for word-embedding, not accounting any form of context. Therefore, it is less informative compared to other alternatives, such as the ELMo embedding by Peters et al. [8].

3.1.2 Padding and Masking

At this stage, a sentence of T tokens is encoded in a $T \times 300$ matrix. To gain a significant speed-up, we ran the training of the models using Tensorflow functionality for GPU

[1] https://tedboy.github.io/nlps/generated/generated/gensim.utils.tokenize.html

acceleration. This functionality employs the GPU to parallelize the training on all the patterns of a batch. Because of this, it is necessary for each batch to be composed by sequences of the same length.
This introduces the need for padding and masking the sequences accordingly. Although a common solution is to simply pad all the sequences in the dataset to a fixed maximum length, we obtained a considerable speed-up by adding a custom "batch-trimming" layer to the input of the model. It trims the input of every batch, avoiding unnecessary padding in the patterns. Thus, while the dataset is effectively padded to maximum length, during training the model is only fed with the needed padding for each batch.

3.1.3 2 layers of BiLSTM

This module contains two identical bidirectional Long Short Term Memory. LSTM-1 reads the input sequence and provides an encoding with the same length to LSTM-2. If both the forward and the backward layer of LSTM-1 have c cells, every step of the encoded sequence contains $2c$ values. For the sake of simplicity, LSTM-2 has always the same size of LSTM-1, so also the final output sequence contains vectors of size $2c$.

3.1.4 Attention module for pooling the sequence

The feed-forward module on top of LSTM-2 requires a single input of fixed size. To pool the encoding of each token into a single vector of dimension $2c$, we replicated the Attention module of [1]. In particular, given the encodings $\mathbf{h_i} \in \mathbb{R}^{2c}$, the pooled output $\mathbf{r}$ is computed as follows:

$$e_i = \tanh(\mathbf{w_h}^\top \mathbf{h_i} + b) \tag{1}$$

$$a_i = \frac{\exp(e_i)}{\sum_{t=1}^{T} \exp(e_t)}, \ \sum_{i=1}^{T} a_i = 1 \tag{2}$$

$$\mathbf{r} = \sum_{i=1}^{T} a_i \mathbf{h_i} \tag{3}$$

Looking at the formulas, we can see that this attention module closely resembles the attention mechanism often used in encoder-decoder LSTM models for sequence transduction, with the difference that only the encoder outputs $\mathbf{h_i}$ are present, while the decoder state vector (also known as context) is here replaced by a fixed (and trainable) vector $\mathbf{w_h}$.

3.1.5 Feed-forward classifier

After pooling the sequence, the feed-forward classifier produces the output of 24 binary indicators. In principle, a single linear layer might be sufficient for such a task; but during model selection we also explore deeper modules with 1 or 2 hidden layers.

3.1.6 Weight initialization

The weights of dense layers and the input kernel of the LSTM layers are initialized following the standard by Glorot and Bengio [5]. Moreover, we initialize the LSTM recurrent layers to be orthogonal and their forget gates to be centered around 1, as suggested by Jozefowicz et al. [6].

3.1.7 Dropout and regularization

To prevent overfitting (see Srivastava et al. [9]), we introduce various dropout stages:

- a vertical dropout of 0.2 at the input of both LSTM layers,
- a dropout of 0.3 before the attention module and after every feed-forward hidden layer.

We explored an L_2 penalty term on the weights, but all the tested values spoiled the performance on the validation set. Hence, dropout is the only adopted regularization technique.

3.1.8 Model selection

For the Deep LSTM, the model selection is carried out by evaluating the performance on the Validation Set. The tested models differ by number of cells in the LSTM layers and by dimension of the terminal feed-forward model.

LSTM cells	**Feed-forward layers**	**AUC**	**F1-score**
2 X 128	(24)	0.9588	0.711
2 X 256	(24)	0.9611	0.739
2 X 128	(100,24)	0.9616	0.718
2 X 256	(100,24)	0.9597	**0.748**
2 X 128	(300,100,24)	0.9607	0.737
2 X 256	(300,100,24)	0.9581	0.744

Table 2: LSTM results on validation data

The training is performed for 40 epochs over mini-batches of 64 patterns and a learning rate of 10^{-4} proves to be the best trade-off between performance and training time.

Dropout values of 0.4 were explored but proved to be too drastic, so we reduced it to 0.2 on the LSTM layers and 0.3 on the rest of the model.

3.2 BERT-based models

Pre-trained BERT models are transformers that were trained on large text corpora, learning to map sequences of tokens to an informative vector embedding. More specifically, given an input sequence, a BERT model outputs a vectorial encoding for each input

token (output sequence), plus a single "context" encoding, that was been pre-trained to bring information about the sequence as a whole.

To perform classification of the input sequence, many approaches include training a classifier (such as a feed-forward NN) on top of the context encoding. However, aiming to a better context encoding, we applied the attention mechanism of Section 3.1.4 for pooling the BERT output sequence. Several pooling approaches are compared in section 3.2.3.

The following paragraphs describe the architecture of the BERT-based models which we tested.

3.2.1 Data-preprocessing

The tokenization and embedding are part of the pre-trained BERT model, as using a different tokenizer and (especially) a different embedding would disrupt the correct functioning of the pre-trained model.
As for padding and masking, we repeat the same process described in 3.1.2.

3.2.2 Pre-trained BERT

We have tested two different pre-trained BERT models: the Italian BERT developed by the MDZ Digital Library team[2] and the GilBERTo model[3]. The two models were trained on large corpora of Italian texts, amounting to 13 GB (Italian BERT) and 71 GB (GilBERTo).
Both models encode every input token into a vector of dimension 768. Since a context vector is added to every pattern, a sentence of m tokens is returned as a sequence of $m + 1$ vectors.
Although the GilBERTo model is bigger and about 60% slower to train, it showed better performances compared to Italian BERT in early experiments. For this reason we decided to keep experimenting using only GilBERTo.

3.2.3 Attention module and Feed-forward classifier

This constitutes the trainable part of the BERT-based modules. For this part, we tried different configurations:

1. **Context encoding only.** A feed-forward classifier is applied to the context encoding.

2. **Sequence encoding only.** A feed-forward classifier is applied to the output of the attention layer encoding.

[2]https://github.com/dbmdz/berts
[3]https://github.com/idb-ita/GilBERTo

3. **Sequence encoding using context.** A single feed-forward classifier is applied to the output of a self-attention layer, using the context encoding as a fixed query and the tokens encodings as values and keys.

4. **Sequence encoding and context encoding.** The output of the attention layer and the context encoding are concatenated into a single vector, and then passed to the feed-forward classifier.

5. **Multi-headed model.** The tokens encodings are fed to 8 heads (one for topic), each outputting three class labels (topic presence, positiveness and negativeness). Each head is composed by an attention layer and a feed-forward classifier.

As feed-forward classifier, we used a feed-forward network with 1 hidden layer of 200 units, showing better performances than other alternatives (such as 0 or 2 hidden layers).

Adam optimizer is adopted, with a learning rate of 10^{-3}. The results are shown in Table 3.

BERT-based model type	**Final validation F1 Score**
1. Sequence encoding	0.723
2. Context encoding	**0.798**
3. Seq. + Cont. encoding	0.775
4. Seq. encoding using Cont.	0.782
5. Multi-headed model	0.792

Table 3: F1 Score for different BERT-models architectures

As observable, using the output of the attention layer (Sequence encoding only) leads to better results compared to the Context encoding only. Adding the context encoding to the sequence encoding, or including it in the attention mechanism (models 3 and 4) does not improve the performance, suggesting that the context encoding does not provide additional information to the sequence encoding. Also the multi-headed solution (model 5) does not bring improvements, confirming model 2 as the winner for both performance and simplicity.

3.2.4 Final BERT-based model

The architecture is illustrated in Figure 2. The pre-trained BERT model is GilBERTo. The tokens encodings outputted by the BERT module are attention-pooled into a single sequence encoding, to which a feed-forward network is applied. The feed-forward network is composed by input, hidden and output layers of size 768, 200 and 24 respectively. A dropout rate of 0.2 is applied to both input and hidden layer. The training is performed for 35 epochs over mini-batches of 64 patterns, using Adam optimzer, with learning rate set to 10^{-3} and learning rate decay set to 10^{-6}.

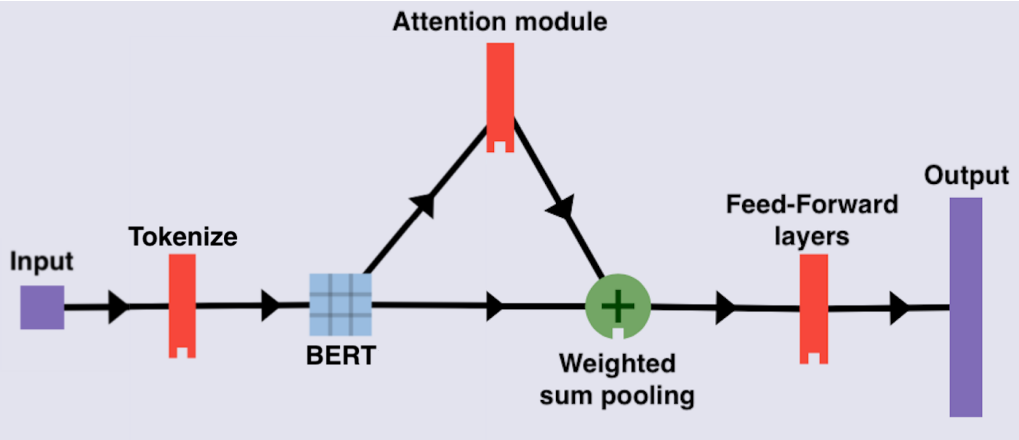

Figure 2: Sketch of the final BERT architecture

4 Testing

The ABSITA challenge provides a test dataset to make a final evaluation of the chosen models. Results of the final models for the ACD and ACP tasks are shown respectively in Table 4 and 5. For benchmarking, we also included the scores of the first and second ranked submissions to the challenge[4]. The challenge submissions are ranked based on the micro-averaged F1 score.

As we can see, the final BERT-based model outperforms both the final BiLSTM and the first two ranked submission to the ABSITA challenge.

Model	Precision	Recall	**F1-score**
Cimino et al. [3]	0.8397	0.7837	0.8108
Bennici et al. [2]	0.8713	0.7504	0.8063
Final BERT-based model	0.8551	0.7768	**0.8141**
Final BiLSTM model	0.8253	0.7370	0.7787

Table 4: ACD Task. Micro-averaged test scores.

Model	Precision	Recall	**F1-score**
Cimino et al. [3]	0.8264	0.7161	0.7673
Bennici et al. [2]	0.8612	0.6562	0.7449
Final BERT-based model	0.8377	0.7241	**0.7768**
Final BiLSTM model	0.7908	0.6610	0.7201

Table 5: ACP Task. Micro-averaged test scores.

[4]http://sag.art.uniroma2.it/absita/data/

5 Conclusions

While LSTMs still represent the state of the art for many NLP tasks, recent works have shown that the transformers perform better in many applications.
Our results show that by leveraging transfer learning from transformers pre-trained on large corpora text, we can get better results than older approaches such as bi-directional LSTMs. Nevertheless, using an attention enhanced BiLSTM trained from scratch, we were able to achieve a good performance on the task at hand.

We also note how our custom padding trimming layer introduced a significant speedup on training times ($\sim$3X), showing the importance of avoiding unnecessary padding in the batches. Moreover, from an efficiency perspective, the fine-tuning of a pre-trained BERT is faster than the training of a full BiLSTM: the training time is almost the half.

In the end, by combining GilBERTo, a powerful pre-trained model, and an attention-based pooling, we were able to outperform the winning model at the 2018 ABSITA challenge.

References

[1] Christos Baziotis, Nikos Pelekis, and Christos Doulkeridis. DataStories at SemEval-2017 task 4: Deep LSTM with attention for message-level and topic-based sentiment analysis. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 747–754, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[2] Mauro Bennici and Xileny Seijas Portocarrero. Ensemble of lstms for evalita 2018 aspect-based sentiment analysis task (absita). 12 2018.

[3] Andrea Cimino, Lorenzo De Mattei, and Felice Dell'Orletta. Multi-task learning in deep neural networks at evalita 2018. 2018.

[4] Giovanni Di Gennaro, Amedeo Buonanno, Antonio Di Girolamo, Armando Ospedale, Francesco A. N. Palmieri, and Gianfranco Fedele. An analysis of word2vec for the italian language. *Smart Innovation, Systems and Technologies*, page 137–146, Jul 2020.

[5] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, 2010.

[6] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. ICML'15, page 2342–2350. JMLR.org, 2015.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[8] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.