KU KONKUK UNIVERSITY

Transformer

자연어처리연구실

본 챕터의 내용은 "Ta-Chun (Bgg/Gene) Su's blog (https://medium.com/@bgg/seq2seq-pay-attention-to-self-attention-part-2-cf81bf32c73d)" 자료를 참고하여 편집되었습니다.

Attention Models

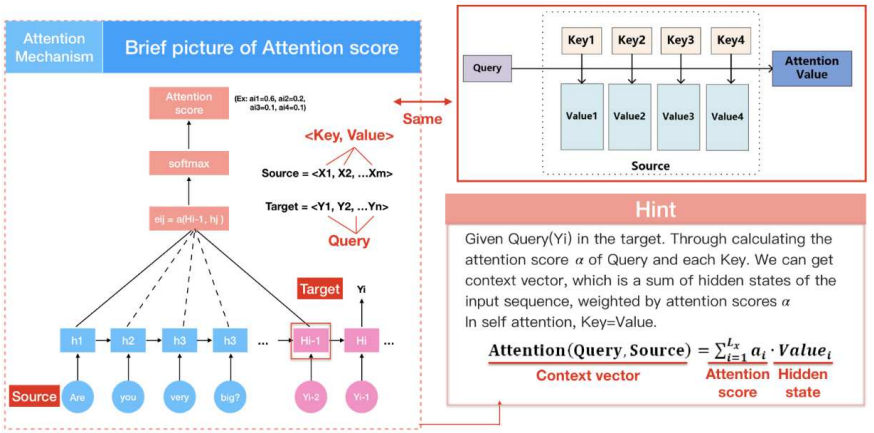
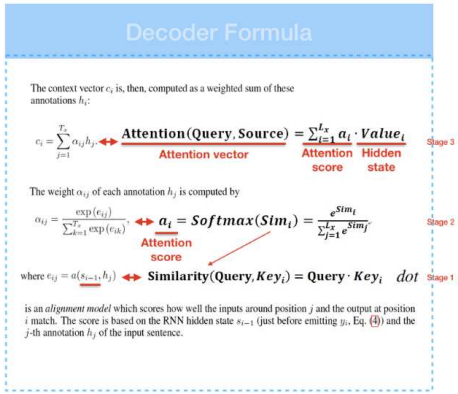


그림 출처: Ta-Chun (Bgg/Gene) Su's blog

Edited by Harksoo Kim

Attention Models in Detail



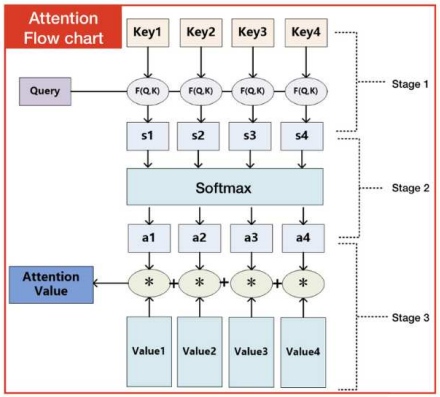


그림 출처: Ta-Chun (Bgg/Gene) Su's blog

Edited by Harksoo Kim

Problems of Attention-Based Models

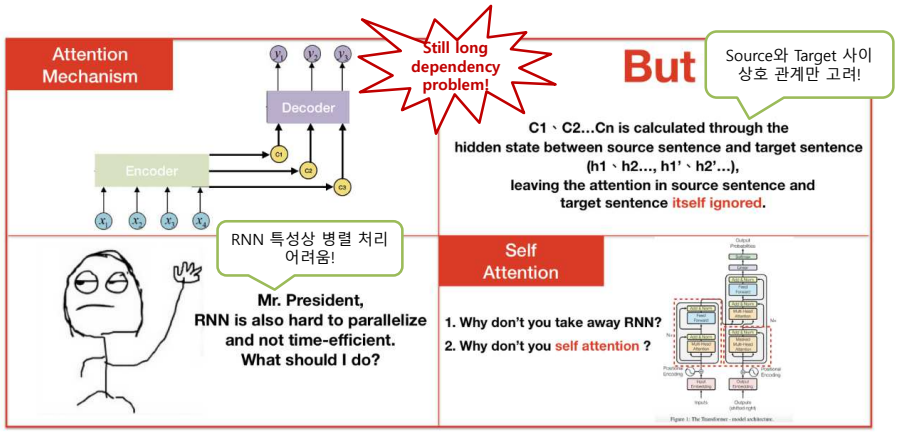


그림 출처: Ta-Chun (Bgg/Gene) Su's blog

Edited by Harksoo Kim

Transformer

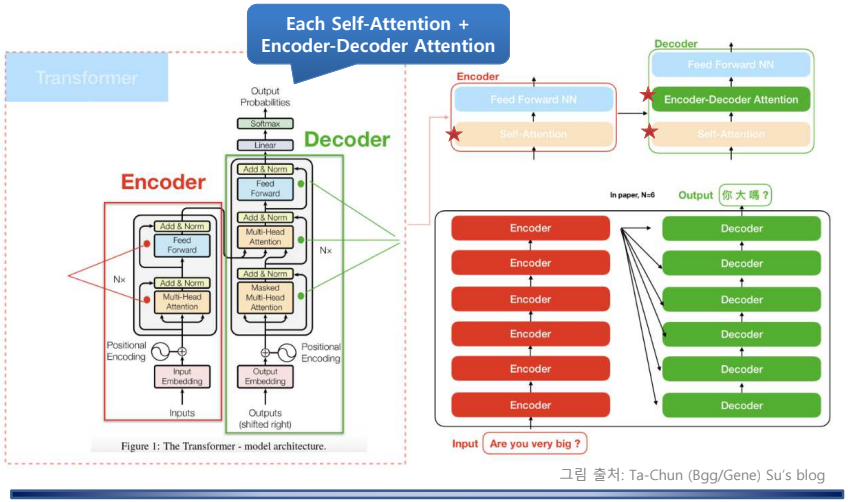
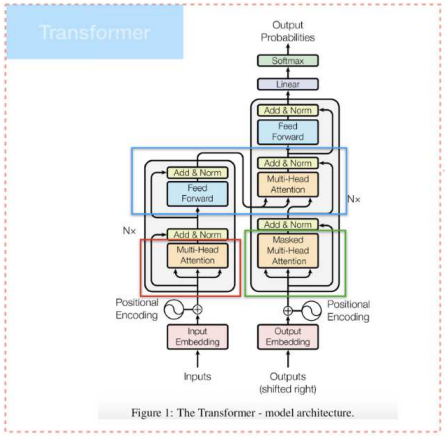


그림 출처: Ta-Chun (Bgg/Gene) Su's blog

Edited by Harksoo Kim

Attentions in Transformer



encoder self attention

1. Multi-head Attention
2. Query=Key=Value

decoder self attention

1. Masked Multi-head Attention
2. Query=Key=Value

encoder-decoder attention

1. Multi-head Attention
2. Encoder Self attention=Key=Value
3. Decoder Self attention=Query

그림 출처: Ta-Chun (Bgg/Gene) Su's blog

Edited by Harksoo Kim

Scaled Dot-Product Attention

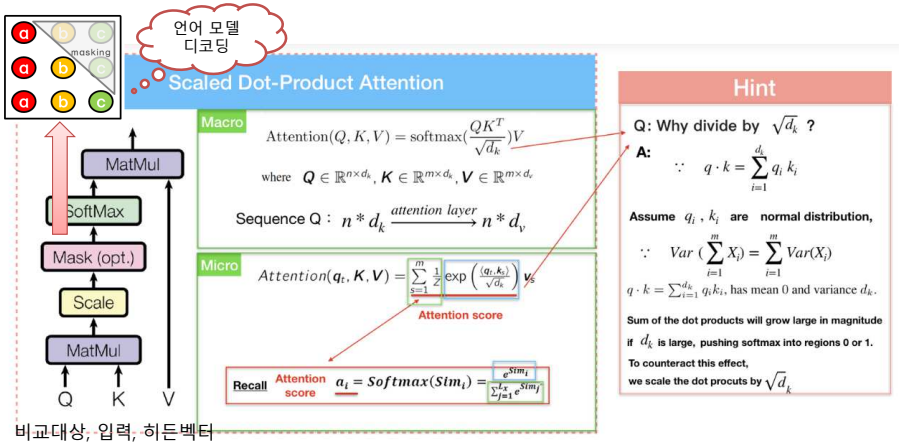


그림 출처: Ta-Chun (Bgg/Gene) Su's blog

Edited by Harksoo Kim

Calculation of Attentions

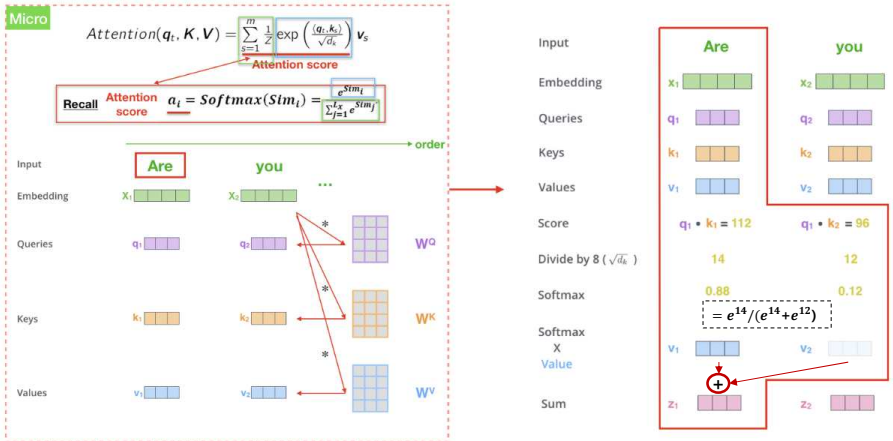


그림 출처: Ta-Chun (Bgg/Gene) Su's blog

Edited by Harksoo Kim

Calculation of Attentions

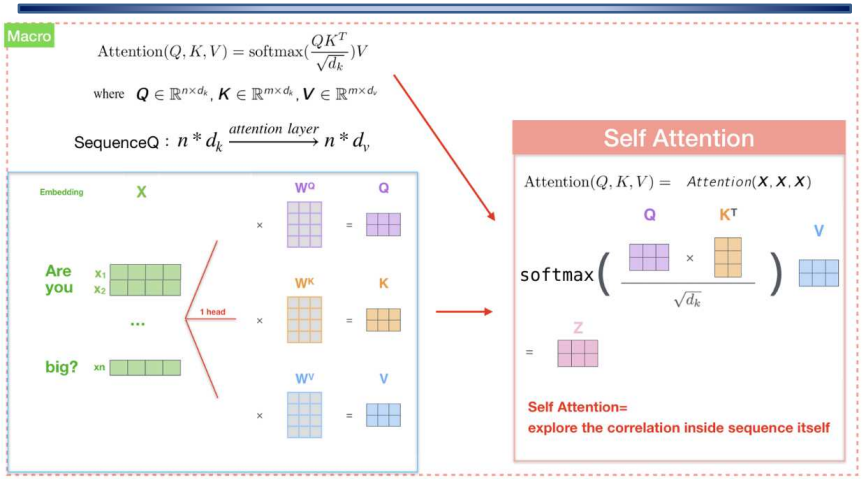


그림 출처: Ta-Chun (Bgg/Gene) Su's blog

확인 문제

- 다음과 같이 단어 임베딩이 주어졌을 때, self-attention score를 계산하시오. (소수점 이하 두 자리에서 반올림)
  - are: [1,1], you: [2,1]
  - root(2)=1.4로 계산

$$Attention(q_t, K, V) = \sum_{s=1}^{m} \frac{1}{Z} \exp\left(\frac{(q_t, k_s)}{\sqrt{d_k}}\right) v_s$$

Scaled dot-product

	are	you
are	?	
you		

Self-attention score

	are	you
are	?	
you		

Multi-Head Attention

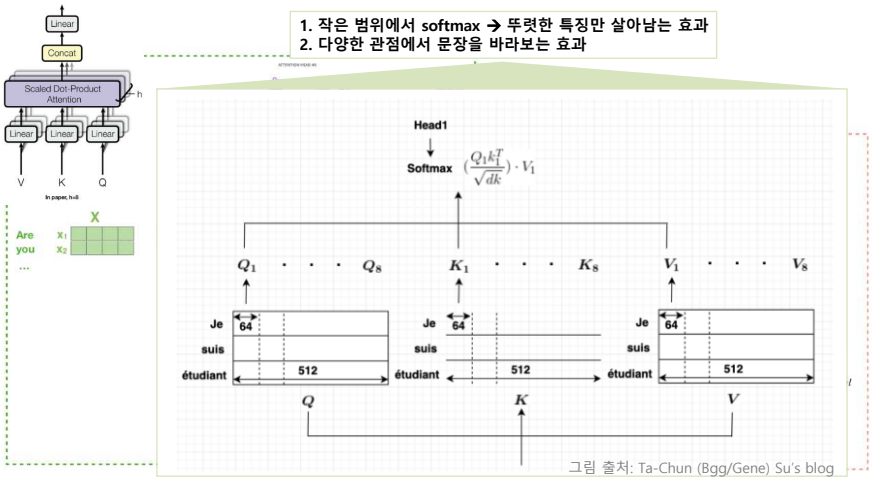


그림 출처: Ta-Chun (Bgg/Gene) Su's blog

Layer Norm. & Residual Conn.

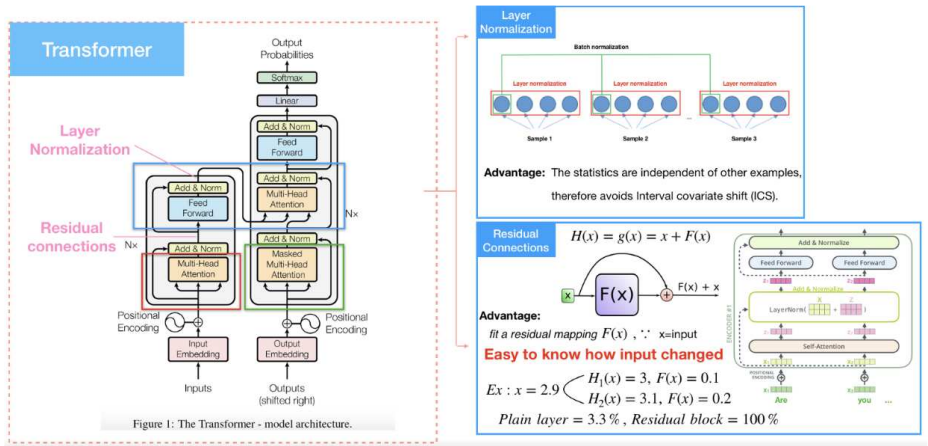
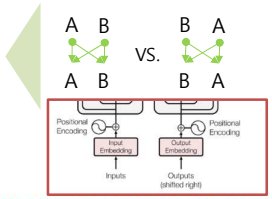


그림 출처: Ta-Chun (Bgg/Gene) Su's blog

Position Encoding

Problem

The multi-head attention network **cannot** naturally **make use of** the position of the words in the input sequence.
The output of the multi-head attention network would be **the same** for the same sentences in different order.



Sol Positional Encoding

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

$if\ shape\ of\ enc, dec = [T, d_{model}]$ **pos: position of the word**

$\rightarrow then\ pos \in [0, T),\ i \in [0, d_{model})$ **i: Element i in d_{model}**

Advantage:

PE[pos+k] can be represented as a linear function of **PE[pos]**, so the relative positon between different embeddings can be easily inferred

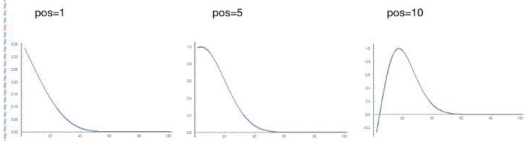
$sin(\alpha+\beta) = sin\alpha cos\beta + cos\alpha sin\beta$
$cos(\alpha+\beta) = cos\alpha cos\beta - sin\alpha sin\beta$
$\therefore$ Trigonometric Periodicity

$\therefore$ **Learned the relation between relative and absolute position**

Then Word embedding + positional encoding

(sum, concat...)

$if\ d_{model} = 512$:



→ **Three plots are the same by doing linear transformation**

그림 출처: Ta-Chun (Bgg/Gene) Su's blog

Linear & Softmax

Linear + Softmax

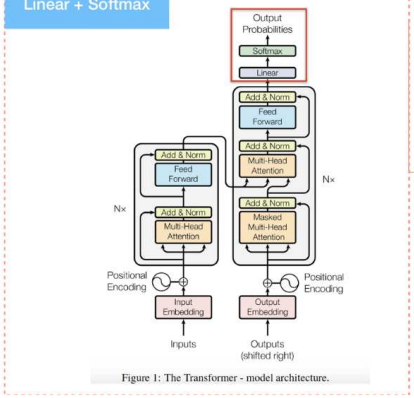


Figure 1: The Transformer - model architecture.

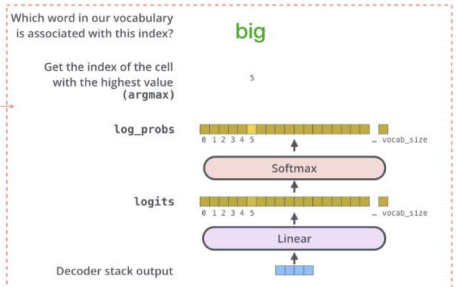


그림 출처: Ta-Chun (Bgg/Gene) Su's blog

실습

실습 코드 다운로드: https://github.com/KUNLP/Lecture

- Transformer를 이용하여 1문 1답 챗봇 프로그램을 작성하시오.
  - 입력 데이터셋
    - 1문 1답 텍스트: 질문 ₩t 응답 (음절로 분리, 공백은 <SP>로 변환)
    - 어휘 사전: 음절 집합
  - 문제
    - 1문 1답 텍스트를 학습하여 새로운 질문에 대한 적절한 응답을 생성

train.txt - Windows 메모장

```
뭐 하 고 있 어 ?	이 제 <SP> 수 업 <SP> 끝 났 어 ! </S>
넌 <SP> 뭐 해	대 학 생 이 야 ? </S>
난 <SP> 지 금 <SP> 과 자 <SP> 먹 어 응 <SP> 대 학 생 </S>
응 <SP> 대 학 생	전 공 이 <SP> 뭐 야 ? </S>
전 공 이 <SP> 뭐 야 ?	넌 <SP> 집 인 가 보 네 <SP> 부 럽 다 <SP> 나
난 <SP> 국 어 국 문 학 과	오 <SP> 좋 은 <SP> 학 문 <SP> 배 우 네 </S>
집 에 <SP> 가 고 싶 다	맹 <SP> 무 슨 <SP> 일 <SP> 하 길 래 ? ? </S>
퇴 근 이 <SP> 언 제 야	평 범 한 <SP> 직 장 인 이 지 <SP> 뭐 </S>
안 녕 하 세 요	안 녕 하 세 요 ! </S>
밥 <SP> 먹 었 나 요 ! ㅎ ㅎ 먹 었 습 니 다 </S>
저 는 <SP> 국 밥 ㅋ ㅋ ㅋ ㅋ ㅋ ㅋ	가 성 비 가 <SP> 좋 은 <SP> 국 밥 을
```

vocab.txt - Windows 메모장

```
가
갂
간
갅
갈
```


실습

```
from tqdm import tqdm
```

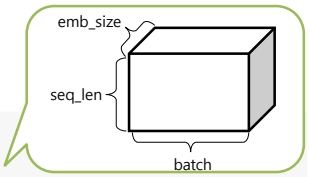
```
100%|██████████| 547958/547958 [00:09<00:00, 59999.28it/s]
```

모델 설계: __init__

```
class TransformerChat(nn.Module):

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

        # 전체 단어(음절) 개수
        self.vocab_size = config["vocab_size"]

        # 단어(음절) 벡터 크기
        self.embedding_size = config['embedding_size']

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

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

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

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

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

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

        # Transformer Encoder-Decoder 설계(선언)
        self.transformer = nn.Transformer(d_model=self.embedding_size, nhead=self.num_heads, num_encoder_layers=self.num_encoder_layers,
                                          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,    1]]
        self.mask = self.transformer.generate_square_subsequent_mask(self.max_length).cuda()

        # 전체 단어 분포로 변환하기 위한 linear
        self.projection_layer = nn.Linear(self.embedding_size, self.vocab_size)
```

실습

모델 설계: forward




```
def forward(self, enc_inputs, dec_inputs):

    # enc_inputs: [batch, seq_len], dec_inputs: [batch, seq_len]
    # enc_input_features: [batch, seq_len, emb_size] -> [seq_len, batch, emb_size]
    enc_input_features = self.embeddings(enc_inputs).transpose(0, 1)

    # dec_input_features: [batch, seq_len, emb_size] -> [seq_len, batch, emb_size]
    dec_input_features = self.embeddings(dec_inputs).transpose(0, 1)

    # dec_output_features: [seq_len, batch, emb_size]
    dec_output_features = self.transformer(src=enc_input_features, tgt=dec_input_features, src_mask = self.mask, tgt_mask = self.mask)

    # hypothesis : [seq_len, batch, vocab_size]
    hypothesis = self.projection_layer(dec_output_features)

    return hypothesis
```


실습

사전 읽기

```
# 어휘사전(vocabulary) 생성 함수
def load_vocab(file_dir):

    with open(file_dir,'r',encoding='utf8') as vocab_file:
        char2idx = {}
        idx2char = {}
        index = 0
        for char in vocab_file:
            char = char.strip()
            char2idx[char] = index
            idx2char[index] = char
            index+=1

    return char2idx, idx2char
```

전처리

```
# 문자 입력열을 인덱스로 변환하는 함수
def convert_data2feature(config, input_sequence, char2idx, decoder_input=False):

    # 고정 길이 벡터 생성
    input_features = np.zeros(config["max_length"], dtype=np.int)

    if decoder_input:
        # Decoder Input은 Target Sequence에서 Right Shift
        # Target Sequence :        ["안","녕","하","세","요", "</S>" ]
        # Decoder Input Sequence :  ["<S>", "안","녕","하","세","요"]
        input_sequence = " ".join(["<S>"] + input_sequence.split()[:-1])

    for idx,token in enumerate(input_sequence.split()):
        if token in char2idx.keys():
            input_features[idx] = char2idx[token]
        else:
            input_features[idx] = char2idx['<UNK>']

    return input_features
```

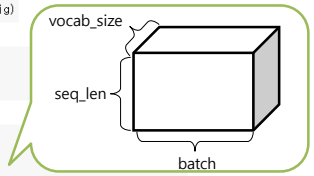
데이터 로드

```
# 데이터 읽기 함수
def load_dataset(config):

    # 어휘사전 읽어오기
    char2idx, idx2char = load_vocab(config['vocab_file'])

    file_dir = config['train_file']
    data_file = open(file_dir,'r',encoding='utf8').readlines()

    # 데이터를 저장하기 위한 리스트 생성
    enc_inputs, dec_inputs, dec_outputs = [], [], []

    for line in tqdm(data_file):

        line = line.strip().split('₩t')

        input_sequence = line[0]
        output_sequence = line[1]

        enc_inputs.append(convert_data2feature(config, input_sequence, char2idx))
        dec_inputs.append(convert_data2feature(config, output_sequence, char2idx, True))
        dec_outputs.append(convert_data2feature(config, output_sequence, char2idx))

    # 전체 데이터를 저장하고 있는 리스트를 텐서 형태로 변환
    enc_inputs = torch.tensor(enc_inputs, dtype=torch.long)
    dec_inputs = torch.tensor(dec_inputs, dtype=torch.long)
    dec_outputs = torch.tensor(dec_outputs, dtype=torch.long)

    return enc_inputs, dec_inputs, dec_outputs, char2idx, idx2char
```


실습

학습

```
def train(config):

    # Transformer Seq2Seq 모델 객체 생성
    model = TransformerChat(config).cuda()

    # 데이터 읽기
    enc_inputs, dec_inputs, dec_outputs, word2idx, idx2word = load_dataset(config)
```




```
for epoch in range(config["epoch"] + 1):

    for (step, batch) in enumerate(train_dataloader):

        enc_inputs, dec_inputs, dec_outputs = batch

        # hypothesis: [seq_len, batch, vocab_size] -> [seq_len*batch, vocab_size]
        hypothesis = model(enc_inputs, dec_inputs).view(-1, config['vocab_size'])

        # labels: [batch, seq_len] -> [seq_len, batch] -> [seq_len(max_length)*batch]
        labels = dec_outputs.transpose(0, 1)
        labels = labels.reshape(config["max_length"]*dec_inputs.size(0))

        # 비용 계산 및 역전파 수행: cross_entopy 내부에서 labels를 원핫벡터로 변환 (골드레이블은 항상 1차원으로 입력)
        loss = loss_func(hypothesis, labels)
        loss.backward()
        optimizer.step()
```


실습

평가

```
def do_test(config, model, word2idx, idx2word, input_sequence="오늘 약속있으셔요?"):

    # 평가 모드 셋팅
    model.eval()

    # 입력된 문자열의 음절을 공백 단위 토큰으로 변환, 공백은 <SP>로 변환: "오늘 약속" -> "오 늘 <SP> 약 속"
    input_sequence = " ".join([e if e != " " else "<SP>" for e in input_sequence])

    # 텐서 변환: [1, seq_len]
    enc_inputs = torch.tensor([convert_data2feature(config, input_sequence, word2idx)], dtype=torch.long).cuda()

    # input_ids : [1, seq_len] -> 첫번째 디코더 입력 "<S>" 만들기
    dec_inputs = torch.tensor([convert_data2feature(config, "", word2idx, True)], dtype=torch.long).cuda()

    # 시스템 응답 문자열 초기화
    response = ''

    # 최대 입력 길이 만큼 Decoding Loop
    for decoding_step in range(config['max_length']-1):

        # dec_outputs: [vocab_size]
        dec_outputs = model(enc_inputs, dec_inputs)[decoding_step, 0, :]
        # 가장 큰 출력을 갖는 인덱스 얻어오기
        dec_output_idx = np.argmax(tensor2list(dec_outputs))

        # 생성된 토큰은 dec_inputs에 추가 (첫번째 자원은 배치)
        dec_inputs[0][decoding_step+1] = dec_output_idx

        # </S> 심볼 생성 시, Decoding 종료
        if idx2word[dec_output_idx] == "</S>":
            break

        # 생성 토큰 추가
        response += idx2word[dec_output_idx]

    # <SP>를 공백으로 변환한 후 응답 문자열 출력
    print(response.replace("<SP>", " "))
```

```
def test(config):

    # 어휘사전 읽어오기
    word2idx, idx2word = load_vocab(config['vocab_file'])

    # Transformer Seq2Seq 모델 객체 생성
    model = TransformerChat(config).cuda()

    # 학습한 모델 파일로부터 가중치 불러옴
    model.load_state_dict(torch.load(os.path.join(config["output_dir"], config["trained_model_name"])))

    while(True):
        input_sequence = input("문장을 입력하세요. (종료는 exit을 입력하세요.) : ")
        if input_sequence == 'exit':
            break
        do_test(config, model, word2idx, idx2word, input_sequence)
```

실습

메인 함수

```
if(__name__=="__main__"):

    root_dir = "/gdrive/My Drive/colab/transformer/chatbot/"
    output_dir = os.path.join(root_dir, "output")
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    config = {"mode": "train",
              "vocab_file": os.path.join(root_dir, "vocab.txt"),
              "train_file": os.path.join(root_dir, "train.txt"),
              "trained_model_name":"epoch_{}.pt".format(10),
              "output_dir":output_dir,
              "epoch": 10,
              "learn_rate":0.00005,
              "num_encoder_layers": 6,
              "num_decoder_layers": 6,
              "num_heads": 4,
              "max_length": 20,
              "batch_size": 128,
              "embedding_size": 256,
              "hidden_size": 512,
              "vocab_size": 4427
             }

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

```
100%|██████████| 547958/547958 [00:08<00:00, 63267.86it/s]
Current Step : 200 / 1  Current Loss : 2.773489
Current Step : 400 / 1  Current Loss : 2.226587
Current Step : 600 / 1  Current Loss : 2.164585
Current Step : 800 / 1  Current Loss : 2.179907
Current Step : 1000 / 1 Current Loss : 1.884135
Current Step : 1200 / 1 Current Loss : 1.987994
Current Step : 1400 / 1 Current Loss : 1.859728
Current Step : 1600 / 1 Current Loss : 2.048557
```

```
문장을 입력하세요. (종료는 exit을 입력하세요.) : 안녕하세요~
안녕하세요
문장을 입력하세요. (종료는 exit을 입력하세요.) : 밥은 먹었어?
아직 안먹었어
문장을 입력하세요. (종료는 exit을 입력하세요.) : 부대찌개 먹을까?
그래서 그런가
문장을 입력하세요. (종료는 exit을 입력하세요.) : 우리 놀러가자!
ㅋㅋㅋㅋㅋㅋㅋ
문장을 입력하세요. (종료는 exit을 입력하세요.) : 영화 보러 갈래요?
영화보고 있어요
문장을 입력하세요. (종료는 exit을 입력하세요.) : 학교 같이 가자!
아 그래?
문장을 입력하세요. (종료는 exit을 입력하세요.) : 지금 뭐하고 있어?
나 일하고 있어
문장을 입력하세요. (종료는 exit을 입력하세요.) : exit
```

Edited by Harksoo Kim

질의응답

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

Edited by Harksoo Kim