Jun 30, 2022

Coding Practice

- KoBERT

Kyomin Jung

Department of Electrical and Computer Engineering

MiLab @ Seoul National University

Contents

- **Practice**
  - Task description
  - Data Exploration
  - Sentiment Analysis (네이버 영화리뷰)
  - KorQuAD (Question Answering)

PRACTICE

- Task description
- Data exploration
- Task 1. Sentiment Analysis (네이버 영화리뷰)
- Task 2. Korea Question Answering Dataset (KorQuAD)

Task 1. Sentiment Analysis

- The task aims to classify the reviewer's polarity (positive/ negative) on the product.
- e.g. Amazon reviews, movie reviews
- We will use NAVER movie review data

Data Exploration

- Raw data

	text	sentiment
0	아 더빙.. 진짜 짜증나네요 목소리	0
1	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
2	너무재밋었다그래서보는것을추천한다	0
3	교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정	0
4	사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 ...	1
5	막 걸음마 뗀 3세부터 초등학교 1학년생인 8살용영화.ㅋㅋㅋ...별반개도 아까움.	0
6	원작의 긴장감을 제대로 살려내지못했다.	0
7	별 반개도 아깝다 욕나온다 이응경 길용우 연기생활이몇년인지..정말 발로해도 그것보단...	0
8	액션이 없는데도 재미 있는 몇안되는 영화	1
9	왜케 평점이 낮은건데? 꽤 볼만한데.. 헐리우드식 화려함에만 너무 길들여져 있나?	1

데이터셋, 모델정의전Remind!

- 데이터셋 정의
  - 데이터전처리: 모델에 입력할수있는형태로데이터수정 필
  - 수 정의할 함수:_____len__,___getitem__
  - 선택: collate_fn

- 모델 정의
  - 사전학습된언어모델에Classifier 추가하여fine-tuning
  - config, tokenizer, model 가져오기

Dataset

- 데이터 전처리
  - 토큰화: SKT 에서 학습시킨 KoBERT Tokenizer 사용*
  - 자연어 ✎ 토큰화 ✎ ID 변환

```
from kobert_tokenizer import KoBERTTokenizer
tokenizer = KoBERTTokenizer.from_pretrained('skt/kobert-base-v1')
```

Original text: TV용 건담시리즈중에서아직까지도최고봉

After tokenized:

['▁본', '지', '▁', '꽤', '▁지난', '▁후', '에', '▁남', '기', '지만', '▁', '.', '▁', '.', '▁', '.', '▁재
미있', '었', '음'] C

onvert tokens to ids:

[2, 694, 7003, 881, 5798, 2973, 4257, 6903, 3129, 5592, 5859, 4522, 6392, 3]

* https://github.com/SKTBrain/KoBERT

Model

- SKT 에서제공한KoBERT 사용
  - Transformers (huggingface) 라이브러리를사용하여모델생성SKT
  - 에서학습시킨pretrained KoBERT weights 가져오기Sentiment Ana
  - lysis 를 위한 classifier 와 결합하여 모델Fine-tuning

```
class KoBERTClassifier(BertPreTrainedModel):
    def __init__(self, config, args):
        super(KoBERTClassifier, self).__init__(config, args)
        config.num_labels = 2
        self.config = config
        self.args = args
        self.bert = BertModel(config)

        self.classifier = nn.Linear(config.hidden_size, config.num_labels)
        self.loss_fn = CrossEntropyLoss()

    def forward(self, input_ids, attention_mask, targets):
        output = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        pool_output = output[1]
        cls_output = self.classifier(pool_output)
        loss = self.loss_fn(cls_output, targets)

        return (loss, cls_output)
```

SKT 에서학습시킨KoBERT 가져오는부분

긍정/부정으로 예측하기 위한Classifier

학습셋업전Remind!

- 정의해야할 hyperparameter
  - Epoch Lear
  - ning rate Op
  - timizer
    - Scheduler
  - Batch size
  - 이외에도많음…

- loss.backward()
- optimizer.step()
- scheduler.step()

```python
def train(args, model, train_iterator, eval_iterator):

    t_total = len(train_iterator) // args.gradient_accumulation_steps * args.num_train_epochs
    optimizer = AdamW([{'params': model.bert.parameters()},
                       {'params': model.classifier.parameters(), 'lr': args.clf_learning_rate}],
                      lr=args.learning_rate, eps=args.adam_epsilon)
    scheduler = get_linear_schedule_with_warmup(
        optimizer, num_warmup_steps=args.warmup_steps, num_training_steps=t_total
    )

    for epoch in range(int(args.num_train_epochs)):
        tr_loss = 0
        model.zero_grad()
        model.train()

        for step, batch in enumerate(tqdm(train_iterator)):
            optimizer.zero_grad()

            input_ids = batch["input_ids"].to(device)
            input_mask = batch["input_mask"].to(device)
            targets = torch.tensor(batch["targets"]).to(device)

            loss, _ = model(input_ids=input_ids, attention_mask=input_mask, targets=targets)

            loss.backward()

            tr_loss += loss.item()

            # Batch size 가 작으므로 gradient 를 매 batch 마다 업데이트하지 않고 batch_size * gradient_accumulation_steps 마다 업데이트
            if (step) % args.gradient_accumulation_steps == 0:
                torch.nn.utils.clip_grad_norm_(model.parameters(), args.max_grad_norm
                )
                optimizer.step()
                scheduler.step()
                optimizer.zero_grad()

        tr_loss = tr_loss / len(train_iterator)

        eval_acc, eval_loss = evaluate(model, eval_iterator)

        print(f"Epoch: {epoch}, Train_loss: {tr_loss}, Accuracy: {eval_acc}, Eval_loss: {eval_loss}")

    return tr_loss
```

epoch 수 만큼 학습

collate_fn 으로 묶인 batch 를 각각 gpu 위에 올림

↑모델이 계산한 결과값 반환 (loss, logits)

← loss 기반 각 parameter gradient 계산

batch * gradient_accumulation_steps 데이터 단위로 모델 파라미터, learning rate 업데이트

정확도 계산함수

```
def calculate_accuracy(preds, y):
    max_idx = np.argmax(preds, axis=1)
    correct = (max_idx == y)
    acc = correct.sum() / len(correct)

    return acc
```

preds: (N, 4) 형태의numpy array
각 데이터별로 가장 높은 결과를 가진index 를 추출

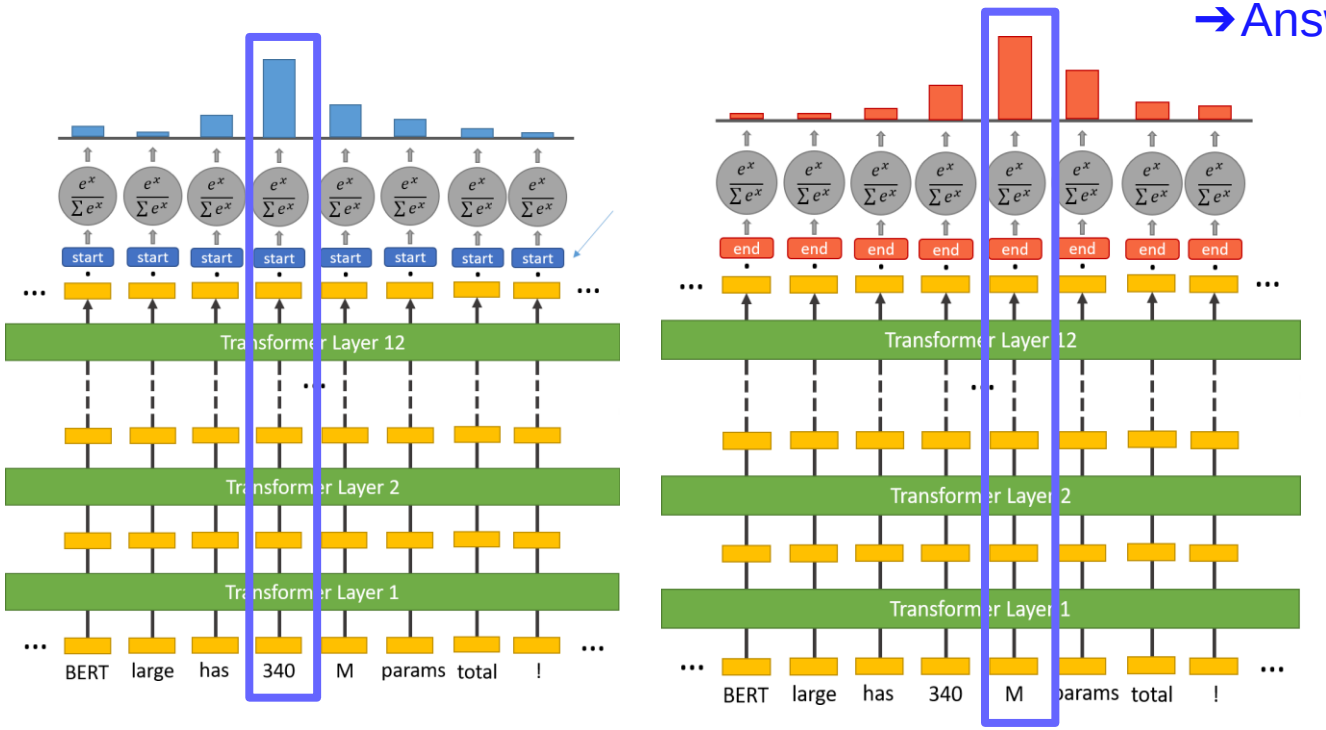
```
def evaluate(model, iterator):
    model.eval()
    labels = []
    preds = []
    eval_loss = 0.0
    with torch.no_grad():
        for batch in tqdm(iterator):
            input_ids = batch["input_ids"].to(device)
            input_mask = batch["input_mask"].to(device)
            targets = torch.tensor(batch["targets"]).to(device)

            loss, logits = model(input_ids=input_ids, attention_mask=input_mask, targets=targets)

            labels.append(targets.detach().cpu().numpy())
            preds.append(logits.detach().cpu().numpy())
            eval_loss += loss.item()

    labels = np.concatenate(labels)
    preds = np.concatenate(preds)
    acc = calculate_accuracy(preds, labels)
    eval_loss = eval_loss / len(iterator)

    return acc, eval_loss
```

결과를cpu 로이동

정확도 계산

Machine Reading Comprehension (MRC, QA)

- Assume the context contains the correct answer span.
- The model predicts which token marks the start of the answer, and which token marks the end.



Task 2. Korea Question Answering Dataset

- The task aims to extract the answer span in the given context.
  = extractive reading comprehension task
- The model outputs probabilities of tokens being the start and end position of answer span.

- SQuAD (Stanford Question Answering Dataset)
  - Version 1, 2 가존재
  - Non-answerable 문제가추가됨

- KorQuAD: Korean version MRC task

Data Exploration

- Raw data
  - 하나의context 에여러question-answer pair 가제공

QA example:
answers [{'text': '교향곡', 'answer_start': 54}]
id 6566495-0-0
question 바그너는 괴테의 파우스트를 읽고 무엇을 쓰고자 했는가?

Context example:
'1839년 바그너는 괴테의 파우스트을 처음 읽고 그 내용에 마음이 끌려 이를 소재로 해서 하나의 교향곡을 쓰려는 뜻을 갖는다. 이 시기 바그너는 1838년에 빚 독촉으로 산전수전을 다 겪은 상황이라 좌절과 실망에 가득했으며 메피스토펠레스를 만나는 파우스트의 심경에 공감했다고 한다. 또한 파리에서 아브네크의 지휘로 파리 음악원 관현악단이 연주하는 베토벤의 교향곡 9번을 듣고 깊은 감명을 받았는데, 이것이 이듬해 1월에 파우스트의 서곡으로 쓰여진 이 작품에 조금이라도 영향을 끼쳤으리라는 것은 의심할 여지가 없다. 여기의 라단조 조성의 경우에도 그의 전기에 적혀 있는 것처럼 단순한 정신적 피로나 실의가 반영된 것이 아니라 베토벤의 합창교향곡 조성의 영향을 받은 것을 볼 수 있다. 그렇게 교향곡 작곡을 1839년부터 40년에 걸쳐 파리에서 착수했으나 1악장을 쓴 뒤에 중단했다. 또한 작품의 완성과 동시에 그는 이 서곡(1악장)을 파리 음악원의 연주회에서 연주할 파트보까지 준비하였으나, 실제로는 이루어지지는 않았다. 결국 초연은 4년 반이 지난 후에 드레스덴에서 연주되었고 재연도 이루어졌지만, 이후에 그대로 방치되고 말았다. 그 사이에 그는 리엔치와 방황하는 네덜란드인을 완성하고 탄호이저에도 착수하는 등 분주한 시간을 보냈는데, 그런 바쁜 생활이 이 곡을 잊게 한 것이 아닌가 하는 의견도 있다.'

Data Exploration

- 데이터 전처리
  - MRC task 로사전학습된모델의Tokenizer 사용

```
from transformers import BertModel, BertConfig, AdamW
from kobert.tokenization_kobert import KoBertTokenizer

tokenizer = KoBertTokenizer.from_pretrained('monologg/kobert')
```

```
features, dataset = squad_convert_examples_to_features(
            examples=examples,
            tokenizer=tokenizer,
            max_seq_length=256,
            doc_stride=128,
            max_query_length=64,
            is_training=False,
            return_dataset="pt",
            tqdm_enabled=False)
```

```
# all_input_ids,
# all_attention_masks,
# all_token_type_ids,
# all_start_positions,
# all_end_positions,
# all_cls_index,
# all_p_mask,
# all_is_impossible,
```

Model

- Huggingface 에배포된KoBERT 사용
  - MRC 로pre-training 된모델('monologg/kobert')

```
model = BertForQuestionAnswering.from_pretrained('monologg/kobert')
tokenizer = KoBertTokenizer.from_pretrained('monologg/kobert')
config = BertConfig.from_pretrained('monologg/kobert')
```

```
for step, batch in enumerate(train_dataloader):
    model.train()
    batch = tuple(t.to(device) for t in batch)

    inputs = {
        "input_ids": batch[0],
        "attention_mask": batch[1],
        "token_type_ids": batch[2],
        "start_positions": batch[3],
        "end_positions": batch[4],
    }
    outputs = model(**inputs)
    break
    loss = outputs[0]
```

https://github.com/monologg/KoBERT-KorQuAD
https://huggingface.co/monologg/kobert

Model

- Huggingface 에배포된KoBERT 사용
  - MRC 로pre-training 된모델('monologg/kobert')

```
outputs
# loss
# start position logits
# end position logits
```

```
(tensor(2.6669, device='cuda:0', grad_fn=<DivBackward0>),
 tensor([[-0.0037, -0.1776, -0.0435,  ..., -0.3144, -0.0641,  0.3362],
         [-0.1546, -0.1929, -0.1595,  ..., -0.1245, -0.3728,  0.1035],
         [-0.0466,  0.0335,  0.2667,  ...,  0.3204,  0.2944,  0.2127],
         ...,
         [ 0.0305, -0.0441,  0.0813,  ...,  0.1357,  0.2252,  0.2461],
         [-0.1391,  0.0720,  0.1327,  ..., -0.0997, -0.1374,  0.2421],
         [-0.2181, -0.2751, -0.1392,  ..., -0.3499, -0.2571, -0.2693]],
        device='cuda:0', grad_fn=<SqueezeBackward1>),
 tensor([[ 0.4503,  0.0245,  0.1203,  ...,  0.3130,  0.0376,  0.3185],
         [ 0.3739,  0.1567,  0.2488,  ..., -0.0466,  0.0646,  0.1313],
         [ 0.3610,  0.2507,  0.0792,  ...,  0.2195,  0.2376,  0.3721],
         ...,
         [ 0.2227,  0.2222,  0.3178,  ...,  0.1600,  0.0495,  0.1642],
         [ 0.4490,  0.0126,  0.1354,  ...,  0.1613,  0.2899,  0.1805],
         [ 0.7047,  0.1114,  0.0291,  ...,  0.6068,  0.4884,  0.3393]],
        device='cuda:0', grad_fn=<SqueezeBackward1>))
```

https://git@github.com/SKTBrain/KoBERT.git

Training

- Huggingface 에서제공하는run_squad 사용
  - 데이터 전처리 ✎ 학습 ✎ output 전처리 ✎ Evaluation

```
!python3 kobert/run_squad.py --model_type kobert \
                        --model_name_or_path monologg/kobert \
                        --output_dir models \
                        --data_dir kobert/data \
                        --train_file KorQuAD_v1.0_train.json \
                        --predict_file KorQuAD_v1.0_dev.json \
                        --evaluate_during_training \
                        --per_gpu_train_batch_size 8 \
                        --per_gpu_eval_batch_size 8 \
                        --num_train_epochs 1 \
                        --max_seq_length 512 \
                        --logging_steps 200 \
                        --save_steps 200 \
                        --do_train
```

- Evaluation
  - Exact matching: 정답과정확히일치할경우
  - F1: harmonic mean of precision and recall