Intent classification & Slot filling (NLU) - Lab. 5

Luca Cazzola (248716)

University of Trento

luca.cazzola-1@studenti.unitn.it

1. Introduction

The goal is to achieve joint learning of **intent classification** & **slot filling** using as backbone deep neural architectures. Starting with the base structure (2) we'll progressively introduce bidirectionality (on LSTM) and dropout to regularize the model. After that we'll discard the embedding + LSTM component and substitute it with a more advanced transformer-based architecture which is BERT [1] to solve the same task. Code available at [2]

2. Implementation details

The base model structure is :

1. Embedding layer : projects word ids into a vector space
2. Recurrent layer : LSTM which extracts knowledge from the sequences
3. Linear layers (2) : one for slots, one for intents

The baseline is straight-forward to implement as all components are available on PyTorch. Switching over BERT, intent classification also has a 1-to-1 translation, as it's just necessary to substitute the input of the intent linear layer with the `pooler_output` [3] as in [4]. More complex was dealing with the slot filling task.

2.1. BERT Tokenizer

Transformer based models such as BERT digest sequences tokenized in a different way than NLP tools such as Spacy. Considering a sequence such as "Don't you love pasta?" BERT tokenizer would produce :

'[CLS]' 'Don' "'" 't' 'you' 'love' 'past' '##a' '?' '[SEP]'

Excluding the first and last tokens which represent the beginning and the end of sentence it's clear the sequence lenght is larger than the one produced by a white space separation. Punctuation marks are always separated and some words splits into what is generally referred as **subtokens**, marked by special chars ('pasta' $\rightarrow$ 'past', '##a'). Such behavior is problematic as in slot filling labels are provided at word (slot) level. If a word is splitted into n parts it means we're going to dispose of n last hidden states for such word, but only 1 label. This leaves to the choice of either combining the n results into 1, or to increase the number of labels to n.

2.2. Assessing the problem

To identify subtokens in a given sequence we can split the input sentence according to whitespaces and run the tokenizer word-per-word, discarding the first and last generated tokens ([CLS], [SEP]). If the sequence length > 1 subtokens have been generated. We keep track of both which words have been splitted and the corresponding subtoken positions in each sentence of the batch via a dictionary. In [4] to assess the discrepancy between number of outputs and labels they simply keep the first subtoken of each word and ignore the others. I've tested with both their solution and with one I've implemented by myself (2.3).

2.3. Subtoken merging

Under the assumption subtoken embeddings $\{y_2, \dots, y_n\}$ of a word retain some additional information which y_1 doesn't exploit by itself, the idea is to merge embeddings $\{y_1, \dots, y_n\}$ into a single one via a weighted sum. Such weights can be learned with a (multi-head) **self-attention** layer, which extracts the importance of each subtoken of the word. After implementing my solution I've realized such an approach probably falls into the category of **attention-pooling** mechanisms [5], as the attention output of the layer is unused and only the softmax component between Query-Key is what matters to reduce the input dimensionality ($n \rightarrow 1$). The unified representation is then used as hidden output associated to the word, instead of using y_1 arbitrarily.

3. Results

The dataset of reference is **ATIS**, evaluation metrics are the **accuracy** for intent classification and **F1 score** for slot filling. All configurations use early stop patience of 5 epochs.

3.1. Test (A) results

Model structure mentioned in (2) is used with one layer LSTM. Training is performed for 200 epochs multiple times per configuration. Dropout is applied with $p = 0.1$ at embedding, hidden and output level. As data suggest (tables 1 & 2) introducing bidirectionality brings benefits to performances, as the model is able to learn more robust representations. Adding some dropout for regularization also benefits the training, leading to better metrics. The choice of optimizer between SGD and AdamW didn't make much difference (except some sparse cases).

3.2. Test (B) results

BERT is used instead of the embedding + LSTM layers. Training is done 30 epochs multiple times per configuration. Dropout is applied before the last 2 linear layers with $p = 0.1$. Both the subtoken handling strategy proposed in [4] and the one propose by myself (2.3) are tested. Data (tables 3 & 4) suggests there's no significant difference between my implementation and the original paper one in both slot filling and intent classification tasks. It was expected that intent accuracy would have been almost the same, as (2.3) doesn't directly interfere with components related intent classification (except at loss level). Observing slot F1 scores being also very close between one implementation and another suggests that the assumptions made at (2.3) don't hold, at least in ATIS dataset. Perhaps the first subtoken hidden output y_1 already retains enough information about the word thanks to the BERT processing.

Model	Size	Learning Rates		
(SGD optimizer)		**5**	**3**	**1**
LSTM (baseline)	1.0M	91.27 ± 0.49	91.04 ± 0.92	$\mathbf{91.60 \pm 0.34}$
LSTM (bid.)	1.8M	94.03 ± 1.25	$\mathbf{95.60 \pm 0.23}$	95.18 ± 0.59
LSTM (bid.) + drop.	1.8M	95.30 ± 0.56	95.30 ± 0.11	$\underline{\mathbf{95.45 \pm 0.39}}$
(AdamW optimizer)		**5e-4**	**1e-4**	**5e-5**
LSTM (baseline)	1.0M	$\mathbf{93.62 \pm 0.30}$	94.33 ± 0.36	93.47 ± 0.28
LSTM (bid.)	1.8M	$\mathbf{94.66 \pm 0.17}$	94.14 ± 0.23	93.69 ± 0.51
LSTM (bid.) + drop.	1.8M	$\mathbf{94.96 \pm 0.81}$	94.77 ± 0.36	94.40 ± 0.59

Table 1: *Task (**A**) **Intent accuracy***

Bold values are the highest for that specific model configuration. The underlined value is the highest in the entire table.

Model	Size	Learning Rates		
(SGD optimizer)		**5**	**3**	**1**
LSTM (baseline)	1.0M	91.68 ± 0.35	$\mathbf{91.75 \pm 0.48}$	91.58 ± 0.23
LSTM (bid.)	1.8M	93.42 ± 0.29	$\mathbf{93.99 \pm 0.16}$	93.45 ± 0.27
LSTM (bid.) + drop.	1.8M	$\mathbf{94.19 \pm 0.41}$	94.07 ± 0.09	94.03 ± 0.29
(AdamW optimizer)		**5e-4**	**1e-4**	**5e-5**
LSTM (baseline)	1.0M	$\mathbf{92.74 \pm 0.27}$	91.92 ± 0.70	90.82 ± 0.38
LSTM (bid.)	1.8M	93.90 ± 0.17	$\mathbf{93.91 \pm 0.23}$	93.03 ± 0.16
LSTM (bid.) + drop.	1.8M	$\underline{\mathbf{94.50 \pm 0.23}}$	93.85 ± 0.66	93.17 ± 0.52

Table 2: *Task (**A**) **Slot F1 score***

Bold values are the highest for that specific model configuration. The underlined value is the highest in the entire table.

Model	Size	Learning Rates		
(Adam optimizer)		**1e-4**	**5e-5**	**1e-5**
BERT	109.6M	$\mathbf{97.35 \pm 0.23}$	97.24 ± 0.32	97.16 ± 0.13
BERT + merger	112.0M	97.28 ± 0.065	$\underline{\mathbf{97.61 \pm 0.36}}$	97.09 ± 0.59
(AdamW optimizer)		**=**	**=**	**=**
BERT	109.6M	$\mathbf{97.35 \pm 0.47}$	97.31 ± 0.40	97.05 ± 0.17
BERT + merger	112.0M	$\mathbf{97.20 \pm 0.19}$	97.13 ± 0.13	97.09 ± 0.30

Table 3: *Task (**B**) **Intent accuracy***

Bold values are the highest for that specific model configuration. The underlined value is the highest in the entire table.

Model	Size	Learning Rates		
(Adam optimizer)		**1e-4**	**5e-5**	**1e-5**
BERT	109.6M	$\mathbf{94.55 \pm 0.30}$	94.37 ± 0.08	93.29 ± 0.15
BERT + merger	112.0M	$\mathbf{94.61 \pm 0.20}$	94.56 ± 0.27	92.48 ± 0.39
(AdamW optimizer)		**=**	**=**	**=**
BERT	109.6M	$\underline{\mathbf{94.69 \pm 0.14}}$	94.66 ± 0.16	92.95 ± 0.30
BERT + merger	112.0M	94.58 ± 0.35	$\mathbf{94.69 \pm 0.20}$	92.48 ± 0.32

Table 4: *Task (**B**) **Slot F1 score***

Bold values are the highest for that specific model configuration. The underlined value is the highest in the entire table.

4. References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[2] L. Cazzola, "Nlu exam project," 2024. [Online]. Available: https://github.com/LuCazzola/NLU-exam

[3] HuggingFace.co, "Bert output," https://huggingface.co/docs/transformers/main_classes/output, accessed: 12/08/2024.

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

[5] F. Chen, G. Datta, S. Kundu, and P. Beerel, "Self-attentive pooling for efficient deep learning," 2022. [Online]. Available: https://arxiv.org/abs/2209.07659