NLP Homework 2: Semantic Role Labeling

Gioele Migno
1795826

1 Introduction

Semantic Role Labeling (SRL) is a Natural Language Processing (NLP) task in which we are interested in analyzing the *predicate-argument* structures within a sentence, or more intuitively answering to question of *"Who did What to Whom, Where, When, and How?"* (Màrquez et al., 2008). A *predicate* is a word or a multi-word expression denoting an event or an action, and an *argument* is a part of the text linked in some way to a predicate. Once extracted the *predicate-argument* structures we perform two disambiguation steps by assigning to each predicate a sense and to each argument a *semantic role*. SRL can be seen as a pipeline of four subtasks: *1) Predicate identification* find predicates, *2) Predicate disambiguation* assign a sense to each predicate, *3) Argument identification* find the argument of each predicate, *4) Argument classification* assign a semantic role to each argument.

In this paper we will perform SRL in two different scenarios: *SRL_34)* results of predicate identification and disambiguation are given, therefore we serve only steps 3 and 4, *SRL_234)* only predicate identification is provided, so tasks 2, 3, and 4 must be performed.

2 Data Processing

The input of all models that we will see is enriched by performing POS Tagging using `nltk` library. We recall that a POS Tag indicates the morphosyntactic category of a word, thus this additional information is very useful to identify the arguments in a predicate-argument structure. The dataset provided in the assignment is split into a training and a validation set, some predicate senses and semantic roles contained in the validation set are not present in the training set, to avoid technical issues during validation and test predictions, we allow the model to predict a class as unknown.

3 SRL_34: Argument Disambiguation + Argument Classification

3.1 Model

For argument identification and classification steps, we use a single model (SRL_34) that performs them as a unique task. Let's see the single components of the proposed architecture.

Sentence representation

Our solution is based on a pre-trained BERT model used as word embedding that allows us to represent a given sentence in a predicate-aware manner. In particular, using a BERT-tokenizer we build the input sentence for the encoder in the form [[`CLS`], *sentence*, [`SEP`], *predicate*, [`SEP`]] as done in (Shi and Lin, 2019), in this way the vectorial representations of words inside *sentence* depend on the highlighted predicate, an example is shown in Fig. 1. A technical detail regarding the tokenization step is the fact that the tokenizer could divide a word in multiple tokens, therefore in the output of the encoder we would have multiple representations referring to a single word in the sentence. For simplicity, we chose to use only the vector that refers to the first token of a word and ignore the others. The tokenizer also returns a useful additional output called `word_ids`, it's a vector with length equal to the number of tokens in which the i-th element indicates what word in the sentence the i-th token refers to. In correspondence to a special token (i.e. [`CLS`] [`SEP`]) or tokens after [`SEP`], `word_ids` is equal to -99.

Add additional information

Once obtained the vectorial representations of the words using the output of BERT, we add additional information by concatenating to them three different things: *1) POS Tags*, *2) Positional Embeddings*, *3) Predicate sense one-hot*. This last is used to

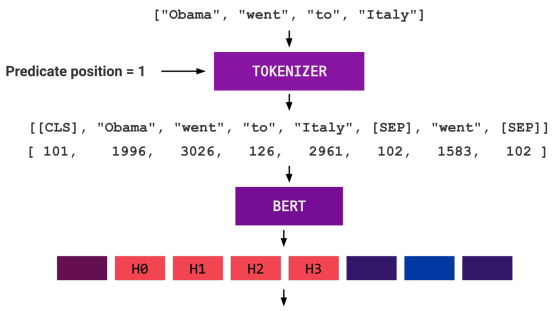

Figure 1: Predicate-aware encoding of the input sentence. The i-th input word is encoded as the token `Hi`

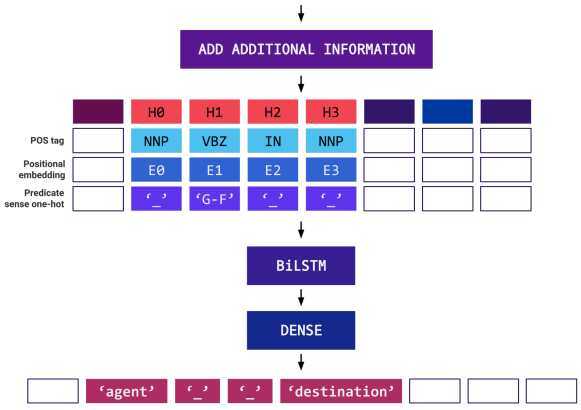

Figure 2: Final steps performed by SRL_34 model. Positional embedding are indicated with `Ei`. The white blocks represent padding tokens used to properly align the addition information to the input sentence. `'G-F'` is a predicate sense and stands for "GO-FORWARD". `'agent'` and `'destination'` are the semantic roles, `'_'` is the null tag.

highlight the position and the sense of the predicate of interest. The positional embedding instead is built by giving as input to a linear layer three quantities: *a) Predicate sense*: sense of the highlighted predicate (e.g. `GO-FORWARD`), *b) Predicate index*: position of the predicate in the input sentence (e.g. `1`), *c) Word index*: corresponding element of `word_ids` returned by the tokenizer. The idea behind the positional embedding is to give to the model a sort of distance of the current element with respect to the predicate, this distance is also influenced by the predicate sense. Fig. 3 shows an example of inputs given to build the positional embeddings.

Final classification

As done in (Shi and Lin, 2019), the final classification is performed using a BiLSTM layer followed by dense layers, see Fig. 2. Since we encode the input sentence by highlighting a single predicate,

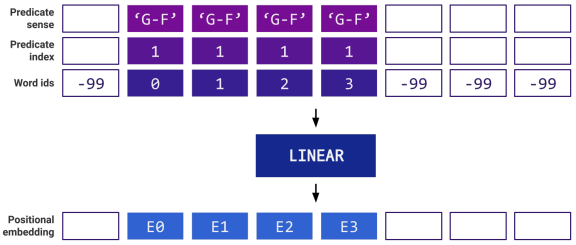

Figure 3: Example of inputs used to build the positional embeddings. `Ei` indicates the embedding generated for the i-th token

when the sentence has more than one we consider them one at the time, then we join all the predictions.

3.2 Experiments and results

In our experiments we use the pre-trained `distilbert-base-uncased` model provided by (Hugging-Face). The best performance can be obtained by fine-tuning it during the training. The positional embedding size is 40. The hidden size of the single BiLSTM layer is 300, by adding other layers the performance does not change, we also tried to remove the BiLSTM layer, but this caused a drop of performance of about 20% F1 score points. The final dense layers are composed of two layers with ReLU as hidden activation function and hidden size equal to 300.

To train the model we use Adam optimizer with learning rates 1e-05 and 2e-04 for the BERT model and the other components respectively, as loss function we use the standard cross entropy. During the training process we use as evaluation metrics the two functions provided in the assignment `evaluate_argument_identification()`, `evaluate_argument_classification()` rewritten as operations between tensors in order to take advantage of the hardware acceleration available (GPU), this dramatically reduces training time. To select the best weights for the model we monitor only the second metric since it's always lower than the first one. Figs 6, 7, 8 show loss function and the evaluation metrics during the training of 20 epochs on Google Colab. The results of the best model obtained are shown on Table 1, Fig. 4 shows instead the confusion matrix.

4 SRL_2: Predicate Disambiguation

To perform the predicate disambiguation step, necessary in the second scenario SRL_234, we use a standalone architecture (SRL_2) similar to the one

SRL_34 Tasks	P	R	F1
arg_ident	90.57	88.69	89.62
arg_class	85.96	84.18	**85.06**

Table 1: Precision (**P**), Recall (**R**) and F1 score (**F1**) obtained by SRL_34 model performing *argument identification* and *argument classification* on validation set.

used in SRL_34. The differences are: *1)* we do not give in input a positional embedding to the BiLSTM, *2)* in place of *predicate sense one-hot* we use *predicate index one-hot* that uses the standard binary encoding, *3)* the final dense layers have as additional input *predicate index one-hot* to help the model to predict a non-predicate word with the null tag '_'.

4.1 Experiments and results

The experiments and the training procedure are the same of SRL_34. Figs. 9, 10 show the training process. The confusion matrix is attached to this paper as external file `srl_2_cf.png` due to the high number of predicate senses.

5 SRL_234: Predicate Disambiguation + Argument Disambiguation + Argument Classification

5.1 Model

For the second SRL scenario in which only the predicates position is given, we simply join the previous two models, in particular we first predict the predicates sense using SRL_2 then we perform the final two steps using SRL_34.

5.2 Experiments and results

We initialized the two components SRL_2 and SRL_34 by taking the best weights obtained in the previous sections. We then performed a short training of 2 epochs with no noticeable improvements, that's probably due to lack of intermediate layers between the two models, indeed we directly provide the predicate senses predicted by SRL_2 to SRL_34 and not its logits for instance. Final performance are shown in Table 2, as could be expected we have lower performance on *argument classification* due to its strong dependency on the predicate sense. However, we have similar and slightly better performance on *argument identification* with respect to SRL_34. Fig. 5 shows the confusion matrix.

SRL_234 Tasks	P	R	F1
pred_disamb	88.37	88.09	88.23
arg_ident	91.26	88.37	**89.79**
arg_class	85.04	82.35	83.67

Table 2: Precision (**P**), Recall (**R**) and F1 score (**F1**) obtained by SRL_234 model performing *predicate disambiguation*, *argument identification* and *argument classification* on validation set.

6 Conclusions

We performed semantic role labeling task in two different scenarios: SRL_34 and SRL_234. In both of them, we always assumed *predicate identification* (step 1) as done. Thanks to the combination of a pretrained BERT model and a BiLSTM layer, we were able to achieve good performance in few epochs.

References

Hugging-Face. Hugging face - the ai community building the future. https://huggingface.co/. Accessed: 2022-09.

Lluís Màrquez, Xavier Carreras, Kenneth Litkowski, and Suzanne Stevenson. 2008. Semantic role labeling: An introduction to the special issue. *Computational Linguistics*, 34:145–159.

Peng Shi and Jimmy J. Lin. 2019. Simple bert models for relation extraction and semantic role labeling. *ArXiv*, abs/1904.05255.

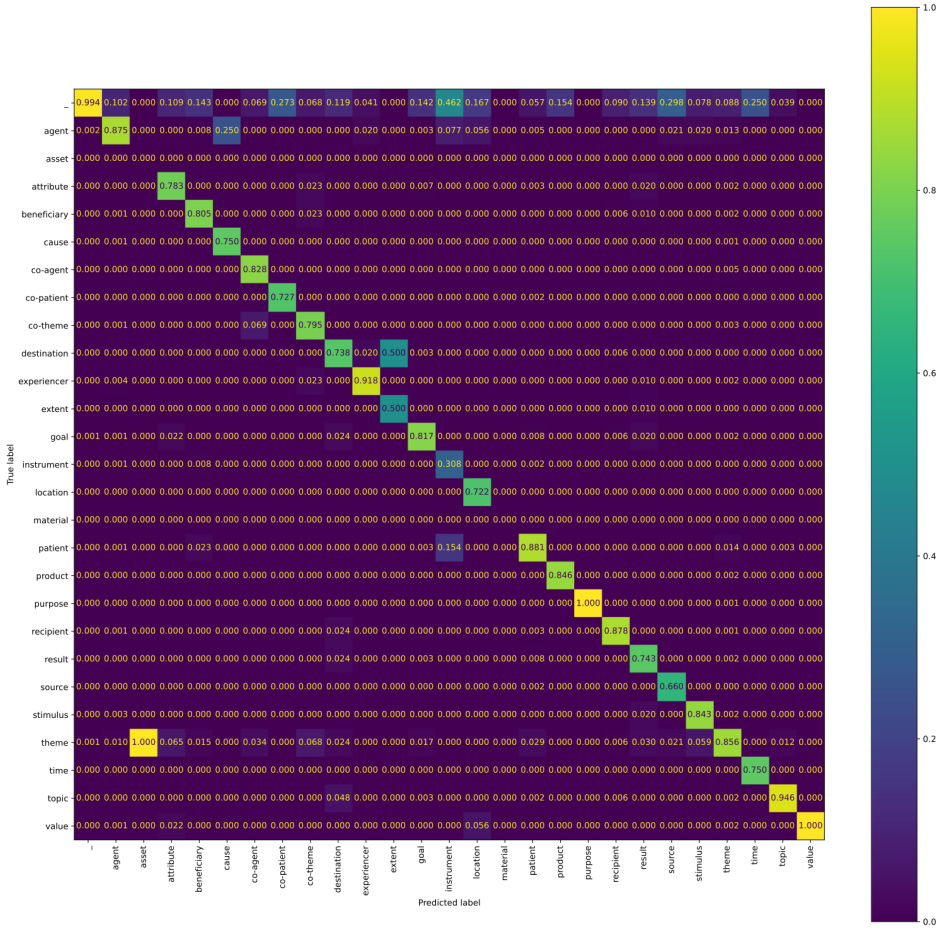

Figure 4: Confusion matrix normalized along model predictions (columns) referring to SRL_34 on validation set. The model predicts correctly 100% of the times `purpose` and `value` semantic roles but has very bad performance with `asset` and `instrument`. The model indeed, when predicts `asset` role, it is actually confusing `theme` with `asset`. Regarding `instrument` instead, the model confuses `patient` with `instrument` 15% of the time and `'_'` (null tag) with `instrument` 46% of the time.

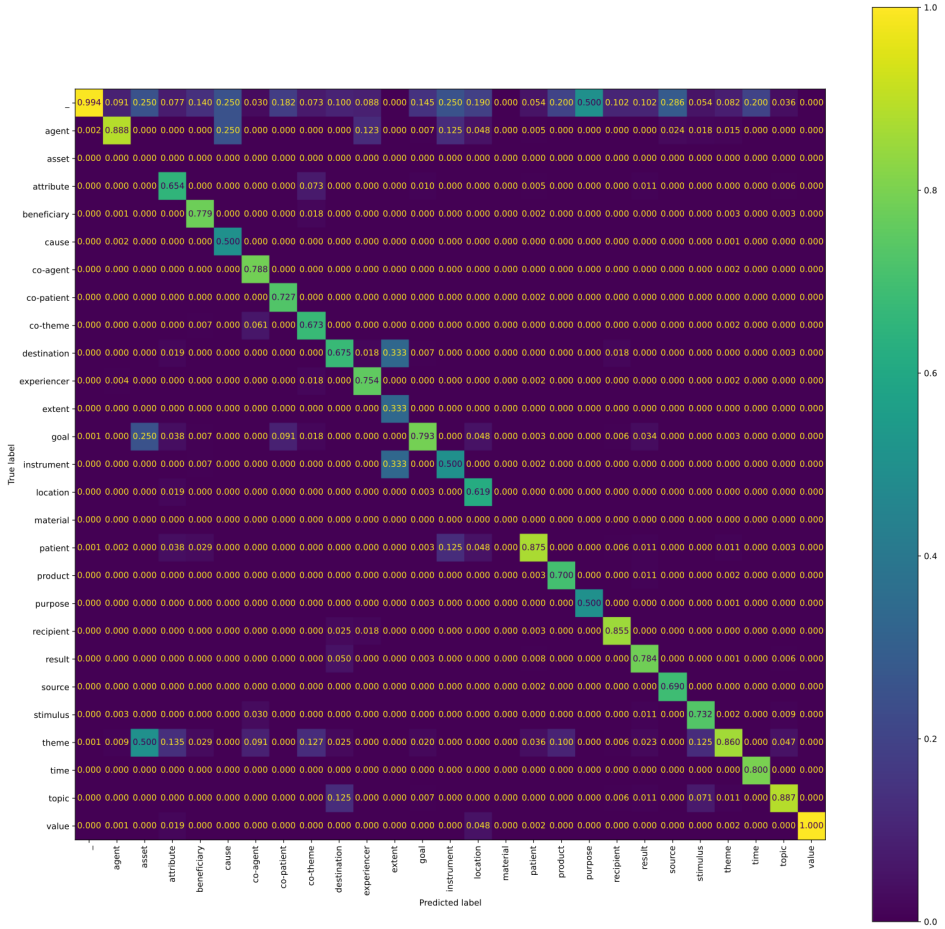

Figure 5: Confusion matrix normalized along model predictions (columns) referring to SRL_234 on validation set. With respect to SRL_34 we have similar performance on `value` and `'_'` but a dramatic drop on `purpose`. Indeed, half of the time in which it predicts a word as `purpose` the actual label is `'_'`.

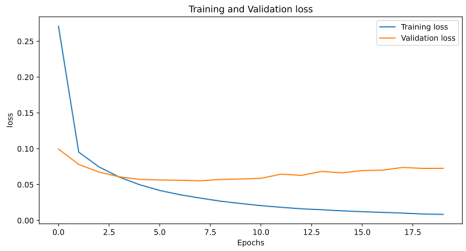

Figure 6: Loss value during training of SRL_34

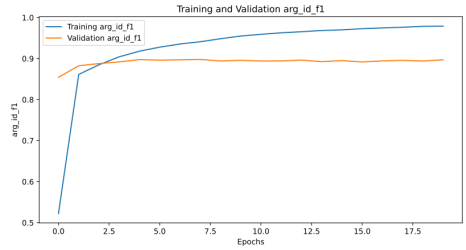

Figure 7: Argument identification F1 score during training of SRL_34

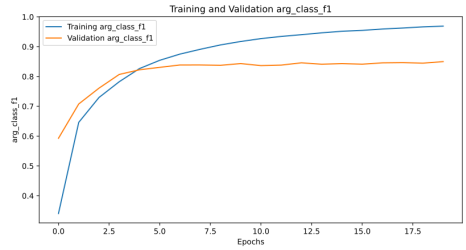

Figure 8: Argument classification F1 score during training of SRL_34

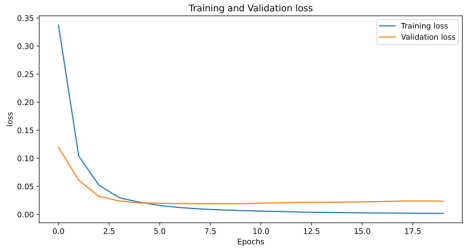

Figure 9: Loss value during training of SRL_2

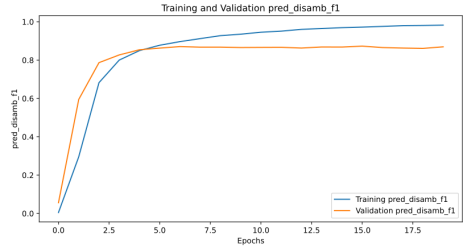

Figure 10: Predicate disambiguation F1 score during training of SRL_2