Final NLU project

Mattia Carolo (232126)

University of Trento

mattia.carolo@studenti.unitn.it

1. Introduction

Multitask learning (MTL) is a field of machine learning where the main goal is to tackle multiple problems at the same time. In this work the focus is on Intent classification and Slot Filling which are two sentence level tasks where the framework needs to:

- *Assign an intent (can be more than one) for a given sentence or utterance*
- *map the sentence to a sequence of domain-slot labels*

As we can see this is the perfect environment for MTL where a model can be built in order to predict both tasks by leveraging even on the gained knowledge of the other task.

To make a comparison between different models i picked two that leverages on BiLSTM layers used as encoders, one is BERT and the last one uses a Stack-Propagation method.

2. Task Formalisation

The first task that needs to be solved is the intent detection part. In this part the task is typically a sentence classification problem where key features are passed through a classification algorithm to predict the class of a sentence or utterance from a specified set of classes (An example is "Have u seen my jacket" that will be classified as "Info Request).

The second critical task is slot filling where a label is attached to each token in an utterance. Through this method each token will have a representation that will encode his semantic information relative to the word represented. Moreover more tokens can be represented as a span that is represented using the BIO notation. In this case the problem is treated as a sequence labelling task (An example is " I want to travel to Rome" will become "O O O O O B-fromloc") [1]

By leveraging on MTL a model can be built so that it can learn and infer on two tasks together in order to be better than two working on the same task individually. This is reached thanks to the better generalization where the system prefers solutions which tackles more than one solution. Through this joint task the goal is to retrieve conditional relationship between the different models in order to better generalize once the learning is done [2]

3. Data Description Analysis

For this project both ATIS and SNIPS dataset were used, separately, in order to train the models as per intstruction.

3.1. ATIS

The ATIS (Airline Travel Information Systems) [3] consist on 4978 samples for the training set and 893 for the test. Since there was no validation test for this dataset I used the same method seen during lab lectures to rearrange the sets in order to obtain a validation test from the training one with the tight distribution. This results will reduce the size of the training sample to 4381 and will create a validation test with thte size of 597. Here 26 different types of intents are found and with 129 slot labels. Each sample of the dataset is composed by three entries that will be the same even for the next dataset. Those are:

```
{
    'utterance': 'what type of aircraft
does eastern fly from atlanta to denver
before 6 pm',

    'slots': 'O O O O O B-airline_name O O
B-fromloc.city_name O B-toloc.city_name
B-depart_time.time_relative
B-depart_time.time I-depart_time.time',

    'intent': 'aircraft'
}
```

3.2. SNIPS

The SNIPS dataset is composed of 13084 samples in the train set, 700 samples in the dev set and 700 samples in the test set. The total number of intents is 7, while the slot labels are 72.

Each sample is composed by the utterance, the sequence of slot labels, and the intent.

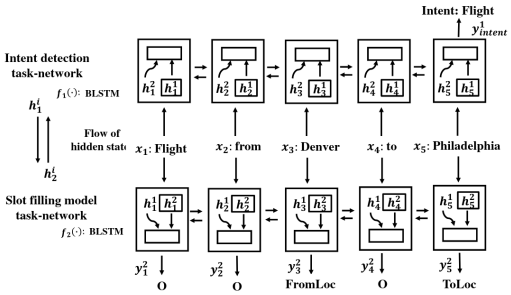
```
{
    'utterance': 'what type of aircraft
does eastern fly from atlanta to denver
before 6 pm',

    'slots': 'O O O O O B-airline_name O O
B-fromloc.city_name O B-toloc.city_name
B-depart_time.time_relative
B-depart_time.time I-depart_time.time',

    'intent': 'aircraft'
}
```

4. Model

For the comparison 4 models where trained in order to get various results that will be compared with the results achievable through the model introduced during classes. For the first three models the same loss was used which is a cross enthropy loss which leverages on both losses during training and testing

4.1. Bi-Model RNN

Bi-Model RNN [4] consists of two identical bi-directional LSTM models, one for predicting the intent and one for the slots. We can get a grasp of this by looking at Fig.1, the flow of hidden states between the two LSTM encoders connects the



(b) Bi-model structure without a decoder

Figure 1: *A bi-model structure without the decoder*

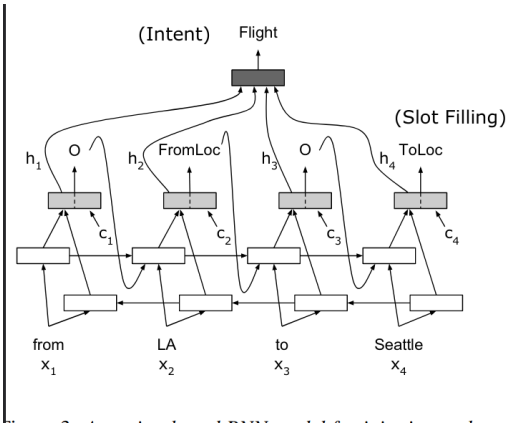


Figure 2: *A bi-model structure without the decoder*

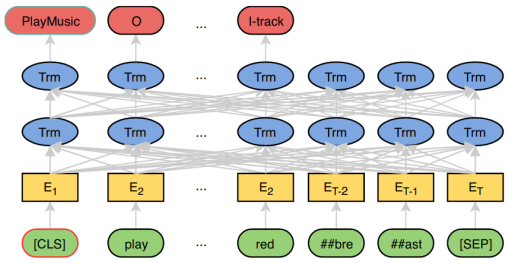


Figure 3: *Illustration of the Stack-Propagation framework*

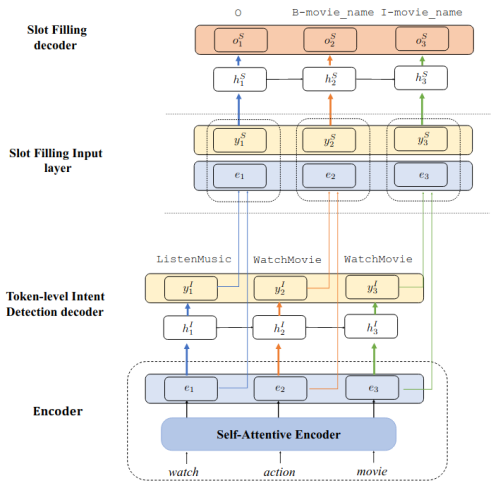


Figure 4: *Illustration of the Stack-Propagation framework*

information learned by the model about both intents and slots. This flow permits to the intent LSTM. Each encoder will contain two LSTM layer with a hidden dimensionality of 200 and here the intent of a sample will be computed only when each slot will have been determined. Finally for this model there wasn't an implementation given so I resorted to an unofficial representations where some adjustments are made like removing the decoder which reduced the time by a total of 54 minutus/epoch. Here

4.2. Bi-LSTM + Self-attention

Another model still based on BiLSTM is the one presented by Liu et al. [5] where a bidirectional LSTM encoder that receives the input token embeddings and feeds the output and hidden states to another LSTM layer. The second layer takes informations from the context as a weighted average of the first layer's output, like the attention mechanism. Finally, intent detection and slot filling is achieved using the outputs of the second layer. This Implementation consists on a Bi-LSTM encoder with a hidden size of 100 and a multi-head self- attention layer. The Bi-LSTM encodes the word embeddings to vectors while thanks to the attention layer discovers semantic dependencies

4.3. BERT

BERT [6] is a transformer-based network pre-trained on vast datasets for masked language modeling and next sentence prediction. Thanks to this it can be fine-tuned for various tasks which includes the one countered in this work

Is important to acknowledge that BERT uses WordPiece as a tokenizer which is capable of slitting complex words in smaller tokens and in order to work at every sample a new token [CLS] must be added to the start of the utterance and will work as token for the sentence classification

The authors of the paper have performed this experiment by feeding a prompt's utterance to BERT and taking the first token's output for intent prediction and all the resulting hidden states for slot filling.

4.4. Stack-Propagation SLU

In this model I haven't done any modifications if not for only a better readability or changing the the internal evaluator in order to have conll like evaluation. This particular model proposed because it takes a different approach during the learning part. With Stack-Propagation the model opposite to learn the correlations between different tasks by the shared encoder it uses the different tasks together during the learning so that different tasks can enhance the learning of others.

The architecture of this framework can be seen in Figure X where there is present one encoder and two decoders. It's worth noting that the BiLSTM is shared for both slot filling and intent detection part where thanks to a self attention mechanism it can leverage information acquired during the learning where after getting the intent information it can later relate the slots to a specific intent [7]

5. Evaluation

5.1. Baseline

As a baseline I took the values given to the project which are close to the performance seen during Lab 10 which is recapped in *Table 1*

5.2. Loss

I define for both models the prediction error as the sum of the cross-entropy between the ground truth and the predicted intent for both slot and intent.

I've choose this particular loss because different works related to Intent Detection and Slot Filling use this particular type of loss or a loss which is based on it. Moreover by experimenting with other basic losses The single loss for the intent is computed as a normal cross entropy loss where:

$$H_{\text{intent}}(p, q) = -\sum_{x \in \text{class}} p(x) log(q(x))$$

where p is the ground truth while q is the predicted value. A similar approach has been take even for the slot filling task where since a token can be associated with multiple type of spans the different losses are computed as the sum of every loss a token is from so that:

$$H_{\text{slots}}(p, q) = -\sum^{token\ slots} \sum p(x) log(q(x))$$

Still we can refine this by applying at the end an L2 Regularization with a λ factor of 0.001 just to prevent overfitting so that the final loss L will be defined as the sum of the previous two losses E and the L2 regularization where:

$$L = E + \lambda \sum_{w \in \text{model param}} w^2$$

5.3. Metrics

The main evaluation has been done on two main metrics, that were specified in the instruction of the project.

- F1 score for slot filling: the F1 score is a metric of accuracy that take into consideration both the precision and the recall of the predictions.
- Accuracy for intent classification: this is ratio of the sum of true positive and true negatives out of all the predictions.

5.4. Running Environments

Since I had a computer that could use the CUDA drivers I tried to run the most complex architectures in local to see if they would behave differently and how since the outputs times usually require the same amount of time elapsed. Interestingly the only model that showed slowing sings was the BERT model which in my computer required even an hour at doing just an epoch. This could be because of the high number of parameters the model has which can tend to be very heavy for a GPU.

Still every model has been tested on Google Colaboratory apart for the Stack-Propagation which has maintained his original structure

5.5. Results

The results can be summarized by Table 2 where for both SNIPS and ATIS datasets and as can be seen Stack-Propagation SLU collects far better results wrt. the others that barely reach a 10% negative difference from the given baseline on average. By taking into account the single runs sometimes we have better results that overcome the baseline but still in average especially the SNIPS datasets the result for the remaining models are quite low in average.

6. Conclusion

The first model in the paper claimed a very high performance on the given dataset but there wasn't any implementation. I tried asking to the curator of the unofficial implementations but it seems that even by adding layers or fine tuning some part the end results won't reach the one given by the authors which cannot be said for the second model that not only has reached the baseline given by the paper but in some runs it managed even to surpass it in average by taking only the epochs after usually 40 generations. Still the implementations made from him and others surpassed mine so a liability here can be found where my model is without a decoder which as showed by the original paper could result in lower performance.

The second one can be seen as the more robust ,if we don't count Stack-Propagation, offering a better generalization across the different datasets and not only averaging always better solutions than BERT but even in less time due to high number of BERT's parameters. And for the same reason I think BERT got worse results in a smaller dataset like ATIS where it needs multiple runs to achieve a result but due to the prohibitive amount of time though this was not possible

The last model was the only one that confirmed the results the original paper claimed. This is surely thanks to the pretrained parameters on the dataset which gives Stack-Propagation the edge. Still not only those parameters enhance the performance but even during the training the training for each epoch the elapsed time lowered almost linearly which I think is tanks to the Stack Propagation where with the time steps the algorithm learns,and share more informations through the different task

A concluding remark is that

7. References

[1] S. L. J. P. S. C. H. Henry Weld, Xiaoqi Huang, "A survey of joint intent detection and slot-filling models in natural language understanding," *Arxiv*, Feb. 2021.

[2] S. Ruder. An overview of multi-task learning in deep neural networks. [Online]. Available: https://ruder.io/multi-task/

[3] H. Anderson. Github page of the atis dataset. [Online]. Available: https://github.com/howl-anderson/ATIS$_dataset$

[4] Y. Wang, Y. Shen, and H. Jin, "A bi-model based rnn semantic frame parsing model for intent detection and slot filling," 2018. [Online]. Available: https://arxiv.org/abs/1812.10235

[5] B. Liu and I. Lane, "Attention-based recurrent neural network models for joint intent detection and slot filling," 2016. [Online]. Available: https://arxiv.org/abs/1609.01454

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

[7] L. Qin, W. Che, Y. Li, H. Wen, and T. Liu, "A stack-propagation framework with token-level intent detection for spoken language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1909.02188

	ATIS		SNIPS
Slots F1	**Intent Acc.**	**Slots F1**	**Intent Acc.**
0.92	0.94	0.80	0.96

Table 1: *baselines for both ATIS and SNIPS datasets*

Index	**ATIS**		**SNIPS**	
Models	**Slot F1**	**Intent Acc**	**Slot F1**	**Intent Acc**
Bi-Model RNN	0.89	0.7077	0.1352	0.8475
Bi-LSTM + Self-attention	0.8241	0.8385	0.7995	0.8629
BERT	0.7263	0.2001	0.7826	0.8371
Stack-Propagation SLU	0.9552	0.9880	0.958	0.969

Table 2: *Average precision on datasets*