

---

# Machine Reading Comprehension

---

Practical Exercise

# 실습: MRC

예제 코드 다운로드:  
<https://github.com/KUNLP/KTAI-Practice>

- 대용량 언어 모델인 ELECTRA를 이용하여 기계 독해(MRC; Machine Reading Comprehension) 시스템을 구현 하시오.

질의와 문서를 입력으로 받아  
질에 대한 답변의 위치를 문  
서에서 찾아주는 인공지능 모델

대한민국 서쪽에는 어느 나라가 있나?

대한민국(大韓民國, 영어: Republic of Korea; ROK, 문화어: 남조선; 南朝鮮), 약칭으로 한국(韓國), 남한(南韓)은 동아시아의 한반도 남부에 있는 공화국이다. 서쪽으로는 서해를 사이에 두고 **중화인민공화국**이, 동쪽으로는 동해를 사이에 두고 일본이 있으며 북쪽으로는 조선민주주의인민공화국과 맞닿아 있다. 수도는 서울특별시이며, 국기는 태극기, 국가는 애국가, 공용어는 한국어이다.

## 데이터 구성

```
"qas": [
  {
    "id": "6548850-0-0",
    "question": "임종석이 여의도 농민 폭력 시위를 주도한 혐의로 지명수배 된 날은?",
    "answers": [{"text": "1989년 2월 15일", "answer_start": 0}]
  },
  ...
]
```

질의, 정답 어휘 및 위치

문서

"context": "1989년 2월 15일 여의도 농민 폭력 시위를 주도한 혐의 (폭력행위등처벌에관한법률위반) 으로 지명수배되었다. 1989년 3월 12일 서울



# 실습: MRC

- 질문과 문서를 하나의 Input Sequence로 변환하여 모델에 입력

- 질문 : 세종은 조선의 몇 대 왕이야?
- 문서 : 세종은 조선의 4대 왕으로, ~~~

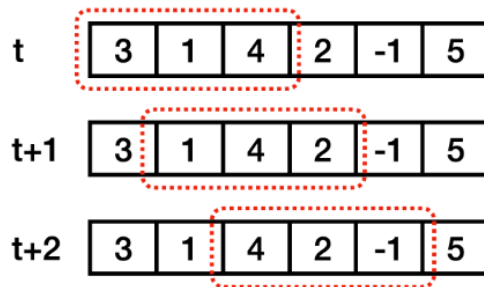
→ [[CLS], 세종, 은, 조선, 의, 몇, 대, 왕이, 야, ?, [SEP], 세종, 은, ... [SEP], [Pad], ... ]  
→ [ 0, 654, 5, 456, 6, 12, 44, 457, 23, 9, 1, 654, 5, ..., 1, 2, ... ]

질문

문서

※ 질문+문서의 길이가 사전 정의된 최대 입력길이를 초과하는 경우?

- Slicing Window 기법을 이용하여 데이터 분할



- Example

- 질문, 문서 등 모두 원문 그대로 Key-Value 쌍으로 존재

- Feature

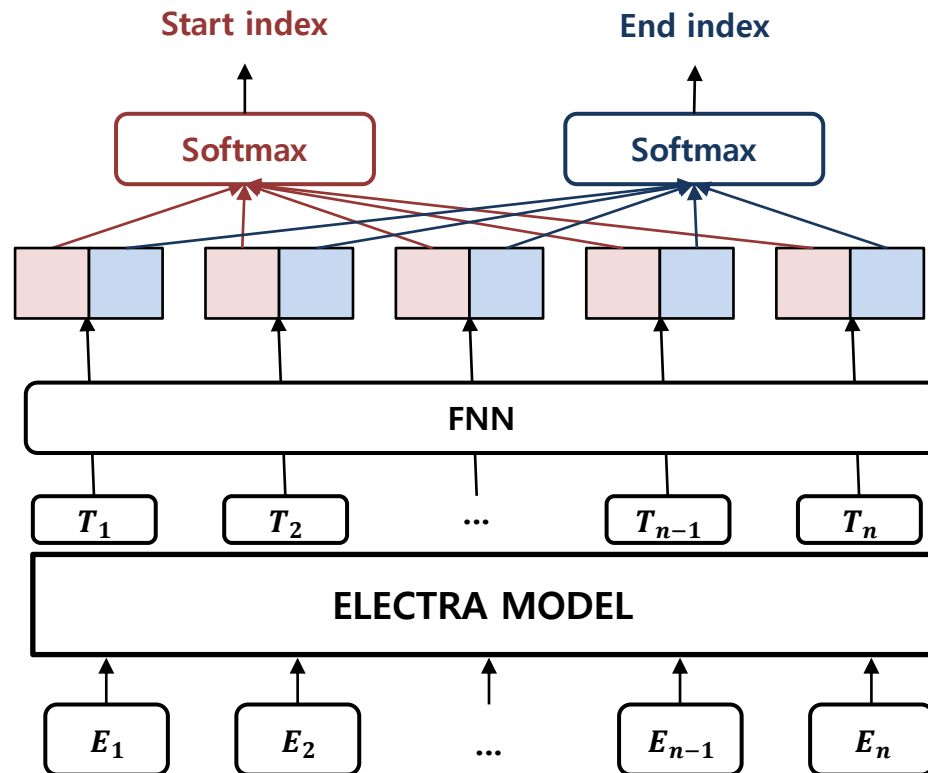
- 질문, 문서, 정답 등 Wordpiece 단위로 표현
- Slicing Window 기법을 통해 모델 입력에 적합한 형태로 저장
- Example에 대응하는 ID와 각 Feature에 대응하는 고유 ID 저장

- Dataset

- Feature와 1:1 대응되며, 인덱스 리스트가 Tensor 형태로 저장 (모델 입력)



# 실습: MRC



## 실습: MRC

## 라이브러리 설치

```
!pip install tokenizers
!pip install transformers==2.11.0
```

```
Collecting tokenizers  
  Downloading https://files.pythonhosted.org/packages/  
    [██████████] | 3.3MB 8.7MB/s  
Installing collected packages: tokenizers  
Successfully installed tokenizers-0.10.3  
Collecting transformers==2.11.0  
  Downloading https://files.pythonhosted.org/packages/  
    [██████████] | 675kB 8.7MB/s  
Requirement already satisfied: tqdm>=4.27 in /usr/local
```

## 라이브러리 import

```
from transformers import ElectraTokenizer, ElectraConfig
from transformers import ElectraModel, ElectraPreTrainedModel
from transformers import (
    squad_convert_examples_to_features
```



# 실습: MRC

## 모델 설계

```
class ElectraForQuestionAnswering(ElectraPreTrainedModel):
    def __init__(self, config):
        super(ElectraForQuestionAnswering, self).__init__(config)

        # 분류해야할 라벨 수 (start / end)
        self.num_labels = config.num_labels
        # ELECTRA 모델 선언
        self.electra = ElectraModel(config)
        # 최종 출력
        self.qa_outputs = nn.Linear(config.hidden_size, config.num_labels)

    def forward(self, input_ids=None, attention_mask=None, token_type_ids=None,
                outputs = self.electra(
                    input_ids=input_ids,
                    attention_mask=attention_mask,
                    token_type_ids=token_type_ids,
                    position_ids=position_ids
                ))
        sequence_output = outputs[0]
        # sequence_output : [batch, max_length, hidden_size]

        logits = self.qa_outputs(sequence_output)
        # logits : [batch, max_length, 2]

        start_logits, end_logits = logits.split(1, dim=-1)
        # start_logits : [batch, max_length, 1]
        # end_logits : [batch, max_length, 1]

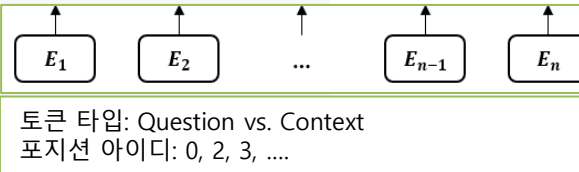
        start_logits = start_logits.squeeze(-1)
        end_logits = end_logits.squeeze(-1)
        # start_logits : [batch, max_length]
        # end_logits : [batch, max_length]

        return start_logits, end_logits
```

ELECTRA 언어모델 사용

ELECTRA 사전학습 모델  
생성자 오버라이딩

마지막 레이어 출력



$(\text{max\_length}, \text{hidden\_size}) * (\text{hidden\_size}, \text{num\_lables})$   
 $= (\text{max\_length}, \text{num\_lables})$

(크기, 기준차원): 마지막 차원의 크기를 1로 만들기

불필요한 마지막 차원을 제거



# 실습: MRC

## 데이터 읽기

```
def load_dataset(config, tokenizer, evaluate=False, output_examples=False):  
    processor = SquadV1Processor()
```

```
# 실행 모드에 따라 데이터를 load  
# 학습 시, 사전 부착되어있는 정답까지 Load  
# 평가 시, 정답 Load X
```

```
    if evaluate:  
        examples = processor.get_dev_examples(config["data_dir"],  
                                              filename=config["predict_file"])  
    else:  
        examples = processor.get_train_examples(config["data_dir"],  
                                              filename=config["train_file"])
```

```
    features, dataset = squad_convert_examples_to_features(  
        examples=examples,  
        tokenizer=tokenizer,  
        max_seq_length=config["max_seq_length"],  
        doc_stride=config["doc_stride"],  
        max_query_length=config["max_query_length"],  
        is_training=not evaluate,  
        return_dataset="pt"
```

```
    )  
    if output_examples:  
        return dataset, examples, features  
    return dataset
```

```
from transformers.data.processors.squad import SquadResult, SquadV1Processor  
from transformers.data.metrics.squad_metrics import (  
    compute_predictions_logits,  
    squad_evaluate,  
    SQUAD 형식의 데이터를 처리하는 클래스
```

512 토큰보다 길면  
얼마나 옆으로 이동  
하면서 겹칠 것인지?

- Feature: dictionary type
- Dataset: list type

테스트 모드

```
example : 최대 길이 상관 없이 원문 그대로 저장되어있는 데이터 (질문, 문서, 정답 위치 (어절 기준))  
: context: "세종대왕은 조선의 4대 왕으로 ~~~"  
  
feature : 최대 길이에 맞춰 분할되어있는 원문 + 인덱스 데이터 (질문 + slice 문서, 정답 위치 (WP 기준), 위치 offset)  
: [세종, 대, 왕은, 조], ~ [1, 6, 3, 4, 34]  
: [대, 왕은, 조, 선의], [왕은, 조, 선의, 4]  
  
dataset : 최대 길이에 맞춰 분할되어있는 tensor dataset (질문 + slice 문서 (Tensor), 정답 위치)  
: [42, 6, 23, 5], [123, 356, 234, 4, 12] ~~~
```

# example : 최대 길이 상관 없이 원문 그대로 저장 되어있는 데이터 (질문, 문서, 정답 위치 (어절 기준))

# feature : 최대 길이에 맞춰 분할 되어있는 원문 + 인덱스 데이터 (질문 + slice 문서, 정답 위치 (WP 기준), 위치 offset)

# dataset : 최대 길이에 맞춰 분할 되어있는 tensor dataset (질문 + slice 문서 (Tensor), 정답 위치)



# 실습: MRC

## 가중치 초기화

# 모델의 가중치를 Load하는 부분






```
def _create_model(config):
    print("Create Model with Pretrained Parameters...")
    electra_config = ElectraConfig.from_pretrained(
        config["init_weight"] if config["mode"] == "train" else os.path.join(config["output_dir"],
                                                                              "checkpoint-{}".format(config["checkpoint"]))
    )
    electra_tokenizer = ElectraTokenizer.from_pretrained(
        config["init_weight"] if config["mode"] == "train" else os.path.join(config["output_dir"],
                                                                              "checkpoint-{}".format(config["checkpoint"]))
        do_lower_case=config["do_lower_case"],
    )
    model = ElectraForQuestionAnswering.from_pretrained(
        config["init_weight"] if config["mode"] == "train" else os.path.join(config["output_dir"],
                                                                              "checkpoint-{}".format(config["checkpoint"]))
        config=electra_config
    )

    model.cuda()

    return config, model, electra_tokenizer
```

ELECTRA 사전 학습된 모델  
가져오는 방법 그대로 사용

내 드라이브 > ... > output > checkpoint-3 ▾

이름 ↑	소유자
 config.json	나
 pytorch_model.bin	나
 special_tokens_map.json	나
 tokenizer_config.json	나
 vocab.txt	나





# 실습: MRC

## Train

```
def train(config, model, tokenizer):
    # 데이터 읽기
    train_dataset = load_dataset(config, tokenizer, evaluate=False, output_examples=False)

    # TensorDataset/DataLoader를 통해 배치(batch) 단위로 데이터를 나누고 셔플(shuffle)
    train_dataloader = DataLoader(train_dataset, shuffle = True, batch_size=config["batch_size"])

    # optimizer 선언
    optimizer = torch.optim.Adam(model.parameters(), lr=config["learning_rate"])

    global_step = 0

    # 크로스엔트로피 손실 함수
    loss_func = nn.CrossEntropyLoss()

    best_f1 = 0
    for epoch in range(config["num_epochs"]):
        for step, batch in enumerate(train_dataloader):
            # 학습 모드 셋팅
            model.train()

            # .cuda()를 통해 데이터를 메모리에 업로드
            batch = tuple(t.cuda() for t in batch)
            input_ids, attention_mask, token_type_ids, start_positions, end_positions = batch[:5]

            start_hypothesis, end_hypothesis = model(input_ids, attention_mask, token_type_ids)
            # start_hypothesis : [batch, max_length]
            # end_hypothesis : [batch, max_length]

            # Loss 계산 및 역전파
            start_loss = loss_func(start_hypothesis, start_positions)
            end_loss = loss_func(end_hypothesis, end_positions)

            loss = start_loss + end_loss

            loss.backward()
```

Batch size: 3

```
{'input_ids': tensor([[2, 29347, 30191, 237, ..., 0, 0, 0],
                      [2, 9948, 41, 94, ..., 0, 0, 0],
                      [2, 456, 25, 89, ..., 0, 0, 0]]),
  'attention_mask': tensor([[1, 1, 1, 1, 1, ..., 0, 0, 0],
                             [1, 1, 1, 1, 1, ..., 0, 0, 0],
                             [1, 1, 1, 1, 1, ..., 0, 0, 0]]),
  'token_type_ids': tensor([[0, 0, 0, 1, 1, ..., 1, 0, 0],
                             [0, 0, 0, 0, 1, ..., 1, 0, 0],
                             [0, 0, 1, 0, 0, ..., 1, 0, 0]]),
  'start_positions': tensor([26, 52, 78]),
  'end_positions' : tensor([27, 56, 81])}
```

두 개의 손실을 최소화하도록 설정



# 실습: MRC

## Test

```
def test(config, model, tokenizer, global_step):
    # 데이터 읽기
    dataset, examples, features = load_dataset(config, tokenizer, evaluate=True, output_examples=True)

    # TensorDataset/DataLoader를 통해 배치(batch) 단위로 데이터를 나누고 셔플(shuffle)
    eval_dataloader = DataLoader(dataset, shuffle=True, batch_size=config["batch_size"])

    # 모델 출력 결과를 저장하기 위한 list
    all_results = []

    for step, batch in enumerate(tqdm(eval_dataloader)):
        # 평가 모드 세팅
        model.eval()

        # .cuda()를 통해 데이터를 메모리에 업로드
        batch = tuple(t.cuda() for t in batch)
        with torch.no_grad():
            input_ids, attention_mask, token_type_ids = batch[:3]

            feature_unique_ids = batch[3]
            hypothesis = model(input_ids, attention_mask, token_type_ids)

            for i, feature_unique_id in enumerate(feature_unique_ids):
                eval_feature = features[feature_unique_id.item()]

                # unique_id : dataset과 feature가 공유하는 id
                unique_id = int(eval_feature.unique_id)

                # hypothesis : [s_hypothesis : [batch, seq_len], e_hypothesis : [batch, seq_len]]
                output = [tensor2list(output[i]) for output in hypothesis]
                start_hypothesis, end_hypothesis = output
                result = SquadResult(unique_id, start_hypothesis, end_hypothesis)

            all_results.append(result)
```

Batch size: 3

```
{'input_ids': tensor([[2, 29347, 30191, 237, ..., 0, 0, 0],
                    [2, 9948, 41, 94, ..., 0, 0, 0],
                    [2, 456, 25, 89, ..., 0, 0, 0]]),
  'attention_mask': tensor([[1, 1, 1, 1, 1, ..., 0, 0, 0],
                            [1, 1, 1, 1, 1, ..., 0, 0, 0],
                            [1, 1, 1, 1, 1, ..., 0, 0, 0]]),
  'token_type_ids': tensor([[0, 0, 0, 1, 1, ..., 1, 0, 0],
                            [0, 0, 0, 0, 1, ..., 1, 0, 0],
                            [0, 0, 1, 0, 0, ..., 1, 0, 0]]),
  'example_index': [0, 1, 2]}
```

배치 내 몇 번째 분할 문서  
인지를 나타내는 인덱스  
(example\_index)

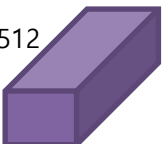
start\_logits : [batch, max\_length]  
end\_logits : [batch, max\_length]

배치 개수 만큼  
feature 가져오기

512 단위로 잘라진  
문단의 고유 번호 (전  
체 데이터셋에 대한  
고유 번호)

(2,3,512) → [... 512 ...] [... 512 ...], [... 512 ...],  
[... 512 ...] [... 512 ...], [... 512 ...]]

512  
2  
3  
2,batch,seseq\_len




# 실습: MRC

## Main

```
if __name__ == "__main__":
    output_dir = os.path.join(root_dir, "output")
    data_dir = os.path.join(root_dir, "data")
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    config = {
        "mode": "train",
        "data_dir": data_dir,
        "output_dir": output_dir,
        "train_file": "refine_hp_train.json",
        "predict_file": "refine_hp_test.json",
        "init_weight": "monologg/koelectra-base-v2-discriminator",
        "do_lower_case": False,
        "checkpoint": 6000,
        "batch_size": 8,
        "num_epochs": 5,
        "max_seq_length": 512,
        "max_query_length": 64,
        "max_answer_length": 30,
        "n_best_size": 20,
        "learning_rate": 5e-5,
        "doc_stride": 128,
    }
    config, model, tokenizer = _create_model(config)
    if config["mode"] == "train":
        train(config, model, tokenizer)
    elif config["mode"] == "test":
        results = test(config, model, tokenizer, config["checkpoint"])
        for key in sorted(results.keys()):
            print(" {} = {}".format(key, str(results[key])))
```

학습에 5시간  
이상 소요

#####

Context : 2008년 2월 25일 이명박은 취임식과 함께 업무 수행을 시작

Question : 이명박이 취임식과 함께 업무수행을 시작한 해는?

Answer Span : 2008년

Predict Span : 2008년

#####

Context : 부모는 사건 이후 일을 그만두고 딸의 치료에만 매달렸다.

Question : 조두순 사건에서 보험사는 부모에게 얼마를 지급했는가?

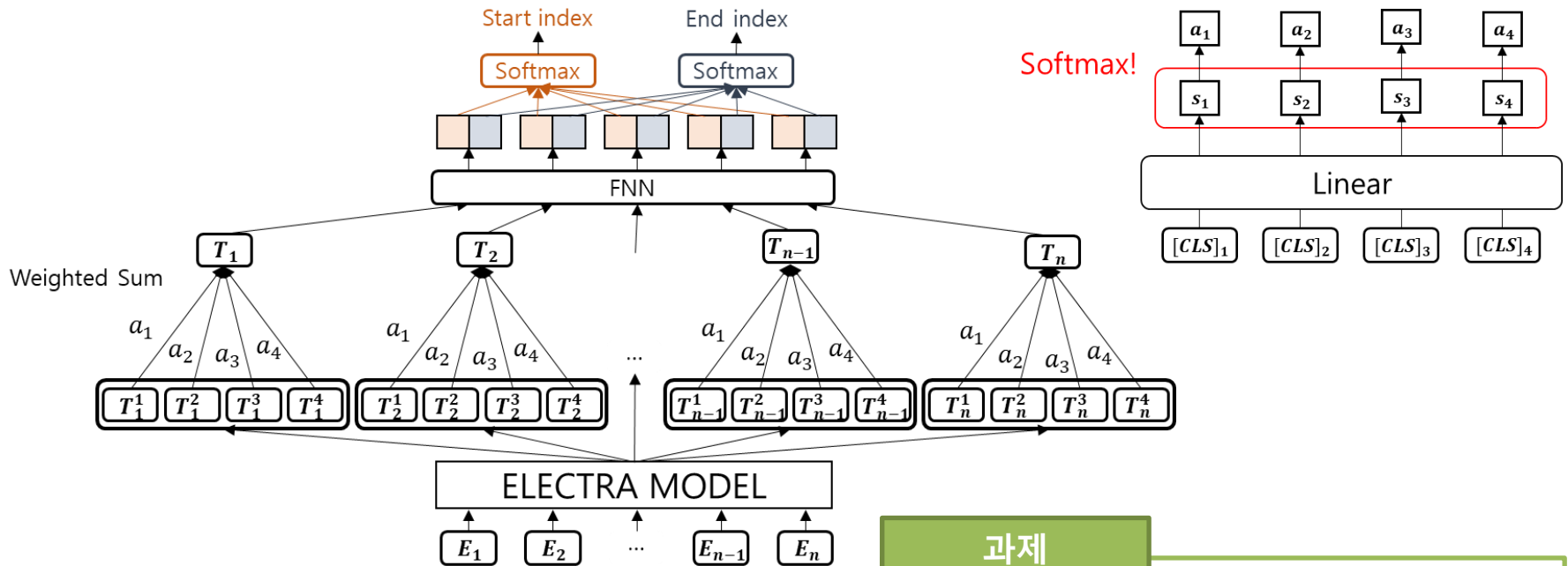
Answer Span : 4000만 원

Predict Span : 4000만 원



# 실습: MRC

- Layer 별 중요도 반영할 수 있는 모델 구현



## 과제

- 모델 구현: 10점 (ipynb에 개선 포인트 기술, 기존 대비 성능 개선 결과 제시)



---

# Document Reader

---

Practical Exercise

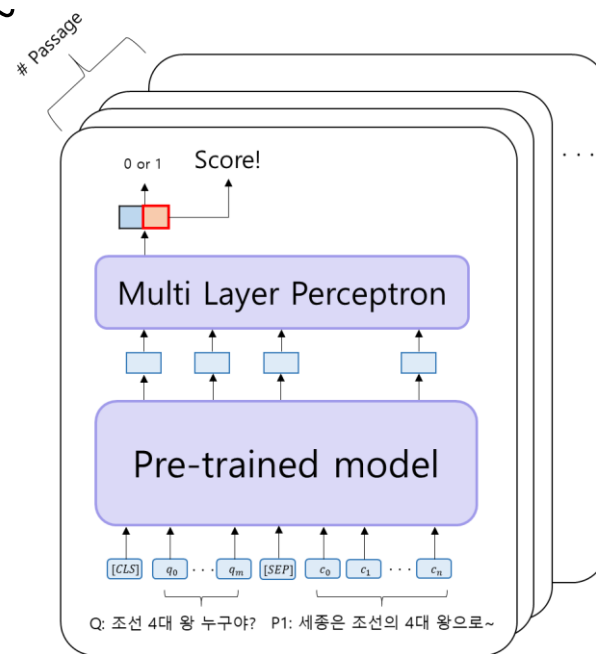
# 실습

예제 코드 다운로드:  
<https://github.com/KUNLP/KTAI-Practice>

- 기계독해를 위한 Passage 재순위화 모델
  - 질문 : 조선의 4대 왕이 누구야?
  - 후보 구절 1 : 세종은 조선의 4대 왕으로 ~~
  - 후보 구절 2 : 세종은 1392년에 태어나 ~

...

모델 구조



# 실습: Passage Reranking

## 모델 설계

```
class RerankingModel(ElectraPreTrainedModel):
    def __init__(self, config):
        super(RerankingModel, self).__init__(config)

        # 분류해야할 라벨 수 (0 / 1)
        self.num_labels = config.num_labels

        # ELECTRA 모델 선언
        self.electra = ElectraModel(config)

        # 최종 출력
        self.output_layer_1 = nn.Linear(config.hidden_size, config.hidden_size//2)
        self.output_layer_2 = nn.Linear(config.hidden_size//2, config.num_labels)

    def forward(self, input_ids=None, token_type_ids=None):
        # input_ids : [batch, max_length] => [[1, 5, 3, 4, 78, 345, 234, 67, ... ], [...]]
        # token_type_ids : [batch, max_length] => [[0, 0, 0, ..., 1, 1, 1, ..., 0, 0, 0], [...]]

        # ELECTRA output
        outputs = self.electra(input_ids=input_ids,
                                token_type_ids=token_type_ids)
        # outputs : [1, batch, max_length, hidden_size]

        sequence_output = outputs[0]
        # sequence_output : [batch, max_length, hidden_size]

        # CLS 벡터 추출
        cls_output = sequence_output[:, 0, :]
        # cls_output : [batch, hidden]

        dense_output = self.output_layer_1(cls_output)
        hypothesis = self.output_layer_2(dense_output)
        # hypothesis : [batch, 2]

        return hypothesis
```

last\_hidden\_state ( torch.FloatTensor of shape (batch\_size, sequence\_length, hidden\_size) ):

Sequence of hidden-states at the output of the last layer of the model.



# 실습: Passage Reranking

## 학습 준비

input\_ids,  
attention\_mask,  
overflowing\_tokens 등  
다양한 정보가 저장

Question과 Passage로 하  
나의 입력 Sequence 구성

```
def load_dataset(config, tokenizer, flag):
    file_dir = config['train_file'] if flag else config['test_file']
    data_file = open(os.path.join(config["data_dir"], file_dir), 'r', encoding='utf8').readlines()

    # 데이터를 저장하기 위한 리스트 생성
    q_ids, all_input_ids, all_token_type_ids, all_question_masks, #
    label_ids = [], [], [], [], []
    for line in tqdm(data_file):
        q_id, query, passage, label = line.strip().split('##t')
        query_sequence = tokenizer.tokenize(query)
        context_sequence = tokenizer.tokenize(passage)
        encoded_dict = tokenizer.encode_plus(query_sequence, context_sequence, padding='max_length')

        q_ids.append(int(q_id))
        all_input_ids.append(encoded_dict["input_ids"][:config["max_seq_length"]])
        all_token_type_ids.append(encoded_dict["token_type_ids"][:config["max_seq_length"]])
        label_ids.append(int(label))

    # 전체 데이터를 저장하고 있는 리스트를 텐서 형태로 변환
    q_ids = torch.tensor(q_ids, dtype=torch.long)
    all_input_ids = torch.tensor(all_input_ids, dtype=torch.long)
    all_token_type_ids = torch.tensor(all_token_type_ids, dtype=torch.long)
    label_ids = torch.tensor(label_ids, dtype=torch.long)

    return q_ids, all_input_ids, all_token_type_ids, label_ids
```





# 실습: Passage Reranking

## 학습

```
def train(config, model, tokenizer):
    # 데이터 읽기
    _, all_input_ids, all_token_type_ids, all_labels = load_dataset(config, tokenizer, flag=True)

    # TensorDataset/DataLoader를 통해 배치(batch) 단위로 데이터를 나누고 셔플(shuffle)
    train_dataset = TensorDataset(all_input_ids, all_token_type_ids, all_labels)
    train_dataloader = DataLoader(train_dataset, shuffle = True, batch_size=config["batch_size"])

    # optimizer 선언
    optimizer = torch.optim.Adam(model.parameters(), lr=config["learning_rate"])
    global_step = 0

    # 크로스엔트로피 손실 함수
    criterion = nn.CrossEntropyLoss()

    best_score = 0
    for epoch in range(config["num_epochs"]):
        for step, batch in enumerate(train_dataloader):
            # 학습 모드 셋팅
            model.train()

            # .cuda()를 통해 데이터를 메모리에 업로드
            batch = tuple(t.cuda() for t in batch)
            input_ids, token_type_ids, label_ids = batch

            hypothesis = model(input_ids, token_type_ids)
            # hypothesis : [batch, 2]

            labels = label_ids.view(-1,)

            # Loss 계산 및 역전파
            loss = criterion(hypothesis, labels)
            loss.backward()

            optimizer.step()
            model.zero_grad()

            # 중간 결과 출력
            if (global_step + 1) % 50 == 0:
                print("{} step Current Loss : {}".format(global_step + 1, loss.item()))

            # 특정 checkpoint 마다 저장 및 평가
            if (global_step + 1) % 2000 == 0:
                top_N_score = do_test(config, model, tokenizer)

                # 모델 저장을 위한 부분
                output_dir = os.path.join(config["output_dir"], "checkpoint-{}".format(global_step+1))

                # 모델 및 vocab 저장
                model.save_pretrained(output_dir)
                tokenizer.save_pretrained(output_dir)

                if best_score < top_N_score[0]:
                    best_score = top_N_score[0]
                    torch.save(model.state_dict(), os.path.join(output_dir, "training_args.bin"))
                    print("Saving model checkpoint to ", output_dir)

                global_step += 1
```

학습모드

1. 하이퍼파라미터(config.json) 파일 저장
2. 학습된 ELECTRA 모델 파라미터 저장



# 실습: Passage Reranking

## 평가

```
def do_test(config, model, tokenizer):
    # 데이터 읽기
    all_ids, all_input_ids, all_token_type_ids, all_labels = load_dataset(config, tokenizer, flag=False)

    # TensorDataset/DataLoader를 통해 배치(batch) 단위로 데이터를 나누고 셔플(shuffle)
    test_dataset = TensorDataset(all_ids, all_input_ids, all_token_type_ids, all_labels)
    test_dataloader = DataLoader(test_dataset, shuffle=True, batch_size=config["batch_size"])

    # 모델 출력 결과를 저장하기 위한 dictionary
    all_results = {}

    for step, batch in enumerate(tqdm(test_dataloader)):
        # 평가 모드 세팅
        model.eval()

        # .cuda()를 통해 데이터를 메모리에 업로드
        batch = tuple(t.cuda() for t in batch)
        with torch.no_grad():
            q_ids, input_ids, token_type_ids, labels = batch

            hypothesis = model(input_ids, token_type_ids)

            # Softmax 함수를 통해 입력 데이터에 대한 점수 계산
            probs = tensor2list(F.softmax(hypothesis, -1)[0], 1))

        # 각 데이터에 해당하는 q_id로 결과 저장
        batch_size = hypothesis.size(0)
        q_ids = tensor2list(q_ids)
        labels = tensor2list(labels)
        for i in range(batch_size):
            cur_q_id = q_ids[i]
            if cur_q_id not in all_results.keys():
                all_results[cur_q_id] = []
            all_results[cur_q_id].append((probs[i], labels[i]))
```

평가모드

Passage 점수로 정렬하여 Top N Recall 측정

```
# Recall at K
def measure(all_results):
    n_samples = 0
    correct = [0]*10
    for q_id in all_results.keys():
        prediction = [e[1] for e in sorted(all_results[q_id], key=lambda x: x[0], reverse=True)]
        if 1 in prediction:
            correct[prediction.index(1)]+=1
        n_samples+=1
    top_N_scores = [sum(correct[:e+1])/n_samples for e in range(len(correct))]
    return top_N_scores
```

동일한 q\_id에 대해  
Passage 점수와 정답 라벨  
쌍으로 리스트에 저장



# 실습: Passage Reranking

## Main

```
if __name__ == "__main__":
    output_dir = os.path.join(root_dir, "output")
    data_dir = os.path.join(root_dir, "data")
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    config = {
        "mode": "train",
        "data_dir": data_dir,
        "output_dir": output_dir,
        "train_file": "refine_train_small.txt",
        "test_file": "refine_test.txt",
        "init_weight": "monologg/koelectra-small-v3-discriminator",
        "do_lower_case": False,
        "checkpoint": 6000,
        "batch_size": 16,
        "num_epochs": 3,
        "max_seq_length": 512,
        "learning_rate": 5e-5,
    }
    model, tokenizer = _create_model(config)
    if config["mode"] == "train":
        train(config, model, tokenizer)
    elif config["mode"] == "test":
        results = do_test(config, model, tokenizer)
```

```
100%|██████████| 4807/4807 [00:13<00:00, 359.11it/s]
100%|██████████| 301/301 [00:29<00:00, 10.20it/s]
```

Top 1 Recall Score : 0.944

Top 2 Recall Score : 0.960

Top 3 Recall Score : 0.974

Top 4 Recall Score : 0.974

Top 5 Recall Score : 0.976

Top 6 Recall Score : 0.978

Top 7 Recall Score : 0.980

Top 8 Recall Score : 0.982

Top 9 Recall Score : 0.982

Top 10 Recall Score : 0.982

## 과제

1. 빈칸 채우기: 7점
2. 성능 개선: 10점 (ipynb에 개선 포인트 기술, 기존 대비 성능 개선 결과 제시)



# 질의응답

---

Q&A

Homepage: <http://nlp.konkuk.ac.kr>  
E-mail: [nlpdrkim@konkuk.ac.kr](mailto:nlpdrkim@konkuk.ac.kr)

