NLP HW 2 - Aspect-level Sentiment Analysis

张含笑 181220078 181220067@smail.nju.edu.cn

Environment: Python 3.7.5 + Pytorch 1.7.0 + cuda 10.1 + Ubuntu 18.04 (on Google Colaboratory)

Related Work

- Transformer[1]
  - Including an encoder and a decoder, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely.
  - Be superior in quality while being more parallelizable and requiring significantly less time to train.
- BERT[2]
  - pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers.
  - can be finetuned with just one additional output layer to create state-of-the-art models for a wide range of tasks.
- Ada-Bert[3],[4]
  - Proposed by Alibaba Group, Ada-Bert leverages differentiable Neural Architecture Search to automatically compress BERT into task-adaptive small models for specific tasks. It incorporate a task-oriented knowledge distillation loss to provide search hints and an efficiency-aware loss as search constraints, which enables a good trade-off between efficiency and effectiveness for taskadaptive BERT compression.

Experiments

1. Bert-base-uncased

The code using Bert-base-uncased is in **bert_base_uncased.py**. Importing the package transformers, I use BertTokenizer to tokenize setenses, BertForSequenceClassification to classify simple raw sentences. I simply replace the tokens T with aspect terms, since there are many same sentences with different aspects, the accuracy of the first try is not very good, the result submission is 0.8123076923076923. I record the training loss of 40 epochs.

What's strange is that when I split all training data into train and val, the accuracy on val stays around 0.6. Maybe it results from the same sentences with different aspects and labels were splited into both train and val.

2. Bert-spc

Bert-spc is a model based on classifies senetnce pair, (raw sentence, aspect terms). Applying early-stop, the best result comes at 8 to 10 epoch. However, the submission result was not so good either, the accuracy was only 0.803, which was even less than simple sentence classification. I record the training loss of 10 epochs when applying early-stop.

3. Ada-Bert

To apply this solution, I used the pre-trained model BERT-base-uncased and the model provided by (https://drive.google.com/file/d/1DmVrhKQx74p1U5c7oq6qCTVxGIpgvp1c/view?

usp=sharing) and fine-tuned on my data set.

I first transformed the data from .txt form to .xml form, and parsed the data from a tree structure. The code is in **bert-ada.py**. The result of this try is 94%, and I record the training loss of 14 epochs.

To run bert-ada.py, use the command below:

```
python bert-ada.py \
--model_name_or_path ../data/models/restaurants_10mio_ep3 \
--data_dir=../data/transformed \
--output_dir=../data/models/run \
```

Details

1. Bert-base-uncased

- pre-trained model: Bert-base-uncased which has 12 layers, 768 hidden, 12 heads, 110M parameters
- number of epochs: 20
- batch size: 32 (for both train and test)
- optimizer: AdamW
- learning rate: 1e-5
- the maximum total input sequence length after tokenization: 128
- AdamW v.s. Adam: AdamW decays the weights in a manner that doesn't interact with the m/v parameters, which is equivalent to adding the square of the weights to the loss with plain (non-momentum) SGD.

```
update = next_m / (tf.sqrt(next_v) + self.epsilon)
```

I first load train and test file and transform which into dataframes in function data_clean(raw_train, raw_test), dataframes have three columns: id, text and label. In this try I simply replace the 'T' with aspect term. Then I split the train set into train and val, with val size = 0.15*len(train set).

```
for it in train:
    key = it[0]
    pattern = re.compile(pattern)
    newKey = re.sub(pattern, it[1], key)
    it[0] = newKey
    it.remove(it[1])
train_id = range(len(train))
train_txt = [train[i][0] for i in range(len(train))]
train_label = [(int(train[i][1])+1) for i in range(len(train))]
df = {"id": train_id, "text": train_txt, "label": train_label}
train = pd.DataFrame(df)
```

Then I train the model on train data in the function train_Bert(train) and do cross validation on val datain the function evaluate(dataset_val).

```
for epoch in tqdm(range(1, epochs+1)):
  num_epoch += 1
  model.train()
  loss_train_total = 0
  progress_bar = tqdm(dataloader_train,
                      desc ='Epoch {:1d}'.format(epoch),
                      leave=False,
                      disable=False
                      )
  for batch in progress_bar:
    model.zero_grad()
    batch = tuple(b.to(device) for b in batch)
    inputs = {'input_ids' : batch[0],
              'attention_mask' : batch[1],
              'labels' : batch[2]
              }
    outputs = model(**inputs)

    loss = outputs[0]
    loss.backward()
    torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

    optimizer.step()
    scheduler.step()
```

2. Ada-Bert

Since the way I process the input data and the way I transform input data into features are almost the same, I won't show the details of my usage of Bert-spc.

- pre-trained model: restaurants_10mio_ep3 based on Bert-base-uncased (which will be downloaded during the running)
- number of epochs: 14
- batch size: 32
- optimizer: AdamW
- learning rate: 5e-5
- the maximum total input sequence length after tokenization: 128
- max step: 800
- max gradient norm: 1.0
- number of updates steps to accumulate before performing a backward/update pass: 1

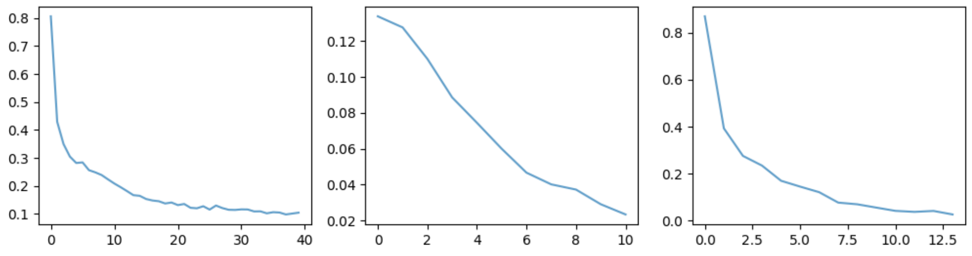
I first parse the input data from input data into sentence pair, and turn them into text_a, text_b, and label, the text_a is the origin sentence without 'T', the text_b is the aspect term, the label is the polarity to the aspect term in the context.

```
for i, sentence_pair in enumerate(sentences):
  guid = "%s-%s" % (set_type, i)
  try:
      text_a = sentence_pair[0]
      text_b = sentence_pair[1]
      label = labels[i]
  except IndexError:
      print("indexerror")
      continue
  examples.append(InputExample(guid=guid, text_a=text_a, text_b=text_b, labe
```

Then I fine-tune on the pre-trained model in the similar way to the model based on Bert-base-uncased, and generate predictions on test set.

Results

The accuracy on the three tries is 0.8123, 0.803 and 0.94. The training loss of epochs of them is as follows.



Acknowledgment

I would specially want to thank Taotian Pang for his help on the choise of pre-trained model.

References

[1] Vaswani, Ashish, et al. "Attention is all you need." Advances in Neural Information Processing Systems. 2017.

[2] Devlin, Jacob and Chang, Ming-Wei and Lee, Kenton and Toutanova, Kristina. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." arXiv preprint arXiv:1810.04805 (2018).

[3] Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. "AdaBERT: Task-Adaptive BERT Compression with Differentiable Neural Architecture Search." IJCAI-2020.

[4] Rietzler, Alexander, et al. "Adapt or get left behind: Domain adaptation through bert language model finetuning for aspect-target sentiment classification." arXiv preprint arXiv:1908.11860 (2019).