Final NLU Project

Roberto Mazzaro (229301)

University of Trento

roberto.mazzaro@studenti.unitn.it

In this paper is analyzed the difference between a model and a baseline. In particular model and baseline use different encoders but same final classifiers and here is analized how small changes applied to the final layers of classifiers change performances both in the baseline and final model. Both models perform jointly slot filling and intent classification tasks and the evaluation is done on two different datasets common for those tasks, ATIS and SNIPS.

1. Introduction

Intent classification and slot filling are two essential tasks for NLU, those tasks are strongly correlated and joint learning methods that exploit and model the dependencies between the two tasks obtain better performance than independent models. Both baseline and model are built in that way, with a shared encoder, that extract features for both tasks, and two different classifiers for the two different tasks.

The baseline encoder consists in a simple bidirectional GRU with a single layer. While the final model encoder is a pretrained BERT model [1].

Final slot and intent classifiers are used both in the baseline and final model and I tried to use different layer combinations and in this paper I analyze the differences. For one of the configurations of the final model I reffered to the method proposed in the paper [2] for which an unofficial implementation is available at that link $https://github.com/monologg/JointBERT$ Firstly I re-implemented the model, I make it compatible with my dataset and requested evaluation methods and than I tried to modify it changing the classifiers, moreover I implemented the baseline model starting from the code of the 10th laboratory Finally I tested all the models and I tried to understand how performances change with different combinations of layers int the classifiers both for baseline and final model.

2. Task Formalisation

The task for this project consists in build a baseline and a final model able to simultaneously perform intent classification and slot filling tasks, in that way the shared encoder is able to learn stronger features that are common to both tasks, while final task specific layers can generalize well in a single task. The two tasks consists in:

- **Intent classification:** a text classification task where we have to assign an intent label to each given sentence (e.g. *"find a flight from memphis to tacoma dinner"*, intent: *"flight"*)
- **Slot filling:** a sequence labelling task where we have to assign to each token of the sentence a slot label in order to acquire more informations needed to perform the right action once understood the intent. (e.g. *"find a flight from memphis to tacoma dinner"*, slots: *"O O O O B-fromloc.city_name O B-toloc.city_name B-meal_description"*)

From that description is clear that the two tasks are highly correlated and that is why joint models perform better than independent ones.

The given baseline results are the following:

- **ATIS:** slot f1 score: 92% and intent accuracy: 92%
- **SNIPS:** slot f1 score: 80% and intent accuracy: 96%

Both the baseline and the final model are supposed to do better than that.

3. Data Description & Analysis

For the training and evaluation of the models I used two common datasets for these tasks, ATIS and SNIPS datasets. I used the JSON format for both of them, given by the professors during the 10th laboratory; in that format each sample is composed by the utterance, the sequence of slot labels, and the intent.

3.1. ATIS

ATIS means Airline Travel Information Systems and as the name suggests it is a dataset that includes audio recordings of people making flight reservations. Training set contains 4978 samples and the test set contains 893 samples. Since there is no development set, I created it with the same method used in the 10th laboratory that split the train set taking into account the label distribution. In that way at the end training set contains 4381 samples, dev set contains 597 samples, that is around the 10% of the train set, and the test set still contains 893 samples. The total number of intents is 26, and the total number of slot labels is 129, but in the training set there are only 21 intents and 120 slot labels, due to that aspect test performances are always $2-3\%$ points lowers that dev results. Moreover some slots and intents have a very low support, so they are difficult to be predicted since there are not enough examples to train the model. Vocabulary is quite small in fact vocabulary length is 863 words, for that reason ATIS can be considered a quite simple dataset.

3.2. SNIPS

The SNIPS dataset collected data from the Snips personal voice assistant. The training set contains 13084 samples and the test set contains 700 samples. In the same way used for the ATIS dataset I extrapolated 700 samples from the training set to build the dev set. The total number of intents is 7 and slot labels are 72 but intents support is well distributed and more or less also the slot labels The vocabulary length is 10621, so SNIPS is a more complicated dataset and with that dataset the performance differences between baseline and final model are magnified.

4. Model

The main difference between baseline and final model is the encoder that presents very different architecture, require a different type of input and also need a different optimizer. Both

models use the same intent and slots classifiers and the same loss functions.

4.1. Baseline Model

Baseline model has the following structure:

- Embedding layer
- Bidirectional GRU
- Intent classifier
- Slot classifier

I started from the model given in the 10th laboratory by professors and I substituted the single layer LSTM, with a bidirectional GRU. I choose to use a bidirectional layer because it has an higher capability to learn the context and I decided to use GRU layer instead of LSTM one because GRU cell has less parameter to learn than LSTM cell so the training is faster. Also intent and slots classifier are different but I will analyze their structure after. Baseline input is quite simple, as done in laboratory 10 a simple white space tokenization is used, and sentences of a batch are all padded to the max length of their batch. The same procedure is applied to slots and a sequence of the same dimension of the tokenized utterance is obtained without further work. Once done that an id is assign to each word and slot label and intent. Word ids and sequence length are given as input to the encoder. The output of the encoder is given as input to the slot classifier while the intent classifier receive as input the last hidden state. Since the model is not pretrained, unlike the BERT model, weights of the model are pre initialized with a standard method. Adam optimizer is used.

4.2. Final Model

Final model exploit a pre trained BERT (Bidirectional Encoder Representations from Transformers) model as encoder and the model has the following structure:

- Pre trained BERT
- Intent classifier
- Slot classifier

I used English uncased BERT-Base model, it consists in a deep bidirectional model, with 12 layers, 768 hidden states, and 12 heads, pretrained on large corpora, that need to be fine tuned. BERT model require a particular word piece tokenization as explained from hugginface $https://huggingface.co/docs/transformers/tokenizer\_summary$. This kind of tokenization is lossy because it splits rare words into meaningful subwords, in that way a vocabulary of reasonable size, but still able to learn meaningful context-independent representations, is obtained. In addition, subword tokenization enables the model to process words it has never seen before, by decomposing them into known subwords. With this type of tokenization number of tokens is no more aligned with number of slots, so we have to add PAD_TOKEN to slot labels in the case a word is splitted into subwords. Moreover a special classification embedding ([CLS]) is inserted as the first token and a special token ([SEP]) is added as the final token of each sentence. The intent is predicted on the hidden state of the first special token ([CLS]) while for slot filling the final hidden states of other tokens is used as can be seen from the figure 1. AdamW optimizer is adopted since it uses a different way to compute weight decay than Adam optimizer and it is strongly suggested for transformer models because it is more efficient.

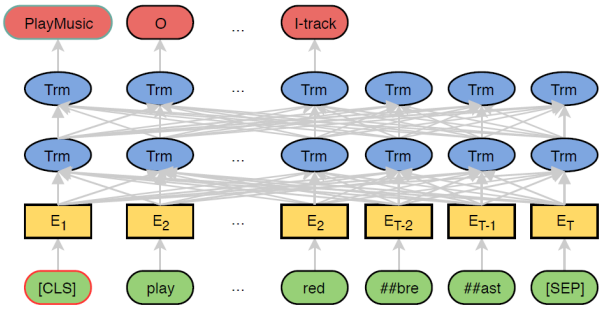

Figure 1: *BERT model for intent and slot classification.*

4.3. Intent and Slot Classifier

The unofficial implementation of the model available at $https://github.com/monologg/JointBERT$ as intent and slots classifier use a dropout layer combined with a linear layer with the possibility to add a CRF layer for the slot filling task since slot label predictions are dependent on predictions for surrounding words. Since dropout is often substituted with a batch normalization layer but we can't exploit it here because it introduces correlations between different examples, I tried to substitute dropout with a layer normalization that is often used with RNN and transformer models.

4.4. Loss Function

Cross entropy loss usually is used both for intent classification and slot filling task but if CRF layer is enabled, the negated output of the CRF forward is used, that correspond to a negative log likelihood.

5. Evaluation

For the evaluation, the main considered metrics, as requested in the instructions for the project are:

- **Accuracy** for the intent classification task. Accuracy consider the sum of the True Positives and True Negatives over the total amount of classified examples.
- **F1 score** for the slot filling task. F1 score is a metric that consider both the precision and recall scores. As requested I computed F1 score exploiting the evaluation function of conll that directly compute the F1 score from the true positive, false positive and false negative without considering the classes separately, this method is called *microaverage*.

I trained and tested baseline and final model with different layer combinations in the intent and slots classifier with both datasets. Moreover only for the ATIS dataset I did also a multi run execution to compute mean and standard deviations of the results; I didn't do that for the SNIPS dataset because this kind of test require plenty of time and even more with SNIPS dataset that is bigger than the ATIS one. I did 5 runs for the baseline and only 3 runs for the final model since its training is much longer.

5.1. Single run results:

Results of the single run tests are reported in table 1. From the table it can be noticed that both baseline and final model are above than the given baseline. The model with BERT encoder outperform the baseline with the bidirectional GRU in all

tests, especially when tested in the slot filling task on the SNIPS dataset that is more complex, here we can appreciate the ability of a more deep model with self attention mechanism to learn better the context. Furthermore on the ATIS dataset we are not able to improve a lot the performances because they are limited by the label distribution, as explained before, and not from the complexity of the dataset. About different intent and slot classifier there is not a configuration that it is always better than the others and differences in the results is small, however substitute dropout with layer normalization seems to be a good idea, the number of test where a configuration with layer normalization performs slightly better than the same configuration with dropout are 7 against 5. Also adding CRF seems to be in general a good idea to improve slots F1 score. I had some issues when I tested the final model with CRF on the SNIPS dataset but those issues are caused by conll because as reference it takes the target label set and if one or some predicted labels are missing from that set it rises an error, so it's not an issue of the model, there are other ways to compute F1 score that don't rise this error because valid labels can be specified.

Model	ATIS		SNIPS	
	Intent	Slot	Intent	Slot
BiGRU+D.out	**0.962**	0.934	0.959	0.860
BiGRU+D.out+CRF	0.914	0.932	0.953	**0.882**
BiGRU+Norm.	0.951	**0.941**	**0.973**	0.843
BiGRU+Norm.+CRF	0.936	0.939	0.950	0.878
BERT+D.out	0.972	0.954	0.981	**0.958**
BERT+D.out+CRF	0.973	0.955	**0.984**	–
BERT+Norm.	0.976	0.954	0.977	0.954
BERT+Norm.+CRF	**0.977**	**0.959**	0.981	–

Table 1: *Single run results.*

5.2. Multi run results:

Results of the single run tests are reported in table 2. This test can confirm observations done before. Final model always outperforms baseline and moreover usually it also has more stable results, the only exception is in the configuration with dropout + CRF. Substitute dropout with layer normalization is a good idea, configurations with layer normalization have always better results than the same configuration but with the dropout layer, apart for the baseline without CRF. Also adding CRF layer is beneficial for slot classification, configurations with CRF always perform better than the corresponding configuration without CRF.

Model	ATIS			
	Intent	std.	Slot	std.
BiGRU+D.out	**0.958**	0.004	0.935	0.004
BiGRU+D.out+CRF	0.931	0.002	0.939	0.002
BiGRU+Norm.	0.952	0.003	0939	0.003
BiGRU+Norm.+CRF	0.947	0.002	**0.948**	0.002
BERT+D.out	0.975	0.001	0.956	0.001
BERT+D.out+CRF	0.972	0.003	0.957	0.003
BERT+Norm.	0.976	0.001	0.957	0.001
BERT+Norm.+CRF	**0.977**	0.001	**0.961**	0.001

Table 2: *Multi run results.*

5.3. Training and dev error

Observing the graphs in figures 2 and 3 about training and deviation errors along epochs I noticed that, when using layer normalization the model seems to have a greater tendency to overfit the training dataset, in particular when tested on the SNIPS dataset. For example in the configuration with CRF, that is the one that usually produce better results, at the 10th iteration the model with dropout has a train loss of 0.103 and a validation loss of 1.664, that means a difference of 1.561. While the model with layer normalization has a train loss of 0.087, smaller with respect to the other model, and a validation loss of 1.786, greater than the other model, that means that the difference between the two losses now is 1.699. However when tested on the test set model with dropout has a test loss of 1.997 while the model with layer normalization has has a test loss of 1.891. I found that result quite strange also because dev set is obtained from the train set. Even if it can be thought that applying early stopping also here could be a good idea to solve that problem, in the paper [2] we can see how the best number of training epochs seems to be 30 but the model trained with 10 epochs performs better than the one trained with 20 epochs, so maybe it is a problem that mitigate with more training epochs; notice that in the paper they only use dropout layer.

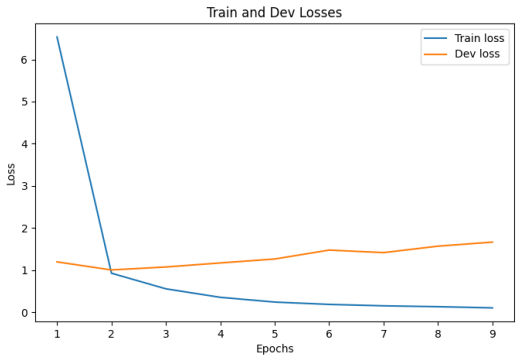

Figure 2: *Train and dev errors for model with dropout.*

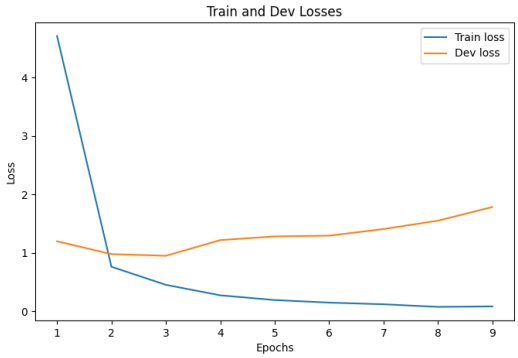

Figure 3: *Train and dev errors for model with layer normalization.*

6. Conclusion

Both the implemented baseline and final model performs better than the given baseline and the final model always outperforms the baseline and also produces more stable results in the multi run tests.

Add a CRF layer in the slot classifier improve performances in the slot filling task.

Substitute the dropout layer with a layer normalization usually produce a little improvement.

The difference between baseline and final model is more evident when the two models are tested on the SNIPS dataset that is more complex. With that dataset an important role is played by the higher capacity of the BERT model to better capture the context thank to the deeper architecture, the self attention layers and also the word piece tokenization that helps to reduce the vocabulary dimensions and recognize rare words splitting them into known sub-words.

Final model outperforms the baseline also on the ATIS dataset but the difference between the two models is lower because ATIS is a simpler dataset, so also the baseline with just a single bidirectional GRU layer works quite well. Moreover performances on that dataset are limited due to the fact that some labels present in the test set are not included in the training set, some of them are completely missing and others are, that are composed labels, are written in the other way around. Maybe, if it is allowed, better performances can be obtained just with a different split of the dataset or applying a preprocessing step in order to adjust labels.

Implementation of those models is available at the following GitHub link: $https://github.com/Rmazze/NLU\_proj\_2021-22\_Roberto\_Mazzaro.git$

7. References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *Google AI Language*, 2018.

[2] Qian Chen and Zhu Zhuo, "BERT for Joint Intent Classification and Slot Filling," *Speech Lab, DAMO Academy, Alibaba Group*, 2019.