NLU course projects

Alfredo Ceneri (249971)

University of Trento

alfredo.ceneri@studenti.unitn.it

1. Introduction

- *Optimization of LSTM* I first optimize the LSTM hyperparameters.
- *Adding bidirecionality* In order to add bidirectionality the size of the input for the last two linear layers must be modified and the forward and backward hidden states must be used accordingly for inference.
- *Adding a dropout layer* With the results obtained for the first lab in mind, i added a dropout layer after the embedding layer and a dropout layer before the last two linear layers.
- *Finetuning bert* In order to finetune a bert model i had to create a custom model based on bert with two additional linear layers for prediction of slots and intent. I also implemented a custom dataset and dataloader's collate function. Bert's last hidden state is used as input to two linear layers, similar to the ones used with an lstm module.

2. Implementation details

2.1. LSTM

For the optimization of the LSTM hyperparameters i used a surrogate model technique to minimize the sampled points. After the optimization i set these parameters: embedding size = 500, hidden size = 500, layers = 3, learning rate = 1e-3.

Adding bidirecionality was straightforward. Bidirecionality basically doubles the size of the output for the lstm: one output is the forward pass and one is the bacward pass with respect to the sequence.

The two hidden states are concatenated before being fed into the linear output layers.

For dropout i experimented with a few configurations: - adding dropout before the last two linear layers (to the hidden states) - adding dropout after the embedding layer - both

2.2. Finetuning Bert

For fineuning bert there are some details that need to be decided.

First of all, instead of loading the raw sentences for each batch, i pre-tokenize them and feed the encoded sentence to the model. This saves the computational cost of having to tokenize a raw sentence each time we need to make inference on it.

The most important detail i had to decide was how to handle the subtokenization issue.

The original dataset was built with one slot per each word (sentence split by whitespace). However the pre-trained tokenizer can split a word into different subtokens. During training, the number of slots predicted is equal to the total number of tokens split according to the tokenizer, not according to whitespaces. In order to fix this discrepancy, i used the same slot for subtokens of a given word.

During the evaluation, the slot assigned on the first subtoken is responsible for the whole word's slot.

Furthermore, since the tokenization of a sentence also produces two special tokens: the [CLS] and [SEP] tokens, and because those predicted slots should not count towards the calculation of the cross entropy loss, i handled this detail too.

3. Results

Table 1: *Assignment 2 Part 1*

Module parameters	slot f1 score	intent accuracy	parameters
best LSTM	96.9	96.7	best parameters
bidirecionality	97.1	98.3	””
last linear layer dropout	96.8	96.6	””
embedding dropout	97.1	97.4	””
both dropouts	97	97.9	””

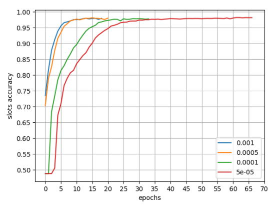
* embedding size = 500, hidden size = 500, layers = 3, learning rate = 1e-3, batch size = 8.

Table 2: *Assignment 2 Part 2*

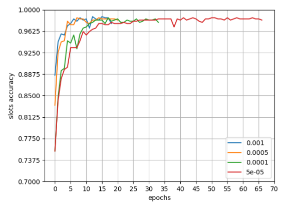
Module parameters	slot f1 score	intent accuracy	parameters
bert-base	97.6	99.5	lr = 5e-5, batch size

4. Images

Figure 1: *different learning rates for base lstm*

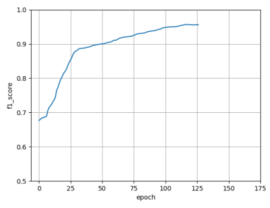


(a) slots f1 score

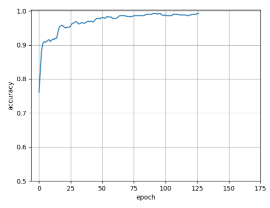


(b) intent accuracy

Figure 2: *Bert progress per epoch*

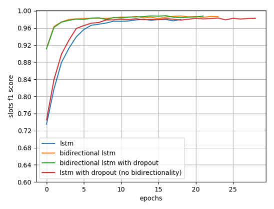


(a) slots f1

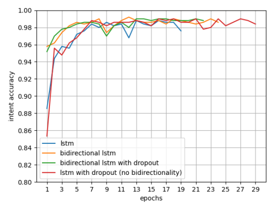


(b) intent accuracy

Figure 3: *comparisons between different settings for lstm*



(a) slots f1 score



(b) intent accuracy