Final NLU project

Davide Modolo (229297)

University of Trento

davide.modolo@studenti.unitn.it

Abstract

This is the report for the final project of the Natural Language Understanding course. The goal was to implement a neural network that predicts both intents and slots in a multitask learning setting. Every model implemented has been tested 5 times in order to produce a more accurate values.

1. Introduction

Joint intent detection and slot filling is a natural language processing task that involves identifying the intent of a user's input and extracting specific pieces of information that are relevant to that intent (labelling each word of a sentence to a sequence of domain-slot labels). This task is commonly used in conversational artificial intelligence systems, such as virtual assistants and chatbots, to understand and respond appropriately to user requests.

2. Task Formalisation

We were tasked with improving the results of a baseline model by 2/3%. Specifically, we needed to implement models that were capable of both slot filling and intent detection tasks (joint models), which are commonly grouped under the umbrella term "multi-task learning".

Slot filling is a sequence labelling task in which the goal is to assign labels to the words in a sentence or utterance, indicating the role each word plays in the overall meaning of the sentence.

Intent detection is a text classification task in which the goal is to assign an intent or purpose to a given sentence or utterance.

Since the datasets (ATIS and SNIPS, given by the professor) we used for training and testing the models were relatively small, we trained each model 5 times and took the average and standard deviation of the accuracy as the final result.

3. Data Description & Analysis

As already mentioned in the previous paragraphs, we were given two dataset to train and test our models:

- **ATIS** (Airline Travel Information Systems): dataset consisting of audio recordings and corresponding manual transcripts about humans asking for flight information on automated airline travel inquiry systems
- **SNIPS**: dataset of queries distributed among 7 user intents of various complexity

As required for this project, each sample consisted in a dictionary with three keys `utterance`, `slots` and `intent`.

3.1. ATIS

ATIS dataset has 5871 entries, divided in 5274 for train and 597 for test. I created a dev/validation dataset (with 893 entries) in order to make the code working with both dataset independently. It has been created using the ratio saw in laboratory (about 10% of training set skipping intents that had only one instance, leaving them in training).

Statistics:

- vocabulary size: 941
- slots number: 129
- intent number: 26
- label distributions: we can clearly see that this is an incredibly unbalanced dataset. The distribution across the entire dataset can be seen in Figure 1 (even if, given the high number of different labels, it difficult to distinguish them); for subsets statistics: Figure 5, Figure 6, Figure 7

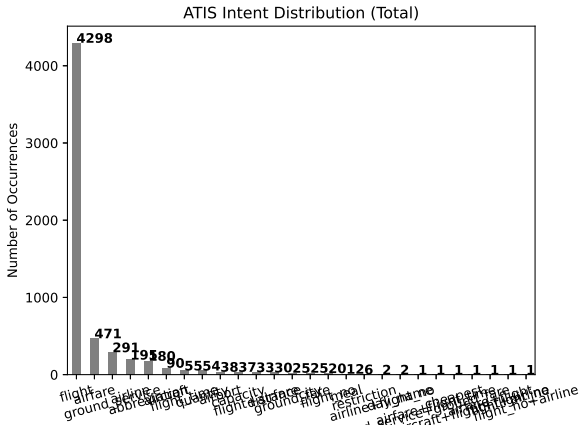

Figure 1: *ATIS label distribution in entire dataset*

3.2. SNIPS

SNIPS dataset was already divided in train, test and dev. It consists in 14484 entries with 13084 for train, 700 for test and 700 for validation.

Statistics:

- vocabulary size: 12134
- slots number: 72
- intent number: 7
- label distributions: the distribution across the entire dataset can be seen in Figure 2; for subsets statistics: Figure 8, Figure 9, Figure 10

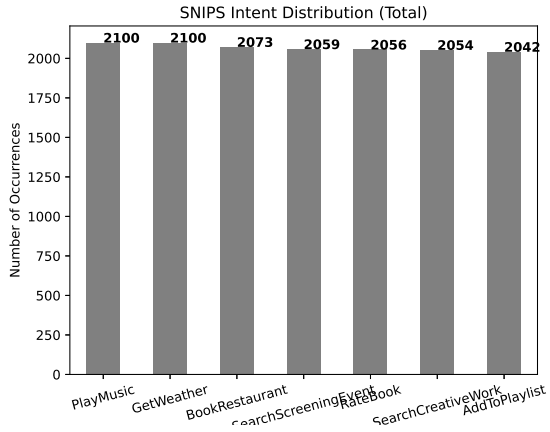

Figure 2: *SNIPS label distribution in entire dataset*

4. Model

Four models (other than the baseline) were trained for 5 times in 200 epochs but with early stopping set to 10. BERT and ERNIE model are in two different Python Notebooks due to training time needs and slightly different training a evaluating loop functions.

4.1. ModelIAS (baseline)

This is the model provided as baseline. First, the sentence is embedded with a PyTorch `nn.Embedding` [1] instance, then it is encoded using a non-bidirectional LSTM layer and through two linear layers we are able to retrieve the slot of each word in the starting sentence and the intent. My results differ a bit from the given performances and they can be seen in Table 1.

Dataset and Measure	Value[%]	s.d.
ATIS Slot F1	92,8386	0,2939
ATIS Intent Acc.	93,617	0,4308
SNIPS SlotF1	80,6082	1,04379
SNIPS Intent Acc.	97	0,4781

Table 1: *ModelIAS results*

4.2. Bidirectional ModelIAS

After reading the code of the baseline model, I wanted to try it setting the LSTM layer to bidirectional.

Unidirectional LSTMs only consider information from the past because the only inputs it has seen are "from the past". Bidirectional LSTMs, on the other hand, process input in two directions. This allows them to consider information from both the past and the future at any given point in time. BiLSTMs have been shown to perform well in tasks that require understanding context.

So, after changing the layer to bidirectional and changing dimensions where needed, I got better results (Table 2) in both Intent Accuracy and Slot F1 in the ATIS dataset and partial improvements even in the SNIPS dataset (getting the best results among the other models in the Intent Detection job).

Dataset and Measure	Value[%]	s.d.	difference
ATIS Slot F1	94,4575	0,1758	+1,6189
ATIS Intent Acc.	95,7671	0,2493	+2,1501
SNIPS SlotF1	87,314	0,4035	+6,7058
SNIPS Intent Acc.	96,6571	0,63	-0,3429

Table 2: *Bidirectional ModelIAS results*

4.3. Encoder-Decoder

In general, in an Encoder-Decoder model, the encoder processes the input sequence and produces a fixed-length context vector that represents the meaning of the input sequence. The decoder then, uses this context vector to generate the output sequence, one element at a time (Sequence-to-Sequence tasks).

To build this model, I took a cue from the paper presented by Bing Liu and Ian Lane [2], specifically selecting LSTM bidirectional layers (that also got me good performances in the Bidirectional ModelIAS model). I tried building a simplified version of it starting right from the Bidirectional model, enabling the Dropout layer that was present in the ModelIAS but was not used.

My Encoder-Decoder model starts embedding the sentence and encoding it with the first LSTM layer. Then, two LSTM layers are set for the decoding stage and in the end two Linear layers retrieve the predictions.

I tried removing the Dropout layer, getting better performances in the SNIPS dataset (surpassing in both Slot F1 and Intent Acc. the basline model, even if not by 2%), but the results in the ATIS dataset dropped by almost 2% losing all the improvement reached. So I decided to keep the Dropout and results can be seen in Table 3.

Dataset and Measure	Value[%]	s.d.	difference
ATIS Slot F1	94,9467	0,1174	+2,1081
ATIS Intent Acc.	96,4165	0,2554	+2,7995
SNIPS SlotF1	90,6623	0,4421	+10,0541
SNIPS Intent Acc.	96,6286	0,2321	-0,3714

Table 3: *Encoder-Decoder results*

4.4. BERT

Following the 'Final NLU Presentation' given by the professor, I tried implementing some pretrained models. I started with BERT[3], that is a transformer-based model.

My implementation works as follows: our vocabulary is added to the *BERT tokenizer*, than the sentence is passed into the BertModel (from the pretrained bert-base-uncase) and then in two Linear layers to retrieve the results. Specifically, in the forward function first I have to retrieve the starting sentence in order to have it in a format that the tokenizer (and so the BertModel) accepts, then I get the intent value from the `pooler_output`.

For the slot output, I take from the last layer (before the softmax of the Bert) the sequence with length equal to the length of the input sentence.

Even though the BERT model is a powerful one, I didn't get amazing results (Table 4)

Dataset and Measure	Value[%]	s.d.	difference
ATIS Slot F1	93,2161	0,1282	+0,3775
ATIS Intent Acc.	96,2598	0,9385	+2,6428
SNIPS SlotF1	86,8585	0,8564	+6,2503
SNIPS Intent Acc.	83,5714	2,3613	-13,4286

Table 4: *BERT results*

4.5. ERNIE

ERNIE[4] 2.0 is a natural language processing model that uses multi-task learning to improve representation learning of input text by integrating external knowledge from a large unannotated corpus, and has been applied to various NLP tasks. I loaded it into my model and, in combination with its Tokenizer, I run the model getting Table 5 results. Performances are in line with BERT, but with a lower training time.

Dataset and Measure	Value[%]	s.d.	difference
ATIS Slot F1	92,5169	2,4166	-0,3217
ATIS Intent Acc.	96,3046	1,7247	+2,6876
SNIPS SlotF1	94,4878	0,388	+13,8796
SNIPS Intent Acc.	84,9143	1,1125	-12,0857

Table 5: *ERNIE results*

5. Evaluation

For every model I used the same loss function: the sum of both intent and slot Cross Entropy Loss.

Adam optimizer [5] has been used (with its implementation in PyTorch [6]), with the difference that in the pretrained models I decided to force the lowering of the learning rate halving it every 3 epochs without any improvement in the results, in order to gain an additional 1% in the results (I tried both with and without my change).

As required, `CONLL.py` script was used for evaluating my results.

The first row of both Table 6 and Table 7 represents baseline, that are some times higher or lower than the ones we got in Lab10.

Metrics:

- *F1-Score*: metric used to evaluate Slot Filling results. It is a measure of a model's overall performance on the task, taking into account the balance between precision and recall for each class. It can be calculated as the average of the F1 scores for each class, weighted by the number of instances of each class;
- *Accuracy*: metric used to evaluate Intent Detection results. From ScikitLearn classification report [7], that is computed with: $\left(Acc.\% = \frac{T_P+T_N}{T_P+F_P+T_N+F_N} \cdot 100\right)$; it's the number of True Positive and True Negative predictions over all predictions. In a unbalanced set, it could not represent properly the actual performance of the model.

As already discussed, each training and each testing loop for every model was run five times; results can be seen in Table 6 and in Table 7 with average, standard deviation and difference from baseline as percentages.

5.1. Results

5.1.1. ATIS Results

measure	model	avg [%]	s.d.	difference
Slot F1	Baseline	92,8386	0,2939	
	Bi-dir.	94,4575	0,1758	+1,6189
	Enc-Dec	**94,9467**	**0,1174**	**+2,1081**
	BERT	93,2161	0,1282	+0,3775
	ERNIE	92,5169	2,4166	-0,3217
Intent	Baseline	93,617	0,4308	
	Bi-dir.	95,7671	0,2493	+2,1501
	Enc-Dec	**96,4165**	**0,2554**	**+2,7995**
	BERT	96,2598	0,9385	+2,6428
	ERNIE	96,3046	1,7247	+2,6876

Table 6: *ATIS Results*

As we can see in Table 6, with this dataset, Encoder-Decoder models reaches the goal in both metrics, surpassing the +2% threshold. Every model (other than ERNIE) still surpassed the baseline results in both metrics.

Furthermore, the plot in Figure 3 shows that the ERNIE model is the less "stable"; depending on the run we take in account, it can be the best or the worst model in my project. In the same plot we can see that the Encoder-Decoder model is the only one in the top-right square of the plane, showing it is the only one that surpasses +2% in both Intent Accuracy and Slot F1.

In the end, in the Intent Accuracy every model beats the baseline, but with slot F1 only Encoder-Decoder succeeds.

As already discussed in the explanation of Accuracy, there could be a possible problem with unbalancing between labels (Figure 1); having more than 70% of the dataset with the same label (and with some labels appearing only once or twice) can produce some unreliable results: a model that says always "flight" would get a 70% accuracy on intent detection, even though it is not making any useful prediction.

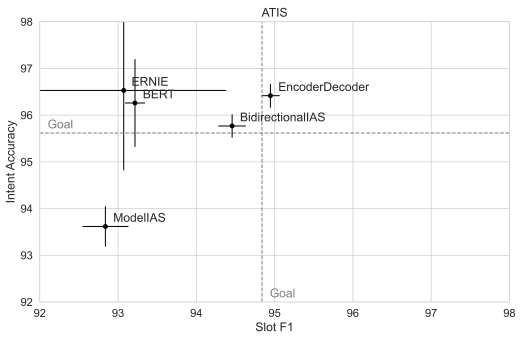

Figure 3: *Model results with ATIS*

5.1.2. SNIPS Results

SNIPS dataset, unlike ATIS, is very balanced: almost the same number of entry for each label (Figure 2).

First of all, looking at Figure 4, we can see that with this dataset, every model surpassed the goal line on Slot Detection, but no one did it on Intent Detection.

measure	model	avg [%]	s.d.	difference
Slot F1	Baseline	80,6082	1,04379	
	Bi-dir.	87,314	0,4035	+6,7058
	Enc-Dec	90,6623	0,4421	+10,0541
	BERT	86,8585	0,8564	+6,2503
	ERNIE	**94,4878**	**0,388**	**+13,8796**
Intent	Baseline	97	0,4781	
	Bi-dir.	**96,6571**	**0,63**	**-0,3429**
	Enc-Dec	96,6286	0,2321	-0,3714
	BERT	83,5714	2,3613	-13,4286
	ERNIE	84,9143	1,1125	-12,0857

Table 7: *SNIPS Results*

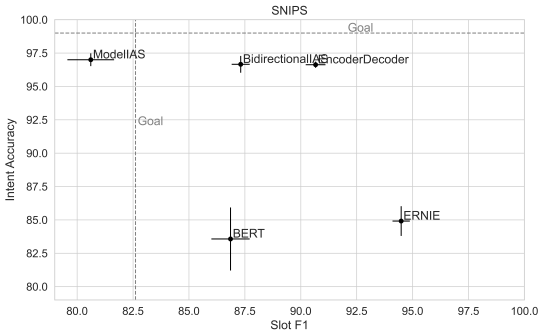

Figure 4: *Model results with SNIPS*

The performance of the baseline model in the intent detection task proved to be more challenging than the slot filling one, being difficult to surpass its performance.

Results can be seen in Table 7.

All of the implemented models were able to significantly improve upon the baseline model's performance in the slot filling task, with a high percentage of accuracy.

However, in the intent detection task, the results were not as promising. While my architectures results were largely similar (to the baseline), the pretrained models did not perform as well as expected.

But, if we take the 96% given in the starting presentation, Bidirectional Baseline and Encoder-Decoder model Intent Accuracy surpass it (even if not by +2% but about +0,6%).

6. Conclusion

The primary aim of this project was to improve the performance of both slot filling and intent detection tasks by experimenting with various approaches. These tasks are crucial for natural language processing systems as they enable the accurate interpretation and understanding of user inputs.

To begin, the results of a baseline model were used as a starting point, and efforts were made to improve upon these results through the modification of the architecture and the implementation of pre-built and pretrained models that were specifically tailored to the current task.

The results of these efforts are shown in Tables 6 and 7, and they demonstrate a consistently strong performance in both ATIS intent and slot filling, with only a slight deviation in the SNIPS slot filling task. In terms of SNIPS intent detection, the results obtained from the baseline training surpassed the 96% benchmark set by the professor. If we consider this 96% as the baseline SNIPS intent accuracy, the Encoder-Decoder and Bidirectional Baseline models outperformed it by approximately 0.65%.

One notable finding was that the Encoder-Decoder model consistently produced the best results overall, particularly in the ATIS task. It was also found to be the most stable model among the different runs. In the SNIPS task, it performed comparably well to the other models.

In conclusion, this project successfully improved the performance of slot filling and intent detection tasks through the modification of the architecture and the implementation of pre-built and pretrained models. The Encoder-Decoder model emerged as the most effective, producing strong results and displaying stability across different runs.

7. References

[1] PyTorch Embedding Class. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html

[2] Bing Liu and Ian Lane: 'Attention-Based Recurrent Neural Network Models for Joint Intent Detection and Slot Filling'. [Online]. Available: https://arxiv.org/abs/1609.01454

[3] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. [Online]. Available: https://arxiv.org/abs/1810.04805

[4] ERNIE: Enhanced Language Representation with Informative Entities. [Online]. Available: https://arxiv.org/abs/1905.07129

[5] ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. [Online]. Available: https://arxiv.org/pdf/1412.6980.pdf

[6] ADAM PyTorch. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

[7] sklearn.metrics.classification_report. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

8. Plots

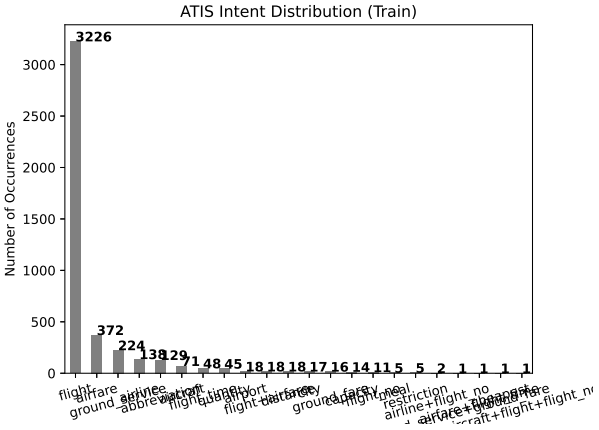

Figure 5: *ATIS label distribution in train set*

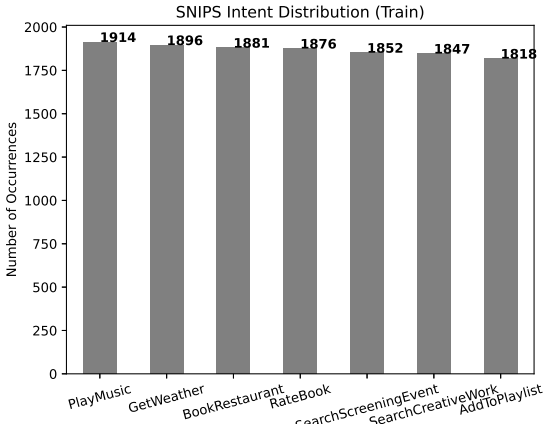

Figure 8: *SNIPS label distribution in train set*

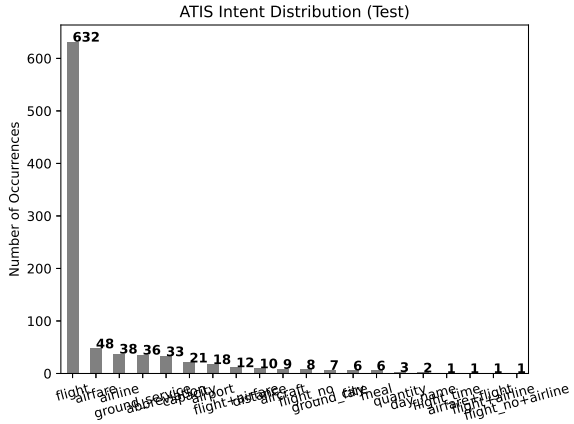

Figure 6: *ATIS label distribution in test set*

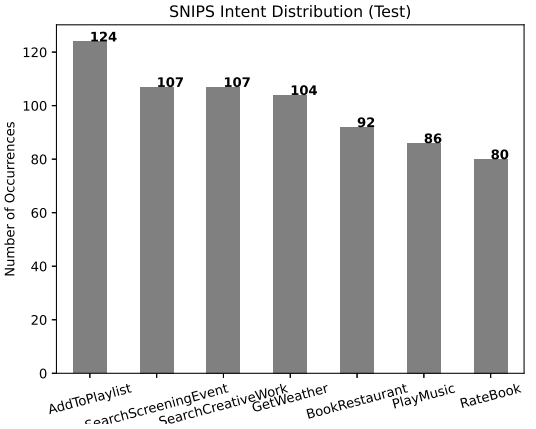

Figure 9: *SNIPS label distribution in test set*

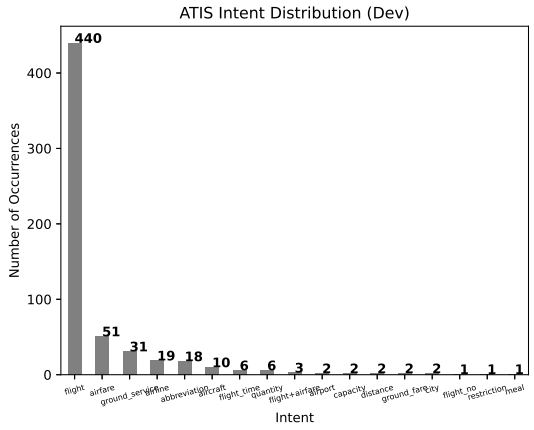

Figure 7: *ATIS label distribution in validation set*

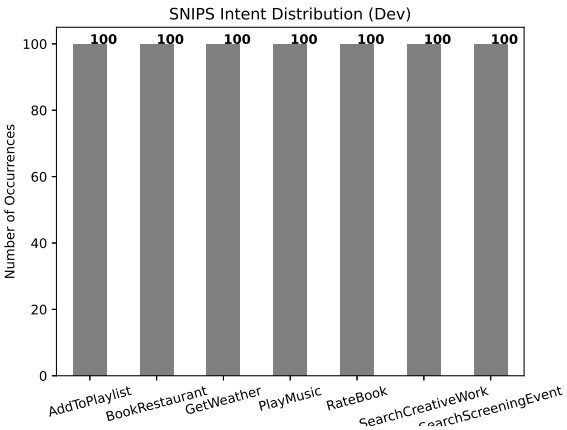

Figure 10: *SNIPS label distribution in valid set*