KU KONKUK UNIVERSITY

KAIST

담화 분석
(Discourse Analysis)

Practical Exercise

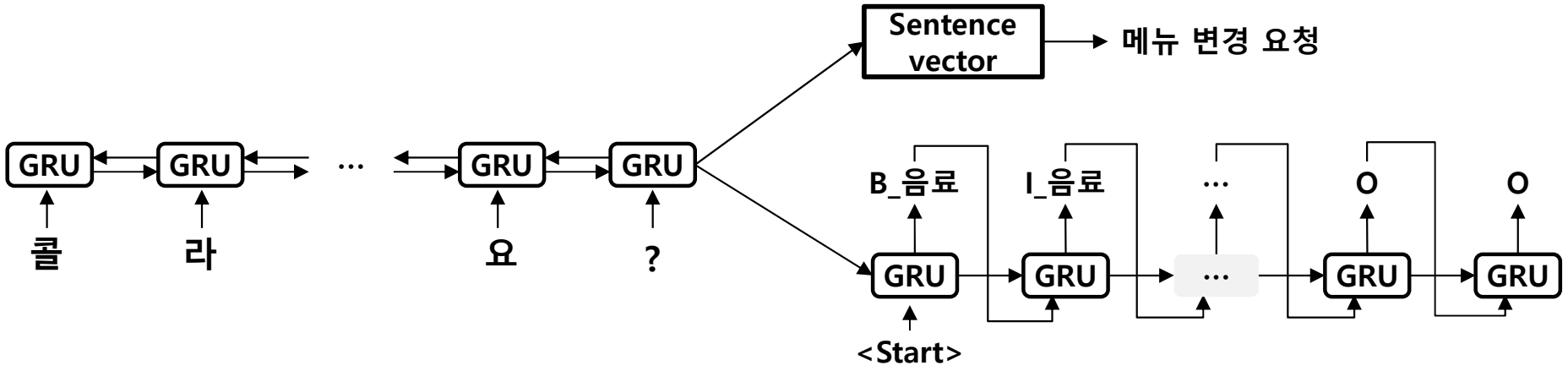
자연어처리연구실

실습

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

- 다중 디코더를 이용한 User Intent Detection/Slot Filling 통합 모델
  - 콜라 대신 커피로 바꿔도 될까요?
    - User Intent : 메뉴 변경 요청
    - Slot Filling : <콜라 : 음료>, <커피 : 음료>

모델 구조



실습: User Intent Detection+Slot Filling

모델 설계

생성자

```
class TaskOrientedChat(nn.Module):
    def __init__(self, flags):
        super(TaskOrientedChat, self).__init__()

        # 전체 음절 개수
        self.char_vocab_size = flags["char_vocab_size"]

        # 음절 임베딩 사이즈
        self.embedding_size = flags["embedding_size"]

        # GRU 히든 사이즈
        self.hidden_size = flags["hidden_size"]

        # 분류할 태그의 개수
        self.number_of_intent = flags["number_of_intents"] # 225
        self.number_of_slot_tag = flags["number_of_tags"] # 290

        # 디코딩 단계에서 필요한 문장 최대 길이
        self.max_sequence_length = flags["max_length"]

        # 예제 코드는 랜덤으로 초기화 한 후 같이 학습하도록 설정

        self.embedding = nn.Embedding(num_embeddings=self.char_vocab_size,
                                      embedding_dim=self.embedding_size,
                                      padding_idx=0)
        # Bi-GRU layer
        self.gru_encoder = nn.GRU(input_size=self.embedding_size, hidden_size=self.hidden_size, batch_first=True,
                                  bidirectional=True)
        self.gru_decoder = Decoder(embedding_size=self.hidden_size*2, hidden_size=self.hidden_size*2,
                                   output_size=self.number_of_slot_tag)

        # fully_connected layer를 통하여 출력 크기를 number_of_tag와 num_intent에 맞춰줌
        # (batch_size, max_seq_length, hidden_size) -> (batch_size, max_seq_length, number_of_tags)
        self.hidden2num_intent = nn.Linear(in_features=self.hidden_size*2, out_features=self.number_of_intent)
        self.hidden2num_tag = nn.Linear(in_features=self.hidden_size*2, out_features=self.number_of_slot_tag)
```

```
class Decoder(nn.Module):
    def __init__(self, embedding_size, hidden_size, output_size):
        super(Decoder, self).__init__()
        self.embedding_size = embedding_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        self.embed = nn.Embedding(output_size, embedding_size)
        self.gru = nn.GRU(input_size=embedding_size, hidden_size=self.hidden_size, batch_first=True,
                          num_layers=1)

    def forward(self, input, last_hidden):
        # Get the embedding of the current input word (last output word)
        # batch
        embedded = self.embed(input) # (batch, embedding_size)
        embedded = embedded.unsqueeze(1) # (batch,1,embedding_size)
        # Calculate attention weights and apply to encoder outputs

        # output : (batch, 1(step), hidden_size) hidden : (1(step x, num layer), batch, hidden_size)
        output, hidden = self.gru(embedded, last_hidden)
        output = output.squeeze(1)  # (batch, hidden_size)
        return output, hidden
```

실습: User Intent Detection+Slot Filling

모델 설계

```
def forward(self, input_ids, slot_labels, intent_labels, train_flag=True):
    batch_size = input_ids.size(0)

    # input_idx : (batch, max_seq_length)
    # char_inputs : (batch, max_seq_length, embedding_size)
    char_inputs = self.embedding(input_ids)

    _, encoder_hidden_state = self.gru_encoder(char_inputs) # encoder_hidden_state : (2, batch, hidden_size)

    # encoder_hidden_state : (1, batch, hidden_size*2)
    encoder_hidden_state = torch.cat([encoder_hidden_state[0] ,encoder_hidden_state[1]], -1)

    # intent_logits : (batch, num_intent_labes)
    intent_logits = self.hidden2num_intent(encoder_hidden_state).squeeze(0)

    outputs = [] # 디코더를 이용한 출력 값을 저장하기 위한 리스트

    last_hidden = encoder_hidden_state.unsqueeze(0) # 디코더 히든 초기값 설정

    # 디코더 입력을 만들기 위한 [START] 심볼
    # start_symbol : (batch, 1)
    start_symbol = torch.ones(size=(batch_size, 1), dtype=torch.int64).cuda() # (batch, 1)

    if train_flag:
        # start_symbol : (batch, 1)
        # slot_labels : (batch, max_seq_length)
        # decoder_input_features : (batch, max_seq_length)
        decoder_input_features = torch.cat([start_symbol, slot_labels], -1)[:, :-1]

        for step in range(self.max_sequence_length):
            decoder_input = decoder_input_features[:, step] # decoder_input : (batch,)

            # decoder_output : (batch, 1, hidden_size)
            # last_hidden : (1, batch, hidden)
            decoder_output, last_hidden = self.gru_decoder(decoder_input, last_hidden)
            slot_logits = self.hidden2num_tag(decoder_output) # slot_logits : (batch, 1, num_slot_tags)
            outputs.append(slot_logits)

        # all_slot_logits : (batch, step, num_tag)
        all_slot_logits = torch.stack(outputs, 0).transpose(0, 1)

        return all_slot_logits, intent_logits
```

모델 학습

모델 평가

```
else:
    # decoder_input_features : (batch,)
    decoder_input_features = start_symbol.squeeze(1)
    slot_output = None
    for step in range(self.max_sequence_length):
        if slot_output:
            decoder_input_features = slot_output

        # decoder_output : (batch, 1, hidden_size)
        # last_hidden : (1, batch, hidden)
        decoder_output, last_hidden = self.gru_decoder(decoder_input_features, last_hidden)

        # slot_logits : (batch, num_slot_tags)
        slot_logits = self.hidden2num_tag(decoder_output).squeeze(1)

        # slot_output : (batch, num_slot_tags) -> (batch)
        slot_output = torch.argmax(slot_logits, -1)
        outputs.append(slot_output)

    # slot_output : (step, batch)
    # all_slot_outputs : (batch, step)
    all_slot_outputs = torch.stack(outputs, 0).transpose(0, 1)
    intent_outputs = torch.argmax(intent_logits, -1) # (batch, num_intent) -> (batch)

    return all_slot_outputs, intent_outputs # (batch, step) / (batch)
```

실습: User Intent Detection+Slot Filling

학습

```
def train(config):
    # 학습에 필요한 딕셔너리 생성
    word2idx, idx2word = load_vocab(config["word_vocab_path"])
    tag2idx, idx2tag = load_vocab(config["tag_vocab_path"])
    intent2idx, idx2intent = load_vocab(config["intent_vocab_path"])

    # 학습 데이터 읽기
    train_char_sequences, train_tag_sequences, train_intents = read_data(config, config["train_data_path"],
                                                                         word2idx, tag2idx, intent2idx)

    # 평가 데이터 읽기
    test_char_sequences, test_tag_sequences, test_intents = read_data(config, config["test_data_path"],
                                                                      word2idx, tag2idx, intent2idx)

    # 학습 데이터를 batch 단위로 추출하기 위한 DataLoader 객체 생성
    train_features = TensorDataset(train_char_sequences, train_tag_sequences, train_intents)
    train_dataloader = DataLoader(train_features, shuffle=True, batch_size=config["batch_size"])

    # 평가 데이터를 batch 단위로 추출하기 위한 DataLoader 객체 생성
    test_features = TensorDataset(test_char_sequences, test_tag_sequences, test_intents)
    test_dataloader = DataLoader(test_features, shuffle=False, batch_size=1)

    #  모델 객체 생성
    model = TaskOrientedChat(config).cuda()

    # 모델 학습을 위한 optimizer
    optimizer = optim.Adam(model.parameters(), lr=config["learning_rate"])


    # epoch 마다 성능을 비교하기 위한 변수
    max_f1 = 0
    criterion = nn.CrossEntropyLoss(ignore_index=0)
```

실습: User Intent Detection+Slot Filling

학습

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

    losses = []
    for step, batch in enumerate(tqdm(train_dataloader)):
        batch = tuple(t.cuda() for t in batch)

        # 음절 데이터, 각 데이터의 실제 길이, 라벨 데이터
        input_ids, slot_tag_labels, intent_labels = batch[0], batch[1], batch[2]
        # (batch, step) / (batch, step) / (batch)


        # 모델 학습
        slot_tag_hypothesis, intent_hypothesis = model(input_ids, slot_tag_labels, intent_labels, True)
        # (batch*step, num_tags) / (batch, num_intent)

        slot_tag_labels = slot_tag_labels.view(-1, ) # (batch*step)
        slot_tag_hypothesis = slot_tag_hypothesis.reshape(-1, config["number_of_tags"])
        tag_loss = criterion(slot_tag_hypothesis, slot_tag_labels)
        intent_loss = criterion(intent_hypothesis, intent_labels)

        total_loss = (tag_loss*0.8) + (intent_loss*0.2)

        optimizer.zero_grad()
        total_loss.backward()

        # 모델 내부 각 매개변수 가중치 갱신
        optimizer.step()

        # batch 단위 loss 값 저장
        losses.append(total_loss.data.item())
```

학습모드

가중치에 따른 Loss 계산

실습: User Intent Detection+Slot Filling

평가

```
def test(config):
    word2idx, idx2word = load_vocab(config["word_vocab_path"])
    tag2idx, idx2tag = load_vocab(config["tag_vocab_path"])
    intent2idx, idx2intent = load_vocab(config["intent_vocab_path"])
    # 평가 데이터 읽기
    test_char_sequences, test_tag_sequences, test_intents = read_data(config, config["test_data_path"],
                                                                      word2idx, tag2idx, intent2idx )

    # 평가 데이터를 batch 단위로 추출하기 위한 DataLoader 객체 생성
    test_features = TensorDataset(test_char_sequences, test_tag_sequences, test_intents)
    test_dataloader = DataLoader(test_features, shuffle=True, batch_size=1)
    model.eval()

    # 모델의 출력 결과와 실제 정답값을 담을 리스트
    total_tag_predicts, total_intent_predicts, total_tag_answers, total_intent_answers = [], [], [], []
    for step, batch in enumerate(tqdm(test_dataloader, desc="test")):
        batch = tuple(t.cuda() for t in batch)

        # 음절 데이터, 각 데이터의 실제 길이, 라벨 데이터
        char_sequence, tag_sequence, intent = batch[0], batch[1], batch[2] # (1, step) / (1, step) / (1)

        # 모델 평가
        tag_predicts, intent_predict = model(char_sequence, tag_sequence, intent, False) # (1, step) / (1

        # Tensor를 numpy array로 변경하고 입력 데이터의 실제 길이만큼 추출
        char_sequence = tensor2list(char_sequence)[0] # (step)
        tag_predicts = tensor2list(tag_predicts)[0] # (1, step) -> (step)
        tag_sequence = tensor2list(tag_sequence)[0] # (1, step) -> (step)

        intent_predict = tensor2list(intent_predict) # (1)
        intent_answer = tensor2list(intent) # (1)

        # batch 단위 출력 결과와 정답을 리스트에 저장
        total_tag_predicts += tag_predicts
        total_tag_answers += tag_sequence

        total_intent_predicts += [intent_predict]
        total_intent_answers += intent_answer
```

평가모드

Slot Tag 결과와
Intent 결과 저장

```
# 모델 출력 라벨 sequence와 정답 라벨 sequence를 기반으로
# 모델 출력 문장과 정답 문장 출력
def make_sentence(char_sequence, tag_predicts, tag_answers, intent_predict, intent_answer,
                  idx2char, idx2tag, idx2intent):
    # 빈 문자열 생성
    input_sentence = []
    predict_tags = []
    answer_tags = []

    for index in range(len(char_sequence)):
        if char_sequence[index] == 0:
            break
        else:
            input_sentence.append(idx2char[char_sequence[index]])
            predict_tags.append(idx2tag[tag_predicts[index]])
            answer_tags.append(idx2tag[tag_answers[index]])
    predict_intent = idx2intent[intent_predict]
    answer_intent = idx2intent[intent_answer]

    origin_sentence = ' '.join(input_sentence)
    predict_result = ' '.join(predict_tags)
    answer_result =  ' '.join(answer_tags)
    print("########################################")
    print(origin_sentence)
    print("정답 의도: {}₩t 예측 의도: {}".format(answer_intent, predict_intent))

    return predict_result, answer_result
```

실습: User Intent Detection+Slot Filling

Main

```
import os
if(__name__=="__main__"):
    output_dir = os.path.join(root_dir, "output")
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    config = {"mode": "train",
              "trained_model_name":"epoch.pt",
              "train_data_path":os.path.join(root_dir, "r_train.txt"),
              "test_data_path": os.path.join(root_dir, "r_test.txt"),
              "output_dir_path":output_dir,
              "word_vocab_path": os.path.join(root_dir, "vocab.txt"),
              "tag_vocab_path": os.path.join(root_dir, "slot_label.txt"),
             "intent_vocab_path": os.path.join(root_dir, "intent_label.txt"),
              "char_vocab_size": 783,
              "embedding_size": 100,
              "hidden_size": 200,
              "max_length": 30,
              "number_of_tags": 290,
             "number_of_intents": 225,
              "epoch":50,
              "batch_size":128,
              "dropout":0.2,
             "learning_rate": 0.005

              }

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

과제

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

```
100%|██████████| 29/29 [00:00<00:00, 31.73it/s]
100%|██████████| 384/384 [00:04<00:00, 87.37it/s]
recall =  0.45171849427168576
precision  0.5494359654943597
Average loss : 0.10816839517190538,     Slot Tag F1 score : 0.495808383233533,  Intent Accuracy : 0.5989583333333334
```

KU KONKUK UNIVERSITY

Chitchat Bot

Practical Exercise

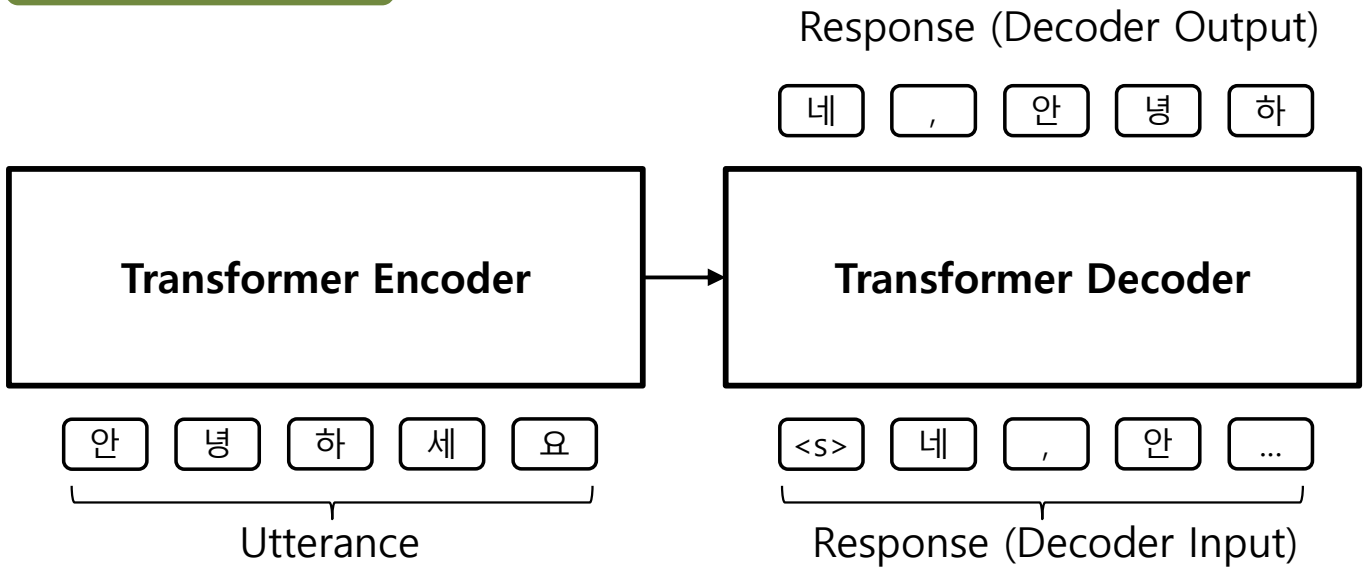
자연어처리연구실

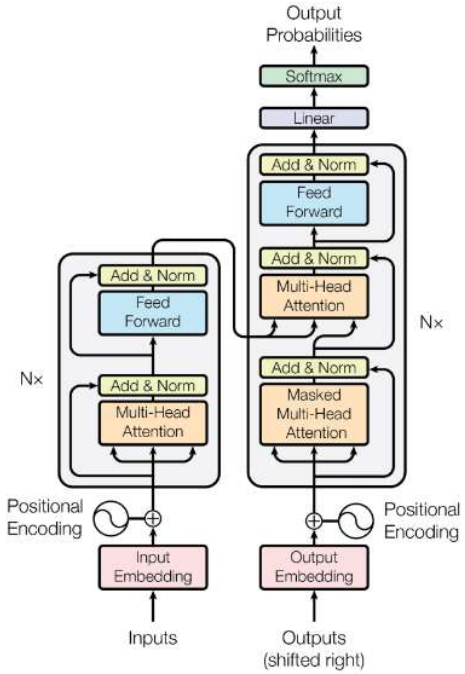
실습

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

- Transformer Encoder-Decoder를 이용한 Chitchat Bot 만들기
  - [Utterance] 오늘 너무 피곤했어..
  - [Response] 일찍 들어가서 쉬어요

모델 구조





실습: Chitchat Bot

모델 설계

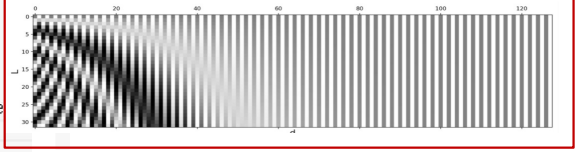
생성자

```python
# Transformer Seq2Seq 모델
class TransformerModel(nn.Module):
    def __init__(self, config):
        super().__init__()

        # 전체 단어(word) 개수
        self.vocab_size = config["vocab_size"]
        self.pad_id = config['pad_id']
        # 단어(word) 벡터 크기
        self.embedding_size = config['embedding_size']

        # Transformer의 Attention Head 개수
        self.num_heads = config['num_heads']

        # Transformer Encoder의 Layer 수
        self.num_encoder_layers = config['num_encoder_layers']

        # Transformer Decoder의 Layer 수
        self.num_decoder_layers = config['num_decoder_layers']

        # 입력 Sequence의 최대 길이
        self.max_encode_length = config['max_length']

        # Target Sequence의 최대 길이
        self.max_decode_length = config['max_decoding_length']

        # Transformer 내부 FNN 크기
        self.hidden_size = config['hidden_size']

        # Word Embedding Matrix 선언
        self.embeddings = nn.Embedding(self.vocab_size, self.embedding_size)

        # Transformer Encoder-Decoder 선언
        self.transformer = Transformer(d_model=self.embedding_size, nhead=self.num_heads, num_encoder_layers=self.num_encoder_laye
                                       num_decoder_layers=self.num_decoder_layers, dim_feedforward=self.hidden_size)
        # 입력 길이 L에 대한 (L X L) mask 생성
        # 이전 토큰들의 정보만을 반영하기 위한 mask
        #       [[1, -inf, -inf, -inf],
        #        [1,    1, -inf, -inf],
        #        [1,    1,    1, -inf],
        #        [1,    1,    1,    1]]
        self.src_mask = None
        self.trg_mask = torch.triu(torch.ones(self.max_decode_length, self.max_decode_length), diagonal=1).bool().cuda()

        # 전체 단어 분포로 변환하기 위한 linear
        self.projection_layer_1 = nn.Linear(self.embedding_size, self.hidden_size)
        self.activation = nn.GELU()
        self.projection_layer_2 = nn.Linear(self.hidden_size, self.vocab_size)
```

```python
# Trm에 순차정보를 반영하기 위한 PositionalEncoding
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout = 0.1, max_len = 5000):
        # d_model : Tensor의 마지막 Dimension
        super().__init__()
        self.dropout = nn.Dropout(dropout)

        position = torch.arange(max_len).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2) * (-math.log(10000.0) / d_model))
        pe = torch.zeros(max_len, 1, d_model)
        pe[:, 0, 0::2] = torch.sin(position * div_term)
        pe[:, 0, 1::2] = torch.cos(position * div_term)
        self.register_buffer('pe', pe)
    # [max_length, 1, emb]
    def forward(self, x):
        # x : [seq, batch, emb]
        x = x + self.pe[:x.size(0)]
        return self.dropout(x)
```

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$



실습: Chitchat Bot

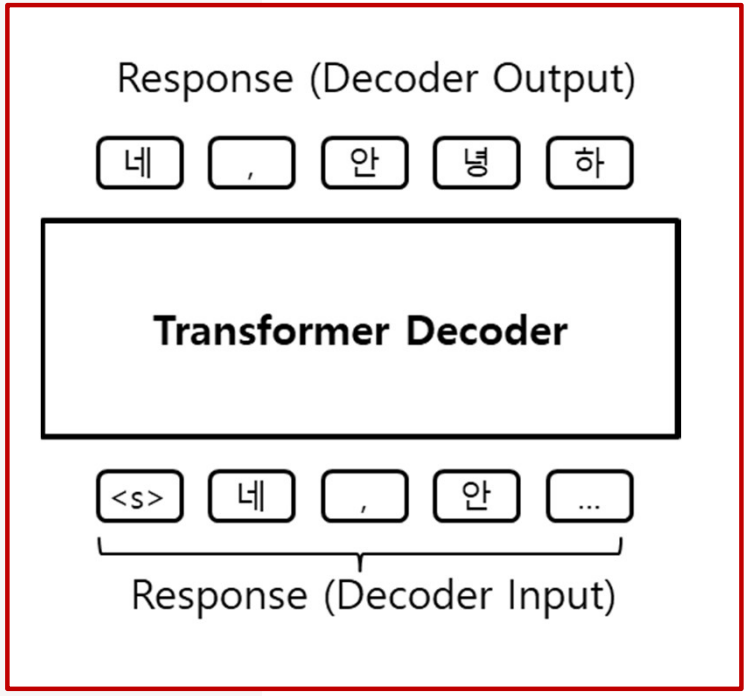
모델 설계

```
def forward(self, input_ids, target_input_ids):
    # input_ids : [batch, enc_seq]
    # target_input_ids : [batch, dec_seq]

    encoder_input_features = self.embeddings(input_ids).transpose(0, 1)
    # encoder_input_features : [enc_seq, batch, emb_size]

    decoder_input_features = self.embeddings(target_input_ids).transpose(0, 1)
    # decoder_input_features : [dec_seq, batch, emb_size]

    # src : [enc_seq, batch, hidden]
    # tgt : [dec_seq, batch, hidden]
    # src_mask : [enc_seq, enc_seq] : 입력 Sequence Attention Mask (일반적으로 사용하지 않음)
    # tgt_mask : [dec_seq, dec_seq] : 출력 Sequence Attention Mask (Trm decoding을 위한 계단식 MASK)
    # memory_mask : [dec_seq, enc_seq] : Encoder-Decoder Cross Attention Mask
    # src_key_padding_mask : [batch, enc_seq] : 입력 Padding Mask
    # tgt_key_padding_mask : [batch, dec_seq] : 출력 Padding Mask
    # memory_key_padding_mask : [batch, enc_seq] : Encoder-Decoder Cross Attention에 사용되는 입력 Padding Mask


    src_key_padding_mask = input_ids.masked_fill(input_ids == self.pad_id, 1).masked_fill(input_ids != self.pad_id, 0).bool()
    # src_pad_mask : [batch, enc_seq]
    tgt_key_padding_mask = target_input_ids.masked_fill(target_input_ids == self.pad_id, 1).masked_fill(target_input_ids != self.pad_id, 0).bool()
    # src_pad_mask : [batch, dec_seq]

    decoder_output_features = self.transformer(src=encoder_input_features, tgt=decoder_input_features,
                                               src_mask = self.src_mask, tgt_mask = self.trg_mask,
                                               src_key_padding_mask=src_key_padding_mask, tgt_key_padding_mask=tgt_key_padding_mask)
    # decoder_output_features : [dec_seq, batch, emb_size]

    output_1 = self.projection_layer_1(decoder_output_features)
    # output_1 : [dec_seq, batch, hidden]

    act_output_1 = self.activation(output_1)
    result = self.projection_layer_2(act_output_1)
    # result : [dec_len, batch, vocab_size]
    return result
```

실습: Chitchat Bot

데이터 전처리

```python
# 입력 Sequence를 대응하는 인덱스로 변환하는 함수
def convert_data2feature(config, input_sequence, word2idx, decoding_flag = False, decoder_input=False):
    sequence_length = config["max_length"] if not decoding_flag else config["max_decoding_length"]
    # 고정 길이 벡터 생성
    input_features = np.zeros(sequence_length, dtype=np.int)

    if decoding_flag:
        # Decoder Input은 Target Sequence에서 Right Shift
        # Reference Sentence :      ["안", "녕", "하", "세", "요"]

        # Target Sequence :         ["안",  "녕","하","세","요", "</S>" ]
        # Decoder Input Sequence :  ["<S>", "안","녕","하","세", "요"]
        if decoder_input:
            input_sequence = " ".join(["<S>"] + input_sequence.split())
        else:
            input_sequence = " ".join(input_sequence.split() + ["</S>"])

    for idx,word in enumerate(input_sequence.split()):
        if word in word2idx.keys():
            input_features[idx] = word2idx[word]
        else:
            input_features[idx] = word2idx['<UNK>']
    return input_features
```



실습: Chitchat Bot

학습

```
def train(config):
    model = TransformerModel(config).cuda() # Transformer Seq2Seq 모델 객체 생성

    # 데이터 읽기
    input_features, target_input_features, target_features, word2idx, idx2word = load_dataset(config, 'train')

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

    # 크로스엔트로피 손실 함수
    criterion = nn.CrossEntropyLoss()

    # 옵티마이저 함수 지정
    optimizer = torch.optim.Adam(model.parameters(), lr=config["learn_rate"])
    max_rouge = 0
    for epoch in range(config["epoch"] + 1):
        model.train()
        for (step, batch) in en
            # batch = (input_fe                    tures[step], target_features)*batch_size
            # .cuda()를 통해 메모리에 업로드
            batch = tuple(t.cuda() for t in batch)

            # 역전파 변화도 초기화
            optimizer.zero_grad()

            input_ids, target_input_ids, target_ids = batch
            hypothesis = model(input_ids, target_input_ids)

            hypothesis = hypothesis.view(-1, config['vocab_size'])
            # hypothesis : [dec_seq * batch, vocab_size]
            # labels : [dec_seq, batch] ==> [dec_seq * batch]

            labels = target_ids.transpose(0, 1) # 생성한 Sequence에 대한 비용 계산
            labels = labels.reshape(input_ids.size(0) * config["max_decoding_length"])
            # labels : [seq_len * batch]
            loss = criterion(hypothesis, labels)

            # 손실 역전파 수행
            loss.backward()
```

학습모드

Loss 계산

실습: Chitchat Bot

평가

```
def do_test(model):
    # 평가 모드 셋팅
    model.eval()

    # 데이터 읽기
    input_features, _, target_features, word2idx, idx2word = load_dataset(config, 'test')

    # TensorDataset/DataLoader를 통해 배치(batch) 단위로 데이터를 나누고 셔플(shuffle)
    test_features = TensorDataset(input_features, target_features)
    test_dataloader = DataLoader(test_features, shuffle=True, batch_size=config["batch_size"])

    utterances = []
    references = []
    responses = []
    for (step, batch) in enumerate(test_dataloader):
        # batch = (input_features[step], target_features[step])*batch_size
        # .cuda()를 통해 메모리에 업로드
        batch = tuple(t.cuda() for t in batch)
        input_ids, target_ids = batch
        batch_size = input_ids.size(0)
        target_input_ids = torch.zeros([batch_size, config["max_decoding_length"]], dtype=torch.long).cuda()
        target_input_ids[:, 0] = word2idx["<S>"]
        # [<s>, 0, 0, 0, 0]

        decoding_idx = []
        for decoding_step in range(config["max_decoding_length"]-1):
            outputs = model(input_ids, target_input_ids)
            # outputs : [seq_len, batch, vocab_size]

            outputs = torch.argmax(outputs, -1)
            # outputs : [dec_seq, batch]

            decoding_idx+= outputs[decoding_step, :].unsqueeze(0)
                        # [dec_seq, batch] ==> [batch] ==> [1, batch]
            target_input_ids[:, decoding_step+1] += outputs[decoding_step, :]

        decoding_idx = torch.stack(decoding_idx, 0).transpose(0, 1)
        # [batch, dec_seq]
```

평가모드

Greedy Decoding 수행

```
# Tensor -> List 변환
utterance = [convert_feature2data(e, idx2word) for e in tensor2list(input_ids)]
response = [convert_feature2data(e, idx2word) for e in tensor2list(decoding_idx)]
reference = [convert_feature2data(e, idx2word) for e in tensor2list(target_ids)]

# 생성 결과 및 실제 답변 추가
utterances += utterance
responses += response
references += reference
```

```
def measure_rouge(input_sentence, answers, predicts):
    rouge_list = []
    for idx in range(len(answers)):
        if idx < 5:
            print("##########################################################")
            print("Input Utterance : {}₩n₩n".format(input_sentence[idx]))
            print("Reference Sentence : {}".format(answers[idx]))
            print("Response Sentence : {}₩n".format(predicts[idx]))
    rouge = Rouge()
    rouge_1_score = rouge.get_scores(predicts, answers, avg=True)["rouge-1"]['f']

    return rouge_1_score
```

실습: Chitchat Bot

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": "train",
              "vocab_file": os.path.join(data_dir, "vocab.txt"),
              "train_file": os.path.join(data_dir, "chat_train.txt"),
              "test_file": os.path.join(data_dir, "chat_test.txt"),
              "trained_model_name": "epoch_{}.pt".format(1),
              "output_dir": output_dir,
              "epoch": 50,
              "learn_rate": 0.0001,
              "num_encoder_layers": 6,
              "num_decoder_layers": 6,
              "num_heads": 4,
              "max_length": 100,
              "max_decoding_length": 100,
              "batch_size": 64,
              "embedding_size": 256,
              "hidden_size": 512,
              "vocab_size": 9714,
              "pad_id":0
             }

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

```
Input Utterance : 한 국 음 식 <SP> 중 에 <SP> 일 본 에 서 <SP> 인 기 <SP> 있 는 <SP> 음 식 은 <SP> 야 키 니 쿠 야 .


Reference Sentence : 어 떤 <SP> 음 식 인 지 <SP> 먹 어 보 고 <SP> 싶 은 걸 . </S>
Response Sentence : 정 말 <SP> 대 단 한 <SP> 것 <SP> 같 아 . </S>

########################################################
Input Utterance : 너 무 하 징 .


Reference Sentence : 사 람 이 <SP> 살 다 보 면 <SP> 그 럴 수 도 <SP> 있 죠 . <SP> ^ ^ ; </S>
Response Sentence : 에 이 ~ <SP> 뭐 <SP> 그 런 걸 루 <SP> 삐 지 구 <SP> 그 래 요 . </S>
```

질의응답

Q&A

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