Semantic Role Labeling

Pietro Nardelli
nardelli.1546380@studenti.uniroma1.it

1 Introduction

Semantic Role Labeling (SRL) is the process that discovers the predicate-argument structure of a sentence, which plays a critical role in deep processing of natural language (Zhang et al., 2018). It essentially aims to determine "who did what to whom", "when", and "where" (He et al., 2017). SRL can be formed as four subtasks, including predicate detection, predicate disambiguation, argument identification and argument classification (Zhang et al., 2018). In this report, we aim to solve the last two subtasks, namely argument identification and argument classification, using different techniques inspired by (Marcheggiani et al., 2017), (Sun et al., 2020) and (Zhang et al., 2018) . In the following sections, we first introduce the preprocessing steps, then we describe the chosen model architecture and its variations that are compared during the analysis of results. Lastly, from the best configuration, hyperparameters tuning is conducted to achieve the final results.

2 Preprocessing

The dataset is composed of tokenized sentences each of which can be represented as words, lemmas, POS tags, dependency relations, dependency heads, predicate senses and roles. The predicate senses and semantic roles are defined according to VerbAtlas [1]. Knowing that SRL can be seen as a sequence labeling task the roles are the labels that the model needs to predict. If a sentence has no roles it is not added to the training dataset. Moreover if it has more than one predicate, we duplicate each sentence and the respective roles associated to that predicate. In order to feed the model with batches and speed up the training process, each input and output sequence needs to be of fixed size. For this reason, we decided to define each sentence as a window, such that when there is no more word in a sentence the remaining spots inside the window are padded. Moreover this is the most common method for defining the context of a word: it determines which contextual neighbours are taken into account. (Lison and Kutuzov) For the SRL task, in which a sentence relies heavily on its predicate, the context may be affected if a sentence does not belongs entirely to a window. So we decided to fix the window size (as well as window shift) equals to the size of the longest sentence found in both train and validation set (143). Based on the sentence representation used (as we will see in section 3) a window is then encoded in a vector of integers such that each integer is mapped to a token of the respective vocabulary.

[1] http://verbatlas.org/

3 Model Architecture

3.1 Base Model: M_{base}

Similarly to (Zhang et al., 2018) and (Marcheggiani et al., 2017) each encoded window is transformed thanks to a word representation component that from a word w_i in a window w build a word representation x_i. We represent each word w as a concatenation of 5 vectors: a randomly initialized word embedding x_r, a pre-trained [2] word embedding x_p, a randomly initialized lemma embedding x_l, a randomly initialized part-of-speech tag embedding x_{pos} and, a binary predicate flag x_{pf} (1 or 0) to indicate whether the current word is a predicate or not. The randomly initialized embeddings x_r, x_l and x_{pos} are fine-tuned during the training, while the weights of the pre-trained one are kept fixed. The final word representation is given by: $x_0 = x_r \circ x_p \circ x_l \circ x_{pos} \circ x_{pf}$ where $\circ$ denotes concatenation operation. A type of bidirectional

[2] wiki-news-300d-1M: 1 million word vectors trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset

recurrent neural network has been used to maintain and learn a state that summarizes past inputs, allowing to answering most of the problems dealing with sequential data (Mesnil et al.), and to process the sequence in forward and backward direction to obtain both past and future contextual information (Zhang et al., 2018). A Bidirectional Gated Recurrent Unit (Bi-GRU) is used: respect the Bidirectional Long Short Term Memory (Bi-LSTM) used in (Zhang et al., 2018) and (Marcheggiani et al., 2017), this network is simpler and in some cases more efficient (Khandelwal et al.), with faster progress in terms of both the number of updates and actual CPU time (Chung et al., 2014). The output of the Bi-GRU, is taken as input from a classifier that is made of two fully-connected layers. Experimental results have shown that the first layer needs a ReLU activation function and its number of units is equal to the number of hidden size divided by 2. The output layer labels each word in a sentence with a role, including the special 'NULL' ('_') role to indicate that it is not an argument of the predicate. The resulting model architecture can be seen in figure 1. Moreover, in order to address the issue of poor generalization results due to overfitting (Mesnil et al.), in addition to the dropout applied to the Bi-GRU, we implemented [3] the word dropout technique as in (Marcheggiani et al., 2017): we replaced a word with the UNK token with a probability $\frac{\alpha}{fr(w)+\alpha}$. The term α is an hyperparameter of the network and $fr(w)$ is the frequency of the word w. This technique will be activated during the hyperparameter-tuning, as we will see in section 4.2. As in (Marcheggiani et al., 2017) and (Zhang et al., 2018), given that SRL task is a multiclass calssification problem, we use a softmax layer over the output layer with the objective of maximizing the logarithm of the likelihood of the labels (cross-entropy loss). Finally, the learning algorithm chosen is Adam.

3.2 Model's variations

3.2.1 BERT Embedding: M_{BERT}

BERT (Bidirectional Encoder Representations from Transformers) is a language model representation that is designed to pre-train bidirectional representations by conditioning on both left and right context in all layers (Devlin et al., 2019). Its internal representations of words are called contextualized word representations because they are a function of the entire input sentence (Ethayarajh, 2019). BERT is basically a bidirectional Encoder stack of transformer architecture (Vaswani et al., 2017) and is released in two sizes: BERT-base and BERT-large. In addition to the base model described in section 3.1, we used the BERT-base model that contains an encoder with 12 Transformer blocks, a pooling layer and the hidden size of 768. BERT takes as input a sequence where the first token is always `[CLS]` (classification token) and another special token `[SEP]` is used for separing segments (Sun et al., 2020). We used the class `BERTEmbedder` [4] that receives as input a window w and produces BERT vectors of all the words therein. In order to do that, it first attaches to the sentence the special token previously described, then it tokenizes the sentences taking care of the indices of the words (to deal with WordPiece subwords tokenization (Devlin et al., 2019): e.g. "walked" is tokenized in "walk" and "##ed"), then the encoded sentences are passed as input to the BERT model in which the last 4 layers are summed, and finally the contextualized word representation x_b is produced as output. The final word representation in this model is: $x_1 = x_0 \circ x_b$.

3.2.2 BERT with Fine-Tuning and Differential Learning Rate : M_{FT}

The BERT model is pretrained over a general domain corpus (Devlin et al., 2019) but we need to adapt it to the target tasks. As suggested in (Sun et al., 2020) it's possible to obtain better results thanks to fine-tuning of the last layer of the BERT model. In fact, the higher the layer, the more context specific the contextualized representations and the more they move away from one another (Ethayarajh, 2019). From the model described in section 3.2.1, we freeze all the layers except the 12-th and the pooler one. Anyway, we need to take into account "catastrophic forgetting" problem (McCloskey and Cohen, 1989) that is a common problem in transfer learning in which the pre-trained knowledge is erased during learning new knowledge. Then, during the fine-tuning of BERT model, we cannot apply the same learning rate used so far. That is why we used a technique called differential learning rate, in which different learning rates are assigned to different part of the

[3] Part of the code for the implementation is taken from https://stackoverflow.com/questions/50174230/implementing-word-dropout-in-pytorch

[4] Colab notebook "NLP Notebook #7 - Sense Embeddings"

network: lr_{pool} for the pooler layer, lr_{bert} for the last Transformer block (12th layer) of BERT and, finally lr_{base} for the remaining part of the network. The model architecture is shown in figure 2.

3.2.3 Argument Boundary Indicator: M_{ABI}

As in (Zhang et al., 2018), following the observation that arguments usually tend to surround their predicate closely, we introduce two tags (in addition to the model described in section 3.2.2): Begin Of the Argument <BOA> and End Of the Argument <EOA>, indicating where the network should start and stop collecting argument candidates. For training, both tags are correspondingly assigned to the previous or next word as soon as the arguments of the current predicate have been saturated with previously collected words (an example is shown in figure 3). The tags should help the network to ignore those words too far away from the predicates which are unlikely ground-truth arguments. This will prevent the main problem in which the arguments and non-arguments distribution differs widely in quantity. For example, for the CoNLL 2009 dataset, the proportion of Args and NonArgs is 1:13. With the new tags the proportion reaches nearly 1:1 (Zhang et al., 2018).

3.2.4 Bi-LSTM instead of Bi-GRU: M_{LSTM}

The choice of type of gated recurrent unit depend heavily on the dataset and the corresponding task (Chung et al., 2014). For this reason we tried to substitute the Bi-GRU with a Bi-LSTM, driven also by the choice of (Marcheggiani et al., 2017) and (Zhang et al., 2018).

4 Evaluation procedures

4.1 Experimental Setup

Each experiment is evaluated with respect to the F1 score [5]. We intentionally made the models to be small enough in order to avoid overfitting which can easily distract the comparison. The initial hyperparameters of the base model M_{base} are also maintained constant for all the model's variations (see table 1). In particular, we added new hyperparameters for the model's variation M_{FT}: as already said, with an aggressive learning rate the training fails to converge. So, we used $lr_{pool} = lr_{bert} =$ 2e-5 as suggested in (Sun et al., 2020), while $lr_{base} = lr =$ 1e-3. Of course these hyperparameters are inheriteds and occur in the models that are based on it (M_{ABI} and M_{LSTM}).

[5] The computation is made from the provided `utils.py`

4.2 Experimental Results

The experimental results are summarized in table 2. We can see that the M_{ABI} model didn't perform well probably because the network learns to predict also that additional tags instead of use it as a signal to begin and stop the search of the arguments. This model is then discarded as well as M_{LSTM} that produces the worst performance, as pointed out also in table 3, confirming the fact that the Bi-GRU, chosen from the start, was the best sequential network in our case. The best model found so far is the one with BERT fine-tuned and differential learning rate (M_{FT}). So, we started from this model to perform hyperparameter tuning. In table 4 we can see the hyperparameter tuning results. We first grow the complexity of the model, increasing the number of features of the hidden state, then the number of the layers of the Bi-GRU. This provide great results mainly in the classification task, as we can see also in table 5. In order to exploit the differential learning rate technique, we try to increase the $lr_{pooling}$ from a value of 2e-5 to 5e-5. The results show an improvement which prove that this little learning rate increase may produce more context specific representations, avoiding the catastrophic forgetting problem. As we can see in figure 4, the model is not overfitting. We then try to increase the number of epochs from 5 to 6, reaching even better results even if the model slightly overfits (see figure 5). In order to contrast this problem, we finally use the word dropout technique with $\alpha = 0.25$ as suggested in (Marcheggiani et al., 2017). In fact, this will lead us to a non-overfitting model (figure 6), that is also the best result: we obtain a F1-score of 92.37% for identification task (see the confusion matrix in figure 7) and 86.25% for the classification task. The best model configuration can be seen in table 6.

5 Conclusions

We proposed a Bi-GRU based neural network to face the SRL task. From a base model, various configuration has been tried in order to improve the performance of the network. The best model chosen is the one with BERT model fine-tuned with differential learning rate. Finally, thanks to hyperparameters tuning, we reached the best performance.

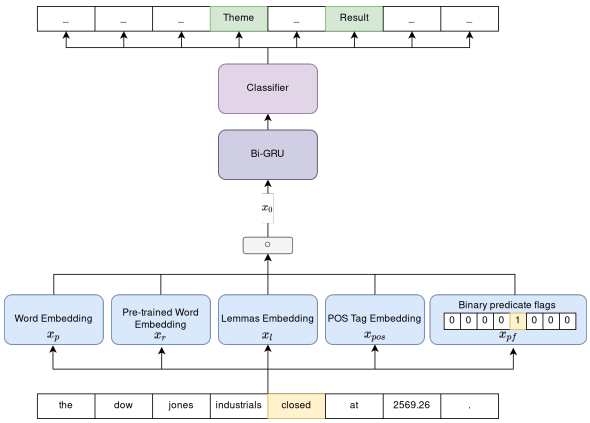

Figure 1: Architecture of the Base model (M_{base}). The input yellow word is a predicate that produce a binary predicate flags that is concatenated with all the other embeddings. The resulting concatenation x_0 is the input of the Bi-GRU.

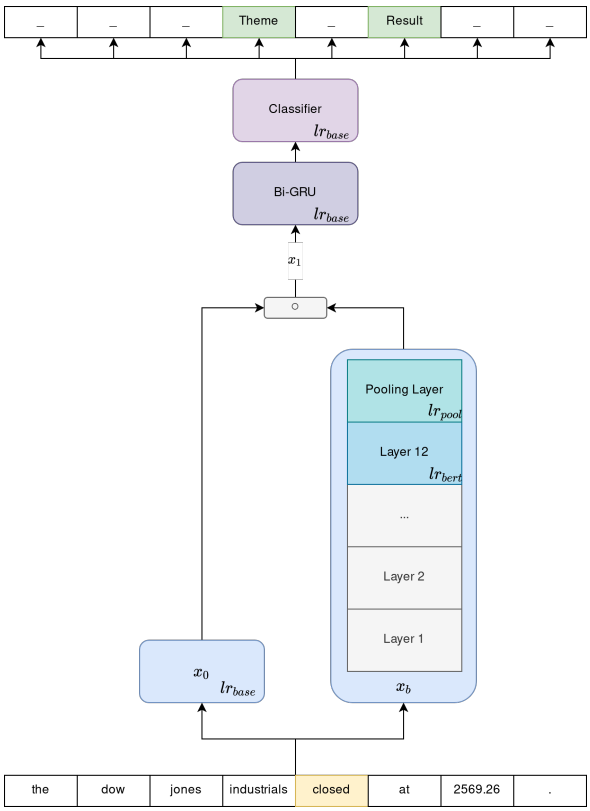

Figure 2: Model with BERT fine-tuning and differential learning rate (M_{FT}) in which the learning rates are highlighted. In particular we can see that the first layers of the BERT model are freezed, while the other have lr_{bert} and lr_{pool}. All the other components have instead a learning rate lr_{base} (except, of course, the pre-trained word embedding x_r that has fixed weights).

a	big	apple	**drops**	from	the	tree
_	_	A1	_	A3	_	_

a	big <BOA>	apple	**drops**	from	the <EOA>	tree
_		A1	_	A3		_

Figure 3: For example, the sentence "a big apple drops from the tree", has "drops" as predicate and two arguments A0 and A1. The two tags are assigned to the next words "apple" and "from" indicating no more arguments farther than them from the predicate (Zhang et al., 2018)

.

Hyperparameter	**Value**
word embedding pretrained size	300
word embedding size	300
lemmas embedding size	300
pos tags embedding size	16
batch size	128
(Bi-GRU) hid. size	100
(Bi-GRU) depth	3
dropout	0.1
α	0
lr	1e-3
epochs	5

Table 1: Initial hyperparameters setting used for the base model and the variants. Some of the embedding sizes, learning rate and dropout values are taken from both (Zhang et al., 2018) and (Marcheggiani et al., 2017). The α parameter is set to zero, in order to deactivate the word dropout (it will be activated during hyperparameter tuning).

Model	**I**	**C**
M_{base}	85.18%	70.11%
M_{BERT}	86.15%	75.21%
M_{FT}	**86.56%**	**75.42%**
M_{ABI}	84.17%	69.53%
M_{LSTM}	69.86%	59.27%

Table 2: Results of the model variations in term of F1-score on the Identification task (I) and Classification task (C). The best model so far is the one with BERT fine-tuned and differential learning rate (M_{FT}).

Model	Δ_I	Δ_C
M_{base}	-	-
M_{BERT}	**+0.97%**	**+5.1%**
M_{FT}	**+0.41%**	**+0.21%**
M_{ABI}	-2.39%	-5.89%
M_{LSTM}	-16.7%	-16.15%

Table 3: F1 scores differences, from the values reported in table 2, between a model and the previous best one (in bold), for both Identification (I) and Classification (C) task (e.g. M_{LSTM} is compared to M_{FT}). We can see that the simple implementation of BERT increased considerably the performance from the base model. The Bi-LSTM instead, produces a drastic decrease in the performance.

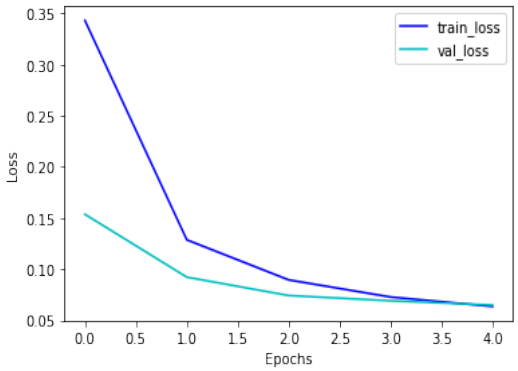

Figure 4: Train and validations loss at each epoch obtained from the training of the M_{FT} model with hidden size 512, Bi-GRU depth 4 and lr_{pool} = 5e-5. The model is not overfitting.

HParam	Value	I	C
(Bi-GRU) hid. size	512	91.86%	84.65%
(Bi-GRU) depth	4	91.68%	84.94%
lr_{pool}	5e-5	92.07%	85.02%
epochs	6	92.72%	86.1%
α	0.25	**92.37%**	**86.25%**

Table 4: Results of the hyperparameter tuning performed on M_{FT} model. The trials are listed in chronological order and each hyperparameter value modification is associated to an F1 score (both for Identification (I) and Classification (C) task). Each trial use the hyperparameter changed in the previous step.

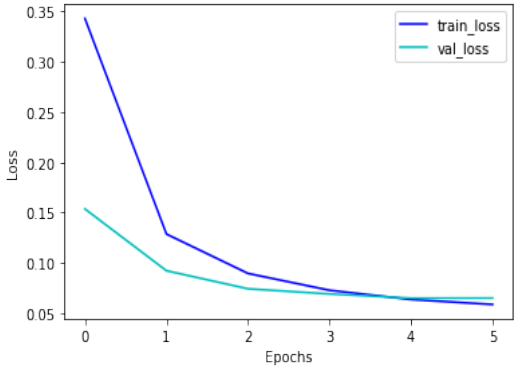

Figure 5: Train and validations loss at each epoch obtained from the training (for 6 epochs) of the M_{FT} model with hidden size 512, Bi-GRU depth 4 and lr_{pool} = 5e-5. As we can see the model is slightly overfitting in the last epoch.

HParam	Value	Δ_I	Δ_C
(Bi-GRU) hid. size	512	+5.3%	+9.23%
(Bi-GRU) depth	4	-0.18%	+0.29%
lr_{pool}	5e-5	0.39%	0.08%
epochs	6	0.65%	1.08%
α	0.25	-0.35%	+0.15%

Table 5: F1 scores differences, from the values reported in table 4, between a tuned model and the previous one, for both Identification (I) and Classification (C) task.

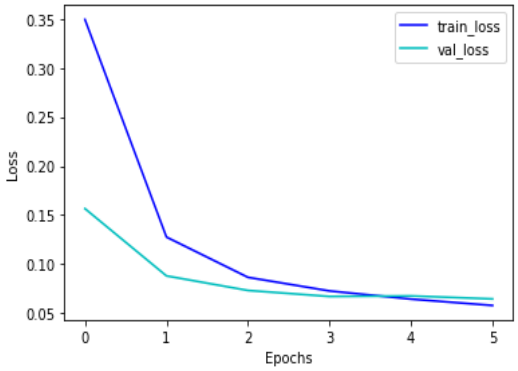

Figure 6: Train and validations loss at each epoch obtained from the training (for 6 epochs) of the (final) M_{FT} model with hidden size 512, Bi-GRU depth 4, lr_{pool} = 5e-5 and $\alpha = 0.25$.

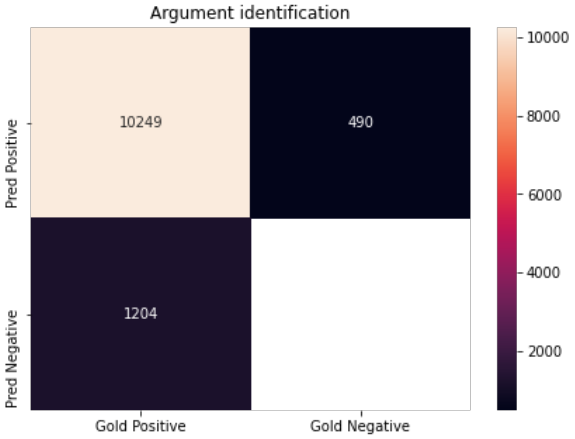

Figure 7: Confusion matrix of the M_{FT} model with its best configuration (table 6) referring to argument identification task. As we can see, there are no true negative samples predicted as negative because the 'NULL' labels (non-arguments) are excluded from the count. We can easily see that the model is more prone to identify non-arguments instead of arguments. This may due to the fact that the non-argument labels prevail over the argument ones. For example, even the short sentence "the dow jones industrials close at 2569.26.", has a proportion of args and non-args ratio of 1:4.

Hyperparameter	**Value**
word embedding pretrained size	300
word embedding size	300
lemmas embedding size	300
pos tags embedding size	16
batch size	128
(Bi-GRU) hid. size	512
(Bi-GRU) depth	4
dropout	0.1
α	0.25
lr_{base}	1e-3
lr_{bert}	2e-5
lr_{pool}	5e-5
epochs	6

Table 6: Final hyperparameters setting used for the H_{FT} model. This model with this configuration obtains (F1-score) 92.37% for identification task and 86.25% for the classification task.

References

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv:1412.3555 [cs]*. ArXiv: 1412.3555.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*. ArXiv: 1810.04805.

Kawin Ethayarajh. 2019. How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings. *arXiv:1909.00512 [cs]*. ArXiv: 1909.00512.

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2017. Deep Semantic Role Labeling: What Works and What's Next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 473–483, Vancouver, Canada. Association for Computational Linguistics.

Shubham Khandelwal, Benjamin Lecouteux, and Laurent Besacier. COMPARING GRU AND LSTM FOR AUTOMATIC SPEECH RECOGNITION. page 7.

Pierre Lison and Andrey Kutuzov. Redefining Context Windows for Word Embedding Models: An Experimental Study. page 5.

Diego Marcheggiani, Anton Frolov, and Ivan Titov. 2017. A Simple and Accurate Syntax-Agnostic Neural Model for Dependency-based Semantic Role Labeling. *arXiv:1701.02593 [cs]*. ArXiv: 1701.02593.

Michael McCloskey and Neal J. Cohen. 1989. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *Psychology of Learning and Motivation - Advances in Research and Theory*, 24(C):109–165. Publisher: Academic Press Inc.

Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. Investigation of Recurrent-Neural-Network Architectures and Learning Methods for Spoken Language Understanding. page 5.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2020. How to Fine-Tune BERT for Text Classification? *arXiv:1905.05583 [cs]*. ArXiv: 1905.05583.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *arXiv:1706.03762 [cs]*. ArXiv: 1706.03762.

Zhuosheng Zhang, Shexia He, Zuchao Li, and Hai Zhao. 2018. Attentive Semantic Role Labeling with Boundary Indicator. *arXiv:1809.02796 [cs]*. ArXiv: 1809.02796.