KU KONKUK UNIVERSITY

KAIST

개체명 인식
(Named Entity Recognition)

Practical Exercise

자연어처리연구실

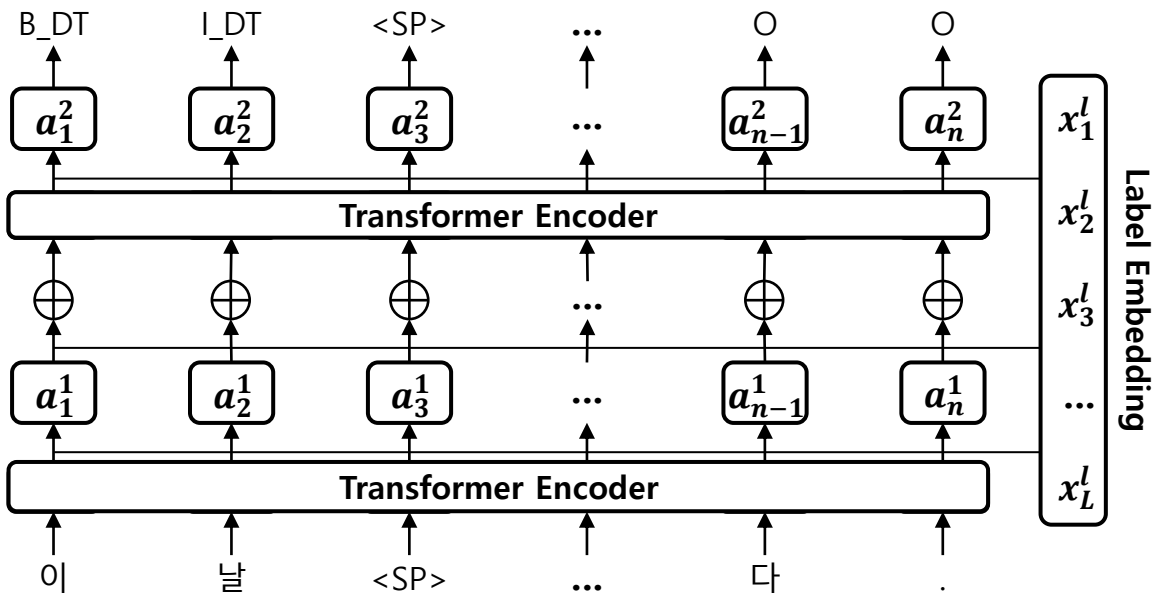
실습: Named Entity Recognition

- Transformer+LAN을 활용한 개체명 인식 코드 r

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

입력	이	날	<SP>	기	자	회	견	에	서	<SP>	임	호	는	<SP>	…
출력	B_DT	I_DT	<SP>	O	O	O	O	O	O	<SP>	B_PS	I_PS	O	<SP>	…

모델 구조



실습: Named Entity Recognition

모델 설계

생성자

```
class TRANSFORMER_LAN(nn.Module):

    def __init__(self, config):
        super(TRANSFORMER_LAN, self).__init__()

        # 전체 character 개수
        self.character_vocab_size = config["character_vocab_size"]

        # character 임베딩 사이즈
        self.embedding_size = config["embedding_size"]

        # 최대 길이 (character 기준)
        self.max_length = config["max_length"]

        # Transformer 히든 사이즈
        self.hidden_size = config["hidden_size"]

        # Transformer head 개수
        self.num_heads = config["num_heads"]

        # Transformer layer 개수
        self.num_layers = config["num_layers"]

        # 분류할 라벨의 개수
        self.num_of_labels = config["num_of_labels"]

        self.dropout = nn.Dropout(config["dropout"])
```

```
# character index를 대응하는 벡터로 치환해주기 위한 임베딩 객체
self.embedding = nn.Embedding(num_embeddings=self.character_vocab_size,
                              embedding_dim=self.embedding_size,
                              padding_idx=0)
# 입력 데이터의 각 character들의 위치 정보를 반영해주기 위한 임베딩 객체
self.position_embedding = nn.Embedding(num_embeddings=self.max_length,
                                       embedding_dim=self.embedding_size)
self.embedding_layer_norm = nn.LayerNorm(normalized_shape=self.embedding_size)

# 분류할 각 라벨에 대응하는 임베딩 벡터들을 갖고 있는 행렬
self.label_embedding = nn.Parameter(torch.randn(size=(self.num_of_labels, self.hidden_size),
                                                dtype=torch.float32,
                                                requires_grad=True))
# Transformer encoder layer
self.first_encoder = nn.TransformerEncoder(encoder_layer=nn.TransformerEncoderLayer(d_model=self.hidden_size,
                                                                                    nhead=self.num_heads,
                                                                                    dim_feedforward=self.hidden_size,
                                                                                    batch_first=True),
                                           num_layers=self.num_layers)

# Transformer encoder layer
self.second_encoder = nn.TransformerEncoder(encoder_layer=nn.TransformerEncoderLayer(d_model=self.hidden_size,
                                                                                     nhead=self.num_heads,
                                                                                     dim_feedforward=self.hidden_size,
                                                                                     batch_first=True),
                                            num_layers=self.num_layers)

self.linear = nn.Linear(in_features=self.hidden_size * 2, out_features=self.hidden_size)

# Multi-head attention layer
self.first_multi_head_attention = Multihead_Attention(hidden_size=self.hidden_size,
                                                      num_heads=self.num_heads,
                                                      dropout_rate=config["dropout"])

# Multi-head attention layer
self.second_multi_head_attention = Multihead_Attention(hidden_size=self.hidden_size,
                                                       num_heads=1,
                                                       dropout_rate=config["dropout"])
```

실습: Named Entity Recognition

모델 설계

멀티헤드어텐션

```
def forward(self, queries, keys, values, last_layer=False):
    # queries : (batch_size, max_length, hidden_size)
    # keys : (batch_size, num_of_labels, hidden_size)
    # values : (batch_size, num_of_labels, hidden_size)

    # Q : (batch_size, max_length, hidden_size)
    # K : (batch_size, num_of_labels, hidden_size)
    # V : (batch_size, num_of_labels, hidden_size)
    Q = self.query_linear(queries)  # (N, T_q, C)
    K = self.key_linear(keys)  # (N, T_q, C)
    V = self.value_linear(values)  # (N, T_q, C)

    # Q_ : (batch_size * num_heads, max_length, hidden_size / num_heads)
    # K_ : (batch_size * num_heads, num_of_labels, hidden_size / num_heads)
    # V_ : (batch_size * num_heads, num_of_labels, hidden_size / num_heads)
    Q_ = torch.cat(torch.chunk(Q, self.num_heads, dim=2), dim=0)
    K_ = torch.cat(torch.chunk(K, self.num_heads, dim=2), dim=0)
    V_ = torch.cat(torch.chunk(V, self.num_heads, dim=2), dim=0)

    # (batch_size * num_heads, hidden_size / num_heads, num_of_labels

    # attention_weights : (batch_size*num_heads, max_length, num_of_labels)
    attention_weights = torch.bmm(Q_, K_.permute(0, 2, 1))
    attention_weights = attention_weights / (K_.size()[-1] ** 0.5)
    if(last_layer == False):
        attention_weights = self.softmax(attention_weights)
```

```
# query_masks : (batch_size, max_length)
query_masks = torch.sign(torch.abs(torch.sum(queries, dim=-1)))
# query_masks : (batch_size * num_heads, max_length)
query_masks = query_masks.repeat(self.num_heads, 1)
# query_masks : (batch_size * num_heads, max_length, num_of_labels)
query_masks = torch.unsqueeze(query_masks,2).repeat(1,1,keys.size()[1])

# attention_weights : (batch_size*num_heads, max_length, num_of_labels)
attention_weights = attention_weights * query_masks
attention_weights = self.dropout(attention_weights)  # (h*N, T_q, T_k)

# attention_outputs : (batch_size*num_heads, max_length,
#                      hidden_size/num_heads)
attention_outputs = torch.bmm(attention_weights, V_)
# attention_outputs : (batch_size, max_length, hidden_size)
attention_outputs = torch.cat(torch.chunk(attention_outputs,
                                          self.num_heads, dim=0),
                              dim=2)

# attention_outputs : (batch_size, max_length, hidden_size)
# attention_weights : (batch_size*num_heads, max_length, num_of_labels)
return attention_outputs, attention_weights
```

실습: Named Entity Recognition

모델 설계

개체명 라벨 분류

```
def forward(self, inputs, positions):
    # inputs : (batch_size, max_length)
    # positions : (batch_size, max_length)
    batch_size = inputs.size()[0]

    # label_embedding : (1, num_of_labels, hidden)
    label_embedding = self.label_embedding.clone().unsqueeze(dim=0)

    # label_embedding : (batch_size, num_of_labels, hidden)
    label_embedding = label_embedding.repeat(batch_size, 1, 1)

    # character_inputs : (batch_size, max_length, embedding_size)
    character_inputs = self.embedding(inputs) + self.position_embedding(positions)
    character_inputs = self.embedding_layer_norm(character_inputs)
    character_inputs = self.dropout(character_inputs)

    # first_encoder_output : (batch_size, max_length, hidden_size)
    first_encoder_output = self.first_encoder(src=character_inputs,
                                              src_key_padding_mask= (inputs == 0))

    # first_attention_outputs : (batch_size, max_length, hidden_size)
    first_attention_outputs, _ = self.first_multi_head_attention(queries=first_encoder_output,
                                                                 keys=label_embedding,
                                                                 values=label_embedding,
                                                                 last_layer=False)

    # middle_outputs : (batch_size, max_length, hidden_size+2)
    middle_outputs = torch.cat(tensors=[first_encoder_output, first_attention_outputs], dim=-1)

    # middle_outputs : (batch_size, max_length, hidden_size)
    middle_outputs = self.linear(middle_outputs)

    # second_encoder_output : (batch_size, max_length, hidden_size)
    second_encoder_output = self.second_encoder(src=middle_outputs, src_key_padding_mask=(inputs == 0))

    # second_attention_weights : (batch_size, max_length, num_of_labels)
    _ , second_attention_weights = self.second_multi_head_attention(queries=second_encoder_output,
                                                                    keys=label_embedding,
                                                                    values=label_embedding,
                                                                    last_layer=True)

    return second_attention_weights
```

실습: Named Entity Recognition

Main

```
if(__name__=="__main__"):
    root_dir = "/gdrive/My Drive/colab/NER/"
    save_dir = os.path.join(root_dir, "save")
    output_dir = os.path.join(root_dir, "output")
    if not os.path.exists(save_dir):
        os.makedirs(save_dir)
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    set_seed(seed=1234)

    config = {"mode": "test",
              "trained_model_name":"epoch_{}.pt".format(18),
              "train_data_path":os.path.join(root_dir, "train_datas.txt"),
              "test_data_path": os.path.join(root_dir, "test_datas.txt"),
              "save_dir_path":save_dir,
              "output_dir_path": output_dir,
              "character_vocab_data_path": os.path.join(root_dir, "character_vocab.txt"),
              "label_vocab_data_path": os.path.join(root_dir, "label_vocab.txt"),
              "character_vocab_size": 2159,
              "embedding_size": 256,
              "hidden_size": 256,
              "max_length": 491,
              "num_heads":4,
              "num_layers": 1,
              "num_of_labels": 15,
              "epoch":20,
              "batch_size":64,
              "dropout":0.3
              }

    if(config["mode"] == "train"):
        train(config)
    else:
        test(config)
```

```
입력 : 0 일 <SP> 유 통 업 계 와 <SP> 정 유 업 계 에 <SP> 따 르 면 <SP> ' 이 마 트 - S K '
출력 : B_DT I_DT <SP> O O O O O <SP> O O O O O <SP> O O O <SP> O O O O O B_OG I_OG O <SP> O
정답 : B_DT I_DT <SP> O O O O O <SP> O O O O O <SP> O O O <SP> O O O O O O O O O <SP> O O O <S

입력 : 김 씨 는 <SP> 이 후 에 도 <SP> 기 초 생 활 수 급 자 나 <SP> 차 상 위 의 <SP> 등 록 과
출력 : B_PS I_PS O <SP> O O O O <SP> O O O O O O O O O <SP> O O O O <SP> O O O <SP> O O O <SP>
정답 : B_PS O O <SP> O O O O <SP> O O O O O O O O O <SP> O O O O <SP> O O O <SP> O O O <SP> O

입력 : A 사 에 <SP> 따 르 면 <SP> 두 <SP> 회 사 는 <SP> 실 제 <SP> 발 생 하 ㄴ <SP> 비 용 을
출력 : O O O <SP> O O O <SP> O <SP> O O O <SP> O O <SP> O O O O <SP> O O O <SP> O O O <SP> (
정답 : O O O <SP> O O O <SP> O <SP> O O O <SP> O O <SP> O O O O <SP> O O O <SP> O O O <SP> (

입력 : 대 우 조 선 해 양 과 <SP> 만 도 , <SP> 현 대 산 업 개 발 <SP> 등 이 <SP> 전 반 적 으
출력 : B_OG I_OG I_OG I_OG I_OG I_OG O <SP> O O O <SP> B_OG I_OG I_OG I_OG O O <SP> O O <SP>
정답 : B_OG I_OG I_OG I_OG I_OG I_OG O <SP> B_OG I_OG O <SP> B_OG I_OG I_OG I_OG I_OG I_OG <

입력 : O O 일 <SP> 삼 성 전 자 에 <SP> 따 르 면 <SP> 갤 럭 시 S O 는 <SP> 독 일 <SP> 출 시 <
출력 : B_DT I_DT I_DT <SP> B_OG I_OG I_OG I_OG O <SP> O O O <SP> O O O I_OG O O <SP> B_LC I_
정답 : B_DT I_DT I_DT <SP> B_OG I_OG I_OG I_OG O <SP> O O O <SP> O O O O O O <SP> B_LC I_LC

입력 : 토 요 ( 당 일 ) O 차 <SP> 낙 동 정 맥 <SP> 종 주 대 원 <SP> 모 집
출력 : B_OG I_OG O B_DT I_DT O O O <SP> B_OG I_OG I_OG I_OG <SP> B_OG I_OG I_OG O <SP> O O
정답 : B_DT I_DT O B_DT I_DT O O O <SP> B_LC I_LC I_LC I_LC <SP> O O O O <SP> O O
```

과제

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

Edited by Harksoo Kim

KU KONKUK UNIVERSITY

KAIST

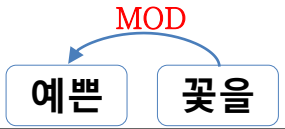
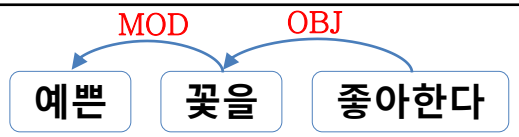
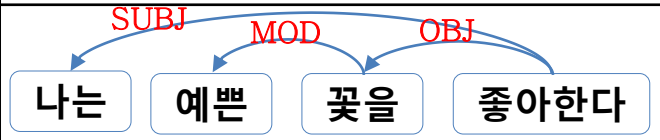
구문 분석 I
(Shift Reduce Parsing)

Practical Exercise

자연어처리연구실

확인 문제: Shift Reduce Parsing

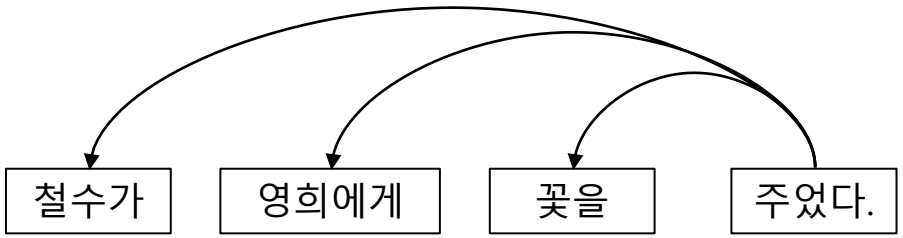
[입력 문장] 나는 예쁜 꽃을 좋아한다.

단계	스택(Stack)	큐(Queue)	분류	의존구조 트리
1	나는	예쁜 꽃을 좋아한다	Shift	
2	나는 예쁜	꽃을 좋아한다	Reduce	
3	나는	꽃을 좋아한다	Shift	
4	나는 꽃을	좋아한다	Reduce	
5	나는	좋아한다	Reduce	

실습: Shift Reduce Parsing

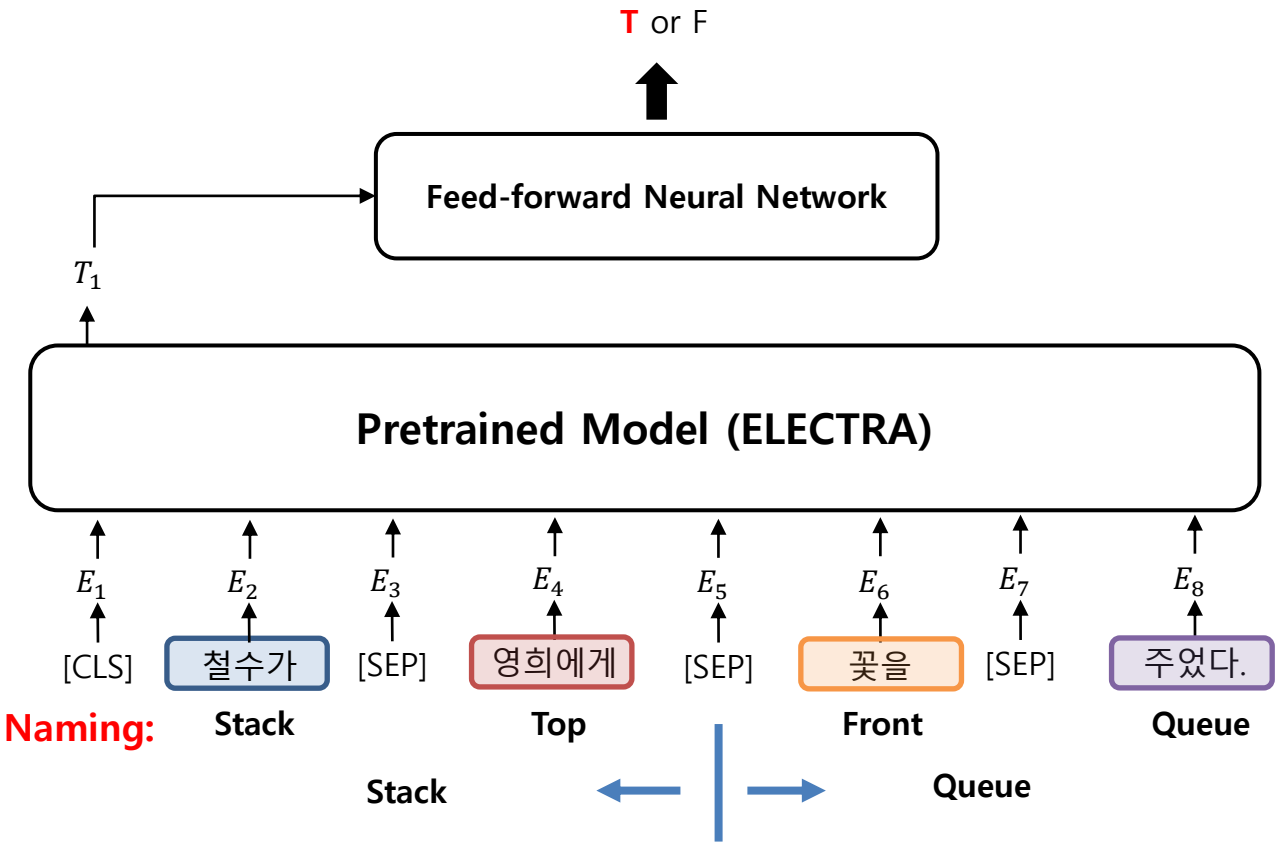
- ELECTRA와 Shift Reduce Parsing을 이용한 구문 분석 (Syntactic Parser) 코드 리뷰

입력				출력
Stack	Top	Front	Queue	
-	철수가	영희에게	꽃을 주었다.	0
철수가	영희에게	꽃을	주었다.	0
철수가 영희에게	꽃을	주었다.	-	1
철수가	영희에게	주었다.	-	1
-	철수가	주었다.	-	1



실습: Shift Reduce Parsing

모델 구조



실습: Shift Reduce Parsing

모델 설계

BERT 기반 Sentiment Analysis와 유사

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

```python
import torch
import torch.nn as nn

from transformers import ElectraPreTrainedModel, ElectraModel

class ElectraForSequenceClassification(ElectraPreTrainedModel):

    def __init__(self, config):
        super(ElectraForSequenceClassification, self).__init__(config)

        # Electra 모델
        self.electra = ElectraModel(config)

        # 분류할 라벨의 개수
        self.num_labels = config.num_labels

        self.hidden_size = config.hidden_size
        self.output_layer = nn.Linear(in_features=self.hidden_size, out_features=self.num_labels)

    def forward(self, input_ids, attention_mask, token_type_ids):
        outputs = self.electra(input_ids=input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids)

        # electra_output : (batch_size, max_length, hidden_size)
        electra_output = outputs[0]

        # cls_vector : (batch_size, hidden_size)
        cls_vector = electra_output[:, 0, :]

        # outputs : (batch_size, num_labels)
        outputs = self.output_layer(cls_vector)

        return outputs
```

ELECTRA 언어모델 사용

ELECTRA 사전학습 모델 생성자 오버라이딩

내가 추가한 config

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: Shift Reduce Parsing

학습 준비

②

```
인천지역에 11층 이상#%#고층#%#건축물은#%#아파트 3887개동과 일반 건축물 등 총 4064개동이다.#%#1
```

①

```
label2idx = {"1":1, "0":0}
idx2label = {1:'1', 0:'0'}
def read_data(file_path):
    with open(file_path, "r", encoding="utf8") as infile:
        datas = []
        for line in infile:
            # 입력 데이터를 #%#을 기준으로 분리
            pieces = line.strip().split("#%#")
            stack, top, front, queue, label = pieces[0], pieces[1], pieces[2], pieces[3], pieces[4]
            datas.append((stack, top, front, queue, label))
    return datas
```

```
def convert_data2feature(datas, max_length, tokenizer, label2idx):
    input_ids_features, attention_mask_features, token_type_ids_features, label_id_features = [], [], [], []

    for stack, stack_top, queue_front, queue, label in tqdm(datas, desc="convert_data2feature"):
        # Electra tokenizer를 사용하여 입력 문장을 word piece 단위로 분리
        tokenized_stack = tokenizer.tokenize(stack)
        tokenized_stack_top = tokenizer.tokenize(stack_top)
        tokenized_queue_front = tokenizer.tokenize(queue_front)
        tokenized_queue = tokenizer.tokenize(queue)

        # CLS, SEP 토큰 추가
        tokens = [tokenizer.cls_token]
        tokens += tokenized_stack
        tokens += [tokenizer.sep_token]
        tokens += tokenized_stack_top
        tokens += [tokenizer.sep_token]
        tokens += tokenized_queue_front
        tokens += [tokenizer.sep_token]
        tokens += tokenized_queue
        tokens += [tokenizer.sep_token]

        # word piece들을 대응하는 index로 치환
        input_ids = tokenizer.convert_tokens_to_ids(tokens)[:max_length]
        # padding을 제외한 실제 데이터 정보를 반영해주기 위한 attention mask
        attention_mask = [1] * len(input_ids)
        # 각 문장을 구분하기 위한 정보를 반영해주기 [illegible] token_type
        token_type_ids = [0] * len(input_ids)

        # padding 생성
        padding = [tokenizer._convert_token_to_id(tokenizer.pad_token)] * (max_length - len(input_ids))
        input_ids += padding
        attention_mask += padding
        token_type_ids += padding
        assert max_length == len(input_ids) == len(attention_mask) == len(token_type_ids)
```

Attention을 계산할 범위 지정

```
        label_id = label2idx[label]

        # 변환한 데이터를 각 리스트에 저장
        input_ids_features.append(input_ids)
        attention_mask_features.append(attention_mask)
        token_type_ids_features.append(token_type_ids)
        label_id_features.append(label_id)

    # 변환한 데이터를 Tensor 객체에 담아 반환
    input_ids_features = torch.tensor(input_ids_features, dtype=torch.long)
    attention_mask_features = torch.tensor(attention_mask_features, dtype=torch.long)
    token_type_ids_features = torch.tensor(token_type_ids_features, dtype=torch.long)
    label_id_features = torch.tensor(label_id_features, dtype=torch.long)

    return input_ids_features, attention_mask_features, token_type_ids_features, label_id_features
```

계산된 길이가 맞는지 검사:
assert [조건], [오류메시지]

실습: Shift Reduce Parsing

학습

```
def train(config):
    # Electra tokenizer 객체 생성
    electra_tokenizer = ElectraTokenizer.from_pretrained(config['pretrained_model_name_or_path'])

    # 학습 및 평가 데이터 읽기
    train_datas = read_data(config["train_data_path"])
    test_datas = read_data(config["test_data_path"])

    # 입력 데이터 전처리
    train_input_ids_features, train_attention_mask_features, train_token_type_ids_features, train_label_id_features = \
        convert_data2feature(train_datas, config["max_length"], electra_tokenizer, label2idx)
    test_input_ids_features, test_attention_mask_features, test_token_type_ids_features, test_label_id_features = \
        convert_data2feature(test_datas, config["max_length"], electra_tokenizer, label2idx)

    # 학습 데이터를 batch 단위로 추출하기 위한 DataLoader 객체 생성
    train_features = TensorDataset(train_input_ids_features, train_attention_mask_features, train_token_type_ids_features, train_label_id_features)
    train_dataloader = DataLoader(train_features, sampler=RandomSampler(train_features), batch_size=config["batch_size"])

    # 평가 데이터를 batch 단위로 추출하기 위한 DataLoader 객체 생성
    test_features = TensorDataset(test_input_ids_features, test_attention_mask_features, test_token_type_ids_features, test_label_id_features)
    test_dataloader = DataLoader(test_features, sampler=SequentialSampler(test_features), batch_size=config["batch_size"])

    # 사전 학습된 Electra 모델 파일로부터 가중치 불러옴
    electra_config = ElectraConfig.from_pretrained(config['pretrained_model_name_or_path'])
    model = ElectraForSequenceClassification.from_pretrained(config['pretrained_model_name_or_path'], config=electra_config).cuda()

    # loss를 계산하기 위한 함수
    loss_func = nn.CrossEntropyLoss()

    # 모델 학습을 위한 optimizer
    optimizer = optim.Adam(model.parameters(), lr=5e-5)

    # 모델의 정확도를 저장하기 위한 변수
    max_accuracy = 0
```

실습: Shift Reduce Parsing

학습

```
for epoch in range(config["epoch"]):
    model.train()

    total_loss = []
    for step, batch in enumerate(train_dataloader):
        batch = tuple(t.cuda() for t in batch)
        input_ids, attention_mask, token_type_ids, label_id = batch

        # 역전파 단계를 실행하기 전에 변화도를 0으로 변경
        optimizer.zero_grad()

        # 모델 예측 결과
        hypothesis = model(input_ids, attention_mask, token_type_ids)

        # loss 계산
        loss = loss_func(hypothesis, label_id)

        # loss 값으로부터 모델 내부 각 매개변수에 대하여 gradient 계산
        loss.backward()
        # 모델 내부 각 매개변수 가중치 갱신
        optimizer.step()

        if (global_step + 1) % 10 == 0:
            print("Current {} Step Loss : {}".format(global_step+1, loss))
        if (global_step+1) % 200 == 0:
            electra_config.save_pretrained(save_directory=config["output_dir_path"])
            model.save_pretrained(save_directory=config["output_dir_path"])
            max_accuracy = evaluate(model, electra_tokenizer, test_dataloader, step, max_accuracy)
        global_step += 1
```

학습모드

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

실습: Shift Reduce Parsing

평가

```python
def evaluate(model, tokenizer, test_dataloader=None, global_step=0, max_accuracy=0):
    model.eval()

    # 모델의 출력 결과와 실제 정답값을 담을 리스트
    total_hypothesis, total_labels = [], []
    for idx, batch in enumerate(test_dataloader):
        batch = tuple(t.cuda() for t in batch)
        input_ids, attention_mask, token_type_ids, label_id = batch

        with torch.no_grad():
            # 모델 예측 결과
            hypothesis = model(input_ids, attention_mask, token_type_ids)
            # 모델의 출력값에 softmax와 argmax 함수를 적용
            hypothesis = torch.argmax(hypothesis, dim=-1)

        # Tensor를 리스트로 변경
        hypothesis = hypothesis.cpu().detach().numpy().tolist()
        label_id = label_id.cpu().detach().numpy().tolist()

        total_hypothesis += hypothesis
        total_labels += label_id
    # 정확도 계산
    accuracy = accuracy_score(total_labels, total_hypothesis)
    print("Accuracy : {}".format(accuracy))

    return accuracy
```

맨 마지막 차원을 기준으로

dim (*int*) – the dimension to reduce. If `None`, the argmax of the flattened input is returned.

```
>>> a = torch.randn(4, 4)
>>> a
tensor([[ 1.3398,  0.2663, -0.2686,  0.2450],
        [-0.7401, -0.8805, -0.3402, -1.1936],
        [ 0.4907, -1.3948, -1.0691, -0.3132],
        [-1.6092,  0.5419, -0.2993,  0.3195]])
>>> torch.argmax(a, dim=1)
tensor([ 0,  2,  0,  1])
```

실습: Shift Reduce Parsing

Main

```
if (__name__ == "__main__"):
    output_dir = os.path.join(root_dir, "output")

    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    set_seed(seed=1234)

    config = {"mode": "train",
              "train_data_path": os.path.join(root_dir, "train.txt"),
              "test_data_path": os.path.join(root_dir, "test.txt"),
              "output_dir_path": output_dir,
              "pretrained_model_name_or_path": "monologg/koelectra-small-v3-discriminator",
              "label_vocab_data_path": os.path.join(root_dir, "label_vocab.txt"),
              "num_labels": 2,
              "max_length": 250,
              "epoch": 10,
              "batch_size": 64,
              }

    if (config["mode"] == "train"):
        train(config)
    elif config["mode"] == 'test':
        tokenizer = ElectraTokenizer.from_pretrained(config["pretrained_model_name_or_path"])
        electra_config = ElectraConfig.from_pretrained(config["output_dir_path"])
        model = ElectraForSequenceClassification.from_pretrained(config["output_dir_path"], config=electra_config).cuda()
        evaluate(model, tokenizer)
```

```
Stack :  20세 이하 월드컵에서 18년 만의 8강 진출이란 성과를 냈기에
Stack Top :  굳이
Queue Front :  없다는
Queue :  판단이다 .
Prediction :  1
Label :  1


Stack :  교육과학기술부 산하 ' 교육사이버안전센터 ( ECSC ) ' 는 " ( 국정원 산하
Stack Top :  있다 " 며
Queue Front :  내용을
Queue :  공지한 것으로 3일 확인됐다 .
Prediction :  0
Label :  0


Accuracy : 0.9472095363418221
```

과제

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

KU KONKUK UNIVERSITY

KAIST

구문 분석 II
(Pointer Network)

Practical Exercise

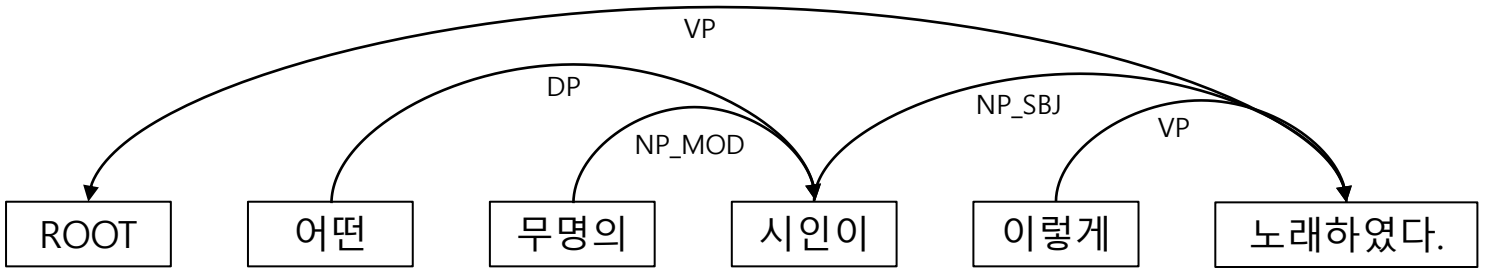
자연어처리연구실

실습: Pointer Network

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

- Transformer Encoder와 Pointer Network를 이용한 의존 구문 분석기 (Dependency Parser) 코드 리뷰

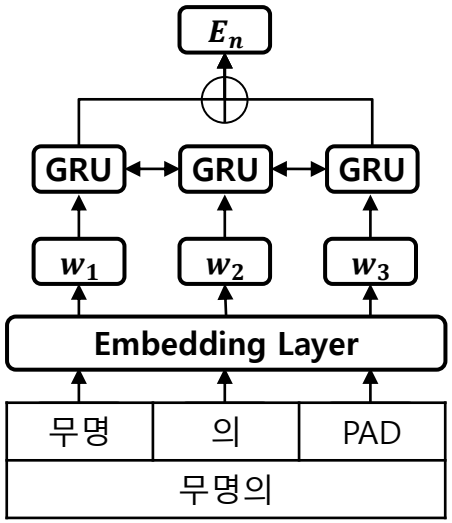
입력	**인덱스**	1	2		3		4		5					
	입력 문장	어떤	무명의		시인이		이렇게		노래하였다.					
	형태소 단위	어떤	무명	의	시인	이	이렇	게	노래	하	았	다	.	
출력	**정답 인덱스**	3	3		5		5		0					
	의존 관계명	DP	NP_MOD		NP_SBJ		VP		VP					

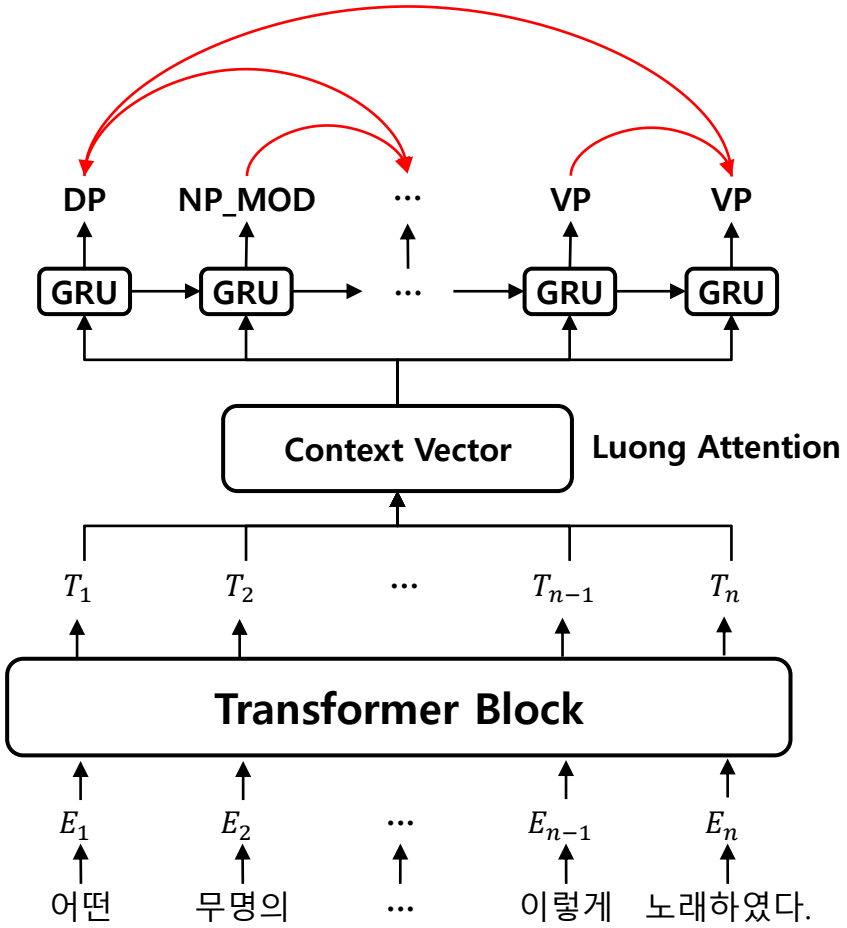


실습: Pointer Network

모델 구조

"무명의"에 대한 어절 임베딩 예시





실습: Pointer Network

모델 설계

생성자

```
class DependencyPaser(nn.Module):
    def __init__(self, config):
        super(DependencyPaser, self).__init__()
        # 형태소 임베딩 Matrix
        self.embeddings = nn.Embedding(num_embeddings=self.word_vocab_size,
                                       embedding_dim=self.embedding_size,
                                       padding_idx=0)
        # 형태소 Encoding layer
        self.word_encoder = nn.GRU(input_size=self.embedding_size,
                                   hidden_size=self.hidden_size,
                                   num_layers=1,
                                   batch_first=True,
                                   bidirectional=True)
        # Transformer Encoder layer
        self.trm_encoder = nn.TransformerEncoder(
            encoder_layer=nn.TransformerEncoderLayer(d_model=self.hidden_size*2,
                                                     nhead=self.num_heads,
                                                     dim_feedforward=self.hidden_size*2,
                                                     batch_first=True),
                                                     num_layers=2)
        # Bi-GRU Encoder layer
        self.gru_encoder = nn.GRU(input_size=self.hidden_size*2,
                                  hidden_size=self.hidden_size,
                                  num_layers=1,
                                  batch_first=True,
                                  bidirectional=True)
        self.att_decoder = Attn_Decoder(self.hidden_size, self.number_of_tags)
```

실습: Pointer Network

모델 설계

인코딩 단계

```
def forward(self, inputs_ids):
    # input_ids : [batch, eojeol_seq, word_seq]

    word_features = self.embeddings(inputs_ids).view(-1, self.max_word_length, self.embedding_size)
    # word_features : [batch, e_seq, word_seq, emb_size] -> [batch * eojeol_seq, word_seq, embedding_size]

    _, enc_hidden_state = self.word_encoder(word_features)
    # _ : [b*e_seq, word_seq, hidden*2]
    # enc_hidden_state : [2, batch * eojeol_seq, hidden]

    enc_eojeol = torch.cat([enc_hidden_state[0], enc_hidden_state[1]], -1)
    enc_eojeol = enc_eojeol.view(-1, self.max_eojeol_length, self.hidden_size*2)
    # enc_eojeol : [batch * eojeol_seq, hidden*2] => [batch, eojeol_seq, hidden*2]

    trm_encoder_outputs = self.trm_encoder(enc_eojeol)
    # trm_encoder_outputs : [batch, eojeol_seq, hidden*2]

    gru_encoder_outputs, gru_encoder_hidden = self.gru_encoder(trm_encoder_outputs)
    # gru_encoder_outputs : [batch, eojeol_seq, hidden*2]
    # gru_encoder_hidden  : [2, batch, hidden]
```

실습: Pointer Network

모델 설계

디코딩 단계

```
# 의존 구문 Tag 결과 저장을 위한 List
decoder_outputs = []

# 의존 구문 head 결과 저장을 위한 List
decoder_head_outputs = []

last_hidden = torch.cat([gru_encoder_hidden[0], gru_encoder_hidden[1]], -1).unsqueeze(0)
# [batch, hidden*2] => [1, batch, hidden*2]

for step in range(self.max_eojeol_length):
    decoder_input = enc_eojeol[:, step, :]
    # decoder_input : [batch, hidden*2]

    decoder_output, attn_weight, last_hidden = self.att_decoder(last_hidden, decoder_input, gru_encoder_outputs)
    # decoder_output : [batch, num_labels]
    # attn_weight : [batch, seq]
    # last_hidden : [1, batch, hidden*2]

    decoder_outputs.append(decoder_output)
    decoder_head_outputs.append(attn_weight)

label_hypothesis = torch.stack(decoder_outputs, 0).transpose(0, 1).contiguous()
# label_hypothesis : [eoj_seq, batch, num_labels] => [batch, seq, num_labels]

head_hypothesis = torch.stack(decoder_head_outputs, 0).transpose(0, 1).contiguous()
# head_hypothesis : [seq, batch, seq] => [batch, seq, seq]

return label_hypothesis, head_hypothesis
```

실습: Pointer Network

Main

```
if (__name__ == "__main__"):
    output_dir = os.path.join(root_dir, "output")
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    data_dir = os.path.join(root_dir, "data")
    config = {"mode": "test",
              "trained_model_name": "epoch_{}.pt".format(14),
              "train_data_path": os.path.join(data_dir, "refine_train.txt"),
              "test_data_path": os.path.join(data_dir, "refine_test.txt"),
              "output_dir_path": output_dir,
              "word_vocab_path": os.path.join(data_dir, "vocab.txt"),
              "label_vocab_path": os.path.join(data_dir, "label_vocab.txt"),
              "word_vocab_size": 44095,
              "embedding_size": 100,
              "hidden_size": 100,
              "max_eojeol_length": 30,
              "max_word_length": 25,
              "number_of_tags": 38,
              "epoch": 20,
              "num_heads": 10,
              "batch_size": 64,
              "dropout": 0.2,
              "learning_rate":5e-4
              }

    if (config["mode"] == "train"):
        train(config)
    else:
        test(config)
```

```
[ROOT] 어쨌든/MAG 이튿날/NNG+부터/JX 나/NP+의/JKG 생활/NNG+에/JKB+ㄴ/JX 다소/MAG 움직임/N
구문 태그          피지배소                                  지배소

ROOT        [ROOT]                          [ROOT]

AP          어쨌든/MAG                        생기/VV+었/EP+다/JX+./SF

NP_AJT      이튿날/NNG+부터/JX                 생기/VV+었/EP+다/JX+./SF

NP_MOD      나/NP+의/JKG                      생활/NNG+에/JKB+ㄴ/JX

NP_AJT      생활/NNG+에/JKB+ㄴ/JX              생기/VV+었/EP+다/JX+./SF

AP          다소/MAG                          생기/VV+었/EP+다/JX+./SF

NP_SBJ      움직임/NNG+이/JKS                 생기/VV+었/EP+다/JX+./SF

VP          생기/VV+었/EP+다/JX+./SF           [ROOT]
```

과제

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

Edited by Harksoo Kim

질의응답

Homepage: http://nlp.konkuk.ac.kr
E-mail: nlpdrkim@konkuk.ac.kr