Final NLU project

Jaime Corsetti (229300)

University of Trento

jaime.corsetti@studenti.unitn.it

1. Introduction

In this work I explore the paper "Open-Domain Targeted Sentiment Analysis via Span-Based Extraction and Classification" [5], SpanABSA for short, by discussing the design choices and implementing the original architecture and some alternatives as well. I also point out some possible flaws in the design and in methodology, and I conclude by reproducing the author's results and by providing performances with the different architectures I tried.

In order to obtain these results, I leveraged on the original code [3] which has been provided by the authors, and I reimplemented most of it. During this operation I found some issues which I discuss in sections Methodological Issues and Structural Issues. Despite different problems in reproducing the results, I could reach similar performances to the one reported in the paper. I also propose some different architectural solution to solve some of the issues I identified. Code is publicy available at `https://github.com/jcorsetti/NLU-SpanSA`

2. Task Formalisation

In Natural Language Processing, Sentiment Analysis is the subfield which deals with understanding characteristics of a written text which are subjective to a person, usually to the one who has written the text. In particular, SpanABSA aims to solve the task of Aspect Extraction and Classification, in which an aspect is defined as an entity in the sentence (possibly made of more than one word) on which the writer express a judgement. Thus, the task can be divided in two subtasks:

- **Aspect extraction**: in which a set of aspects are extracted from a sentence
- **Aspect classification**: in which the extracted aspects are assigned a class from a fixed, closed set. In the particular case of SpanABSA, this set is (Negative, Positive, Neutral)

Depending on the approach, the process of solving one of the tasks can be more or less dependant on the another. In the specific case of SpanABSA, the two tasks are solved in a completely isolated way.

3. Data Description and Analysis

The original work is evaluated on three different datasets of the SemEval14 benchmark [7], namely *Laptop*, *Restaurants* and *Twitter*. Each is composed by a collection of plain text files, containing respectively reviews of computers, reviews of restaurants and tweets. For each sentence in each split, a subdivision in tokens is already provided, and to each token a label is assigned following a single tagging scheme: T-NEU, T-NEG and T-POS for a token of an aspect which is respectively Neutral, Negative or Positive, and O for an unassigned token. Notice that this notation does not support adjacent aspects of the same types, and in fact this case does not appear in the dataset. In this report, for simplicity I only made experiments on the *Laptop* partition of this dataset. An overview of the statistics of the this dataset can be examined in Table 1.

Laptop Partition	Train	Test
number of sent.	3045	800
of which empty	1587	389
avg length in tokens	16.6	14.6
number of aspects	2302	634
number of positive a.	987	339
number of neutral a.	455	165
number of negative a.	860	130
avg aspects per nonempty sentence	1.6	1.5

Table 1: *Statistics extracted from the LAPTOP partition of SemEval2014*

4. Model

In order to solve the problem, SpanABSA relies heavily on a pretrained BERT model [2] as an embedding network. Upon the powerful representation provided by this model, the original architecture adds two simple branches composed by fully connected layers: one for aspect extraction and one for aspect classification. Two different approaches are considered with respect two the interaction between the two tasks, which the authors call respectively **pipeline** and **joint**:

- **Pipeline**: two models are separately trained, one for the aspect extraction and one for the aspect classification task. At inference time, the two models are put together and work in a pipeline fashion.
- **Joint**: a single model is trained, in which the two task are optimized jointly. At inference time, the behaviour is the same as in in the pipeline setting.

The main difference is that in the Joint approach a single embedding network is trained on both tasks, and thus only a single training is necessary. However, this also produces performances which are slightly worse with respect to the pipeline approach. Due to time and computational limits, in this work I only explore the Joint setting. In the following sections, details about the branches are provided.

4.1. Aspect extraction

Given that an aspect is possibly composed by more than one token, the authors employ a span approach: instead of segmenting the sentence to find the entities, two fully connected layers output the likelihood of presence of a start/end of span at each valid token index. Note that in this way, the starts and ends are not matched: this is done subsequently by a specific heuristic function. Given a sequence of word embeddings **h**, the start logits

$\mathbf{g}^s$ are calculated as follows, as well as the probabilities distributions $\mathbf{p}^s$. The same applies for end logits and distributions $\mathbf{g}^e$ and $\mathbf{p}^e$.

$$\mathbf{g}^s = \mathbf{w}_s\mathbf{h}$$
$$\mathbf{p}^s = softmax(\mathbf{g}^s)$$

In the formula, $\mathbf{w}_e$ and $\mathbf{w}_s$ are two learned linear layers. The aspect extraction branch is optimized by a modified binary cross entropy loss, which weights more the sentences in which more aspects are present. This is not actually specified by the authors, who instead state that the branch is optimized by a classic cross entropy loss. Given the span ground truth $\mathbf{y}^s$ and $\mathbf{y}^e$, which are binary vectors with 1 at the start/end of a span and 0 otherwise, and the number of spans N, the loss is defined as follows:

$$Loss_{span} = -\frac{1}{2N}(\sum_{i=1}^{L}\mathbf{y}_i^e log(\mathbf{p}_i^e) + \sum_{i=1}^{L}\mathbf{y}_i^s log(\mathbf{p}_i^s))$$

Note that the N factor is significant, and in particular it is the source of the error in the case where the given sentence has no spans. When this happens, $N = 0$ and thus a division error arises in the original code. During evaluation, the extracted start and end indexes are coupled using an heuristic algorithm which is based on the confidence of each index.

4.2. Aspect classification

Following previous works [4], aspect classification is carried out by computing an average vector representative of the aspect through an attention mechanism. Two further fully connected layers produce logits which are then optimized by a classic cross entropy loss. Given a span of bounds $(\mathbf{s}, \mathbf{e})$, a representative vector $\mathbf{v}$ is computed using a learned attention layer $\mathbf{w}_a$:

$$\boldsymbol{\alpha} = softmax(\mathbf{w}_a\mathbf{h}_{s:e})$$
$$\mathbf{v} = \sum_{i=s}^{e}\boldsymbol{\alpha}_t\mathbf{h}_t$$

This vector is then used to perform the classification itself, by using two additional learnable parameter matrices $\mathbf{W}_p$ and $\mathbf{W}_v$ to obtain the logits and the final probability distribution:

$$\mathbf{g}^p = \mathbf{W}_p tanh(\mathbf{W}_v\mathbf{v})$$
$$\mathbf{p}^p = softmax(\mathbf{g}_p)$$

Given the one-hot encoding ground truth label $\mathbf{y}_p$, the loss function used is a standard cross-entropy:

$$Loss_{class} = -\sum_{i=1}^{C}\mathbf{y}_i^p log(\mathbf{p}_i^p)$$

The final loss is then computed as follows. Unless stated, in all the experiments the balancing factors α and β are set to 1.

$$Loss = \alpha Loss_{span} + \beta Loss_{class}$$

4.3. About code reproducibility

Despite the original code being available, the complete setting of the original paper is unknown beacuse of some missing information:

- A pretrained BERT-Large model is used as backbone, but it is not provided by the authors. I resolved this issue by downloading this model [1] from the popular HuggingFace library. Still, the used model is probably a different version beacuse of the time passed between this work and the original paper publication.
- The logits threshold hyperparameter (λ) used in the heuristic span extraction function is not provided for this particular dataset. The authors specify it as $\lambda = 8$ for a different dataset, and by a limited hyperparameter search I found that for the Laptop dataset $\lambda = 7$ yields good results.

Despite the mentioned issues, I could reproduce within a small error interval the results declared by the original work, as reported in Table 2.

5. Methodological Issues

5.1. Dataset preprocessing

The dataset used presents many sentences with no aspects, both in train and in testing splits, which are discarded during preprocessing. While this can be reasonable for the training split, discarding empty sentences in the test set means structurally reducing the probability of achieving false positives. Moreover, apparently UNIFIED [6], the main method used for confrontation does not exclude empty sentences from the test split, and clearly this makes the comparison unfair. Unsuprisingly, avoiding filtering for empty sentences in the test set causes a drop in the Precision of 3 points.

I also believe that excluding empty sentences from the training set is not a good practice: a method designed to handle a real world scenario should be able also to recognize (and thus, be trained with) sentences in which no opinion is present.

5.2. Evalution modality

The authors do not use a validation split, and instead rely only on train and test splits, using the latter as validation during the training. While this is generally regarded as bad practice, I believe it to be justifiable in this case, because of the small dimension of the dataset itself. Another problem arises with the checkpoint. The one for which the authors are reporting the final metrics is chosen during the training based on the F1 score obtained on the test set. In fact, the checkpoint achieving the best F1 score on the test set during training is saved and used as final checkpoint to report the metric. This practice of early stopping done on the test set is problematic, especially in case of model instability. In my implementation, only the final checkpoint is used instead.

5.3. Ambiguous backbone

In the paper, the authors clearly state that the use a pretrained BERT-Large model as embedding network for the architecture. However, in the GitHub code [3] the provided pretrained model is a BERT-Base, which is much smaller in terms of parameters and thus in expressive power. In fact, with BERT-Base I could not reproduce the reported results.

6. Structural Issues

6.1. Number of classes

The used dataset presents 3 different polarity classes: Positive, Negative and Neutral. However, in the code provided by the

authors the final classification layer has 5 channels, meaning that the network is trained to classify in a set of 5 classes. As noted in the reported results, changing back to 3 the number of classes slightly improves the performance. The reason for this choice in the code remains unclear to me. In all my experiments, unless specified the number of classes is set to 3.

6.2. Span cross-entropy loss

As mentioned in the previous Sections, the modified cross entropy used in the original implementation of the paper presents some subtle behaviour. Recall from previous Section that the loss used for the span extraction task is defined as follows.

$$Loss_{span} = -\frac{1}{2N}(\sum_{i=1}^{L} \mathbf{y}_i^e log(\mathbf{p}_i^e) + \sum_{i=1}^{L} \mathbf{y}_i^s log(\mathbf{p}_i^s))$$

where the span end distribution $\mathbf{p}^e$ is obtained by softmaxing the original logits $\mathbf{g}^e$, and likewise for the span start distribution. Notice that the multiplication by the binary labels amounts to excluding the contribution of the indexes in which there are no starts or ends of span. Moreover, the softmax function is applied along the sequence, meaning that the probability of the span indexes compete with each other. I believe instead that it would be more proper to treat each index as a distribution, meaning that the probability of an index of being a start or end of a span should be independent by the others. In order to do this, I devised a slightly different version of the span extraction head, which I describe in the Double span head Section

7. Alternatives architectures

In the following sections I describe two different architectural choices I tried in order to improve the pipeline performance.

7.1. Double span head

From the premises in the previous sections, I chose to try a different implementation of the span extraction head in which the linear layers which output the index logits are *doubled*, meaning that the model has to learn both the negative and the positive event logits. The original implementation computes softmax across the *positions* of the indexes, effectively producing a probability distribution across the sequence. The proposed one instead, which I call **double span head**, computes a probability distribution in *each index position*. Moreover, the loss contributions of the indexes with no spans bound are not ignored. The final results is very similar to a binary cross entropy loss, with the exception of the N_0 and N_1 values, which are respectively the number of negative indexes and the number of positive indexes.

$$Loss_s = -\frac{1}{N_0}(\sum_{i=1}^{L}(1-\mathbf{y}_i^e)log(1-\mathbf{p}_i^e) - \frac{1}{N_1}\sum_{i=1}^{L}\mathbf{y}_i^s log(\mathbf{p}_i^s))$$

The same formula is applied to the span end loss, and the final loss is computed as the mean of the two.

7.2. Context classification head

In order to improve the performance of the classification, I designed a simple variant of the default classification head which also takes into account a representation derived from the whole sentence. The intuition behind this choice is that giving more information about the context in which the spans exists should help in the classification process. Notice that, because of the way in which the BERT architecture is designed, the token embeddings already include this type of information. Still, I believe that this simple variant to be worth trying. The approach is very similar to the standard span head, but it is applied to the entire sentence to produce a single feature vector. Let $\mathbf{h}$ be the complete sentence embedding sequence, and $\mathbf{w}_s$ and $\mathbf{W}_g$ a learnable vector and matrix respectively. Then this single feature vector is computed as:

$$\boldsymbol{\beta} = softmax(\mathbf{w}_s\mathbf{h})$$
$$\mathbf{v}_g = \mathbf{W}_g \sum_{i=1}^{L} \boldsymbol{\beta}_i \mathbf{h}_i$$

The context feature vector $\mathbf{v}_g$ is then concatenated to every span feature vector $\mathbf{v}$, and the logits and final class probabilities are computed as in the default classification head:

$$\mathbf{g}^p = \mathbf{W}_p tanh(\mathbf{W}_v concat(\mathbf{v}, \mathbf{v}_g))$$
$$\mathbf{p}^p = softmax(\mathbf{g}_p)$$

Note that the term $\mathbf{w}_s$ acts like an attention mechanism, weighting the contribution of each term in the sentence. The matrix $\mathbf{W}_g$ provides an additional learnable transformation. The loss used for optimized remains the same.

8. Evaluation

8.1. Training modality

During the early stage of development, I noticed that the model performance appears to be quite noisy. Because of this, all the metrics I reported are a result of the average performance values of 5 identical runs, in which the seed is set randomly. I also added the standard deviation for each metric to better have a grasp of the model instability on each setting.

In order to make a fair comparison with the original work, I used the same hyperparameters, optimizer and general setting as it is provided by the authors. In particular, all models are training for 3 epochs on the training split, using a BERTAdam optimizer with a learning rate of $2e^{-5}$, and a warmup for the first 10 percent of the total steps. Notice that the authors declare to have used Adam in the paper, but the code presents BERTAdam instead. The batch size is set to 32 with a dropout probability of 0.1 in the classification head. Regarding the heuristic span-selecting algorithm, the max number of candidate spans M is set to 20, while the max numbers of proposed spans N is 10. Unless stated in some specific experiments, the logits threshold λ is set to 7, and the loss balancing factors α and β are set to 1. Also, to better compare with the original work, unless specified the empty sentences as filtered both in training and in testing splits.

- **Reported**: values reported from the authors in the original work [5]
- **Repr.**: baseline reproduced in my code
- **Repr. 5 c**: uses 5 classes in the classification head, as in the original implementation by the authors
- **No filter**: same training procedure as in **Reproduced**, but evaluation is carried on the dataset without filtering empty sentences.
- **Context**: trained using the context head for classification described in Context classification head

- **D. span**: trained using the double span head described in Double span head. To account for the different logits dimension and loss values, the logits threshold has been set to $\lambda = 3.5$, and the span loss balancing factor has been set to $\alpha = 3$

8.2. Evaluation metrics

Following the original paper, I report the general metrics as well as the task specific ones:

- **General metrics**: Precision, Recall and F1 scores follow the usual definition. Note that a perfect match is obtained only when both the span *and* the classification labels are correct. Thus, partial spans count as a miss. These metrics are reported in Table 2.
- **Span metrics**: These metrics evaluate only the performance of the span extraction, meaning that the classification label is ignored. While in the original work only the F1 score is reported, in Table 3 I report Precision, Recall and F1 scores for completeness.
- **Classification metrics**: Only accuracy is reported to compare the classification performance (Table 4). Note that these results are obtained by using the ground truth spans indexes to extract the aspect features vector, which are then classified by the trained classification head. Accuracy is computed as the sum of correct matches for each class, divided by the total number of aspects.

8.3. Result analysis and comparison

With some approximation, I believe that the baseline experiments results are good enough to state that the original paper results where reproduced. The mismatch in Precision and Recall general metrics may very well be due to the undeclared threshold logits hyperparameter. In different dataset, the authors declared this value to be $\lambda = 8$, which is higher than my baseline setting with $\lambda = 7$. This could explain the higher Recall but lower Precision with respect to the reported values: with a lower λ, more uncertain spans are accepted. This would lead to a higher number of false positive (lower Precision) but also to an higher number of successful detection (higher Recall)

As expected, when using a classification head with 3 classes the results are consistently better than with 5. Also, in the experiment without the test set filtering the Precision is worse than in the baseline setting. This confirms the hypothesis that excluding these sentences leads to a better Precision score due to the lower probability of false positives.

The experiment using the context head does not seem successful, as it achieves way lower general metrics, especially in Precision, although Recall seem to be slightly better. However, the Accuracy metric does not reach the baseline, and the higher standard deviation suggests that the model is less stable with this addition. I believe that this result is a sign of the effectiveness of the feature embedding learnt by the BERT backbone: those features already encode a representation of the context due to the transformer multi attention mechanism, and therefore using an additional, similar architecture does not improve performance. On the contrary, it probably makes the training harder, thus leading to worse results.

The experiment using the double span head and the modified loss also performed quite lower than the baseline. I believe that this could be traced back to different possible reasons. Firstly, my solution requires more weights to be trained, which may require a longer training time to reach the baseline performance. Secondly, the loss type also takes into account the contribution of the negative indexes, which may be less important than I expected. However, I believe that the most important source of error comes from my idea of not applying softmax across the span index sequence.

My reason for changing this was the fact that, in my opinion, the heuristic algorithm for span selection works well enough to solve the problem of span competition, and thus the use of the softmax is unnecessary and maybe harmful. However, from the reported results this hypothesis does not seem to be correct.

General	Precision	Recall	F1 score
Reported	67.41	61.99	64.59
Repr.	64.93 ($\mp$ 2.3)	64.07 ($\mp$ 0.7)	64.47 ($\mp$ 1.2)
Repr. 5 c	61.92 ($\mp$ 3.2)	64.51 ($\mp$ 2.9)	63.06 ($\mp$ 1.2)
No filter	61.62 ($\mp$ 0.9)	61.99 ($\mp$ 5.4)	61.64 ($\mp$ 2.6)
Context	57.52 ($\mp$ 3.6)	65.52 ($\mp$ 0.5)	61.20 ($\mp$ 1.9)
D. span	57.21 ($\mp$ 1.9)	61.77 ($\mp$ 2.7)	59.39 ($\mp$ 2.2)

Table 2: *General metrics reproduced on BERT-Large.*

Span	Precision	Recall	F1 score
Reported	-	-	83.35
Repr.	80.49 ($\mp$ 1.8)	79.46 ($\mp$ 1.5)	79.95 ($\mp$ 0.6)
Repr. 5 c	76.80 ($\mp$ 4.0)	80.00 ($\mp$ 3.4)	78.21 ($\mp$ 1.2)
No filter	76.04 ($\mp$ 0.2)	76.59 ($\mp$ 7.5)	76.12 ($\mp$ 4.0)
Context	73.26 ($\mp$ 3.9)	83.50 ($\mp$ 1.4)	77.97 ($\mp$ 1.8)
D. span	71.52 ($\mp$ 1.7)	77.19 ($\mp$ 2.4)	74.23 ($\mp$ 1.8)

Table 3: *Span metrics reproduced on BERT-Large.*

9. Conclusion

In conclusion, I believe the obtained results to be acceptable, although the complete setting of the original experiment where not replicable. I have explored some different solutions to the problems I have pointed out, and I got a better understanding of how the method works. I believe that more study could be carried out about the heuristic algorithm for span extraction, or in order to find a better way of combining together the span and end index. Note that during the learning process the start and end span indexes are optimized separately, so that there is no interaction between them. A different architecture could be designed in order to provide a more complete representation of what a span is, instead of regressing indipendently the start and end indexes.

Class	Accuracy
Reported	81.39
Repr.	79.02 ($\mp$ 1.2)
Repr. 5 c	78.90 ($\mp$ 0.2)
No filter	79.81 ($\mp$ 0.8)
Context	77.73 ($\mp$ 1.8)
D. span	78.45 ($\mp$ 0.9)

Table 4: *Class metrics reproduced on BERT-Large.*

References

[1] *BERT-Large model used in training*. https://huggingface.co/bert-large-uncased. Accessed: 19/07/2022.

[2] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[3] *Github SpanABSA repository*. https://github.com/huminghao16/SpanABSA. Accessed: 19/07/2022.

[4] Luheng He et al. "Jointly Predicting Predicates and Arguments in Neural Semantic Role Labeling". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. July 2018, pp. 364–369. DOI: 10.18653/v1/P18-2058.

[5] Minghao Hu et al. "Open-Domain Targeted Sentiment Analysis via Span-Based Extraction and Classification". In: *Proceedings of ACL*. 2019.

[6] Xin Li et al. "A unified model for opinion target extraction and target sentiment prediction". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2019, pp. 6714–6721.

[7] Maria Pontiki et al. "SemEval-2014 Task 4: Aspect Based Sentiment Analysis". In: *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*. Dublin, Ireland: Association for Computational Linguistics, Aug. 2014, pp. 27–35.