NLU course projects - Assignment 2 - NLU

Davide Cavicchini (247823)

University of Trento

davide.cavicchini@studenti.unitn.it

1. Introduction

The objective for this assignment is to jointly perform slot filling and intent prediction on the ATIS dataset. Intent prediction requires the model to classify the whole sequence to a predefined set of possible user intentions. Slot filling consists of identifying and classifying the relevant entities in the sequence, used to parameterize the user intent.

For the first part, we are tasked with implementing a functioning system using an LSTM backbone for the word token representations. For the second part, we are tasked with fine-tuning a pre-trained transformer BERT encoder to do the same task.

2. Implementation details

The simplest implemented architecture does not differ from the laboratory. It uses an embedding layer to convert the IDs assigned to each word to vectors, feeding them to a 1-layer LSTM model whose representations are used for classification. For intent classification only the last representation is used, while for slot filling each word representation is fed into the classifier. I chose to use an embedding size of 512 and a hidden size for the LSTM of 256.

To train the model I used multi-objective training. I use cross entropy for both tasks and define the weight for each class to avoid biases toward the majority class. However, I also want to have fine control over the weight of the two tasks. For this reason, I chose to take a 10% slice of the current batch to approximate the F1 score for the slot-filling task and the accuracy of the Intent classification task, used for dynamically rescaling the losses:

$$Loss = \frac{Loss_{slot}}{F1_{slot}} + \frac{Loss_{intent}}{Acc_{intent}}$$

In the first part, I trained the models using a learning rate of $5e-4$, while for the second part, I used $1e-4$. I utilized the AdamW optimizer and a batch size of 128 for both parts.

2.1. Part 1

Having set a baseline, the assignment required modifying the LSTM backbone to use **bidirectionality** and to add **dropout**.

Implementing a bidirectional LSTM itself is straightforward. But now the model has to work with double the representations for each token. The strategies I decided to experiment with to combine the two are the naive **concatenation** and **sum**, and the use of a **gated sum** inspired by the GRU gates. For the gated sum, I use the concatenated representations to feed a linear layer followed by a sigmoid activation to compute the scores S, the gated sum of the forward F and backward B representations will then be $R = S \odot F + (1 - S) \odot B$.

As per the dropout, I decided to add two dropout layers. One right after the embedding layer, and the other applied to the representations before feeding them to the classification heads. In my tests, both use a 0.5 probability of dropping the representations.

2.2. Part 2

As said, for this part I had to replace the LSTM backbone with a **BERT encoder**. The intent classification was straightforward since I only had to take the [CLS] token representations and feed them to the classifier. While realigning the tokens outputted by the BERT tokenizer to compute the slots at word-level required more work. Thankfully, the transformer library implements the "Fast Tokenizers" with the methods "word_ids" and "word_to_chars", which I used to extract only the first tokens of each word (separated by a space) to filter out the unwanted token representations. This idea comes from this paper [1].

I also decided to experiment with some level of **dropout** right after the BERT encoder. The idea is to use it as a stabilizer for the representations developed by the encoder during the pre-training, allowing the classification head to gradually adapt to them without causing too much catastrophic forgetting on the encoder. However, due to time constraints, I only trained these models once.

3. Results

The requested metrics to evaluate the model results are the F1 score for the slot-filling task and the accuracy for the intent classification. Table 1 presents the results from 5 training runs over the training set.

3.1. Part 1

The baseline model is already quite powerful in this dataset, but I was able to boost the performances of the model thanks to the use of the bidirectional LSTM. Looking at the combination strategy, we can see that the "gated sum" has a slight edge over the others, while "concatenation" and "sum" did not yield significant differences in performance. Applying dropout had an overall beneficial effect on both metrics, particularly on Intent accuracy.

3.2. Part 2

Fine-tuning an already pre-trained model required fewer epochs overall. However, due to the model size, it required a lot more time. The best-performing model for my training configuration was the *bert-base-uncased*. As expected, the dropout layer has a positive effect on the results. Meanwhile, preliminary experiments on the classification head size, which are not reported in the table, did not yield significant changes.

4. References

[1] Q. Chen, Z. Zhuo, and W. Wang, "Bert for joint intent classification and slot filling," 2019. [Online]. Available: https://arxiv.org/abs/1902.10909

Part	Model	**F1 Slot** (σ)	**Acc Intent** (σ)
0	LSTM	92.4 (0.4)	94.0 (0.3)
1.1	Bi-LSTM Concat	94.4 (0.2)	94.8 (0.4)
1.1	Bi-LSTM Sum	94.4 (0.1)	94.6 (0.1)
1.1	Bi-LSTM Gated	**94.7 (0.3)**	**95.2 (0.4)**
1.2	Bi-LSTM Drop Concat	94.2 (0.5)	96.1 (0.2)
1.2	Bi-LSTM Drop Sum	94.4 (0.2)	96.1 (0.3)
1.2	Bi-LSTM Drop Gated	**94.7 (0.1)**	**96.5 (0.5)**
2.0	BERT-base-uncased	**95.6 (0.2)**	**97.5 (0.4)**
2.0	BERT-base-cased	95.1 (0.2)	97.4 (0.3)
2.0	BERT-large-uncased	94.5 (0.6)	97.3 (0.1)
2.0	BERT-large-cased	94.7 (0.6)	97.4 (0.2)
2.0.1	BERT-base-uncased Drop	**96.1 (-)**	**97.6 (-)**
2.0.1	BERT-base-cased Drop	95.2 (-)	97.1 (-)
2.0.1	BERT-large-uncased Drop	94.5 (-)	97.1 (-)
2.0.1	BERT-large-cased Drop	95.0 (-)	97.5 (-)

Table 1: *Results for each configuration with the mean and std of the executed runs.* ***Bold*** *values represent the best performance for each part.*