대덕특구 SPACE-S X K KOREA 케라스 코리아

DLD 2022 | 대전러닝데이

케라스 NLP를 활용한 GPT 텍스트 생성 도전하기

네이버
최태균

INNOPOLIS 연구개발특구진흥재단 | 따뜻한 데이커 연구소 | AIFactory

시작하기 전에

- 이 강의는 머신러닝과 NLP를 이해할 수 있는 초중급 수준의 내용으로 구성되어 있습니다.
- 강의 중에 잘 모르는 용어나 표현이 있다면 “참고” 링크를 통해서 확인해주세요.

목차

실습 링크

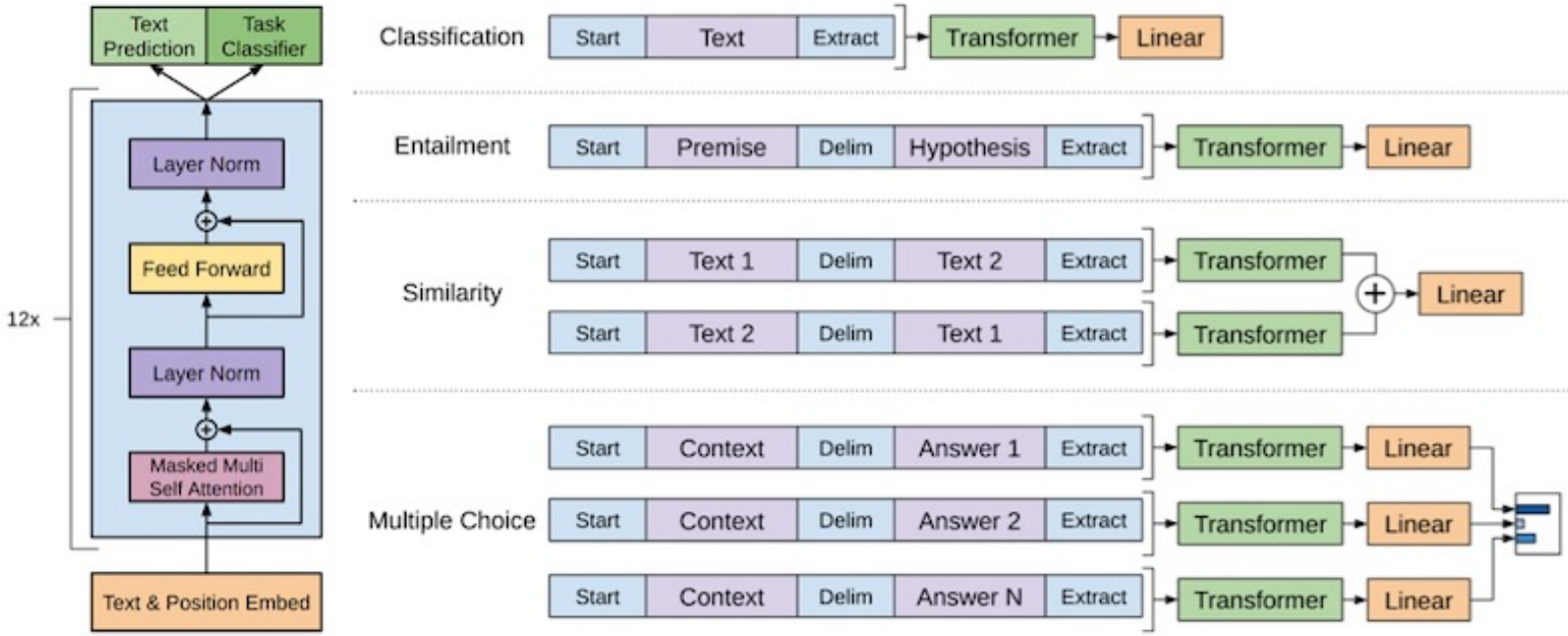
- Colab link: https://bit.ly/3eXFBMI
- Github link: https://bit.ly/3z4Mouz

GPT와 텍스트 생성

- GPT 모델 소개
  - GPT 모델은 Transformer 모델의 Decoder만을 활용한 텍스트 생성 모델입니다. (참고: Transformer 모델)
  - 초기 Pre-trained model을 목적으로 self-supervised 방식의 학습을 하여 기존 NLP task 모델의 성능을 높히고자 했습니다.
  - GPT-3 모델에서 학습 없이 정답 예측을 하는 Few-shot learning이 가능해 지면서 NLP 영역에서 많이 사용되고 있습니다.
    (참고: OpenAI GPT)

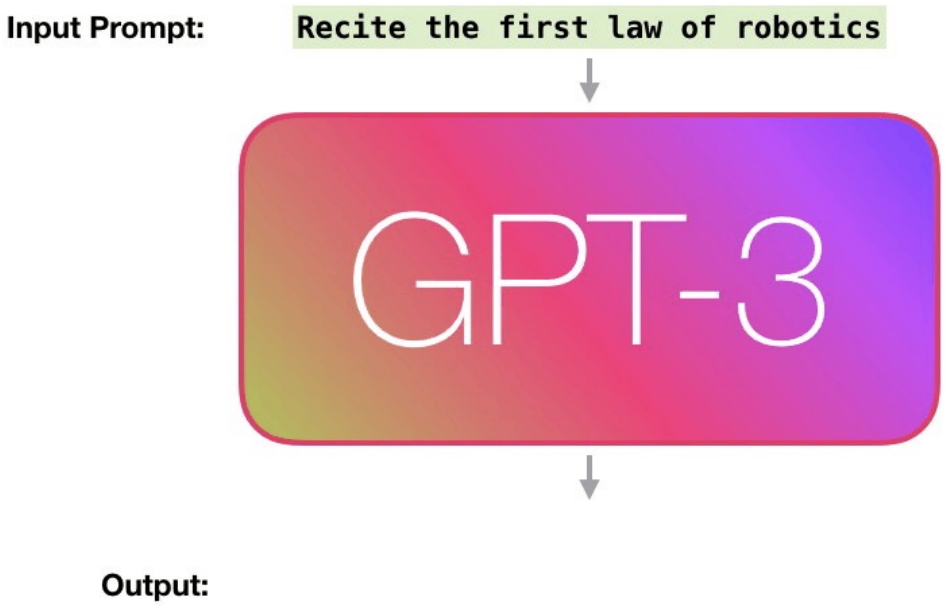
GPT와 텍스트 생성

- GPT 모델 소개



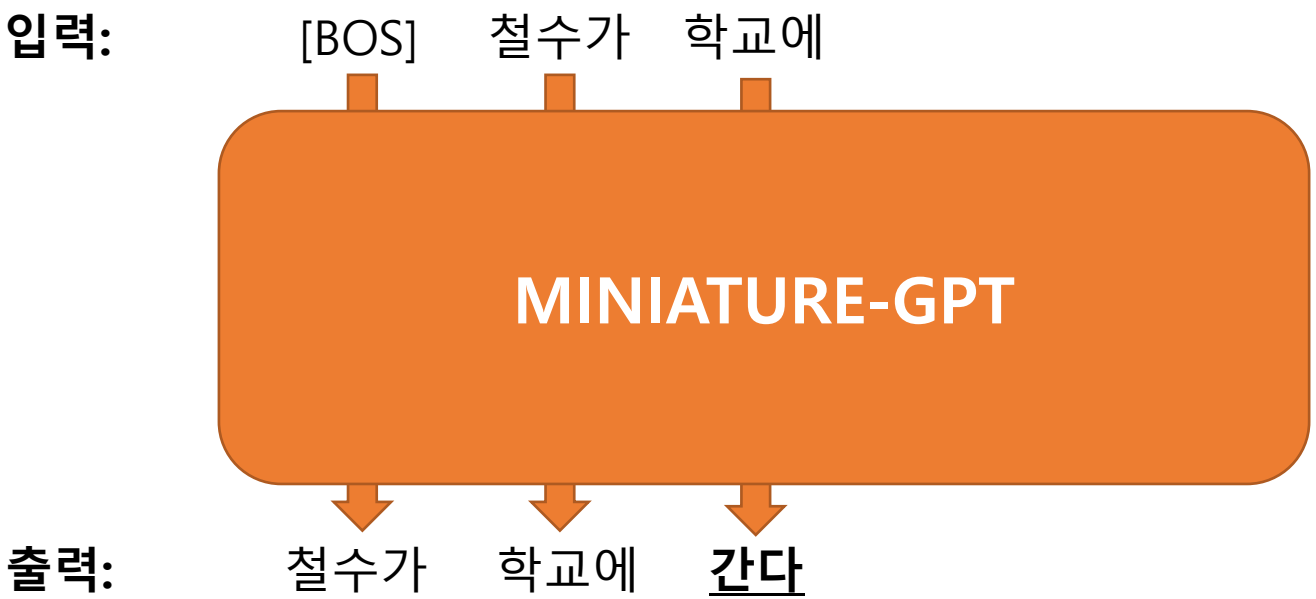
GPT와 텍스트 생성

- GPT 모델 소개



GPT와 텍스트 생성

- GPT를 이용한 텍스트 생성
  - 이번에 실습할 GPT 모델은 미니어쳐 GPT 모델로 기존 논문에 소개된 GPT 모델에 비해 작은 모델입니다. (참고: min-GPT)



실습 준비

- 라이브러리 설치
  - 실습에서는 keras를 활용하기 위해 tensorflow 라이브러리를 설치합니다.
  - 추가로 keras-nlp를 활용하기 때문에 keras-nlp도 같이 설치해줍니다.
    - keras-nlp를 “pip install keras-nlp”로 설치하시면 실습 진행이 안되실수도 있습니다.

```
!pip install tensorflow==2.10.0
!pip install git+https://github.com/keras-team/keras-nlp.git --upgrade
```

실습 준비

- 학습 데이터 준비
  - keras.utils.get_file을 통해서 데이터를 다운로드 받습니다.
  - 학습 데이터는 simplebooks 라는 영어 데이터를 활용할 예정입니다.

```
keras.utils.get_file(
    origin="https://dldata-public.s3.us-east-2.amazonaws.com/
            simplebooks.zip",
    extract=True,
)
dir = os.path.expanduser("~/.keras/datasets/simplebooks/")
```

실습 준비

- 학습 데이터 준비
  - 다운로드 받으신 데이터는 아래와 같은 텍스트로 확인할 수 있습니다.

```
!head -n20 ~/.keras/datasets/simplebooks/simplebooks-2-raw/train.txt
```

```
More Jataka Tales
By
Ellen C. Babbitt
I The Girl Monkey And The String Of Pearls


One day the king went for a long walk in the woods. When he came back to his own garden, he sent for his family to come down to the lake for a swim.
When they were all ready to go into the water, the queen and her ladies left their jewels in charge of the servants, and then went down into the lake.
```

텍스트 전처리

- tf.data를 이용한 전처리하기 (참고: 텐서플로 파이프라인 구성 가이드)
  - 모델 학습을 위해 텍스트를 모델에 입력할 수 있도록 구성합니다.
  - 학습을 위해 실습에서는 tf.data.Dataset를 활용합니다.
  - Dataset을 통해 전처리를 할 수 있는 파이프라인을 만들 수 있습니다.

```
dataset = tf.data.TextLineDataset(dir + "simplebooks-92-raw/train.txt")
print(dataset)
```

```
OUTPUT:
<TextLineDatasetV2 element_spec=TensorSpec(shape=(), dtype=tf.string,
name=None)>
```

텍스트 전처리

- tf.data를 이용한 전처리하기
  - Tf.data.Dataset 객체는 iterable이 가능하기 때문에 for 구문을 적용하여 데이터를 확인 가능합니다.

```
for i, element in enumerate(dataset):
    print(element)
    if i > NUM_ITERS: break;
```

OUTPUT:

```
tf.Tensor(b"Dave Darrin's Second Year At Annapolis", shape=(), dtype=string)
tf.Tensor(b'', shape=(), dtype=string)
tf.Tensor(b'Or', shape=(), dtype=string).
tf.Tensor(b'', shape=(), dtype=string) …
```

텍스트 전처리

- tf.data를 이용한 전처리하기
  - filter 함수 활용하기

```
# 일정 텍스트 길이보다 짧은 텍스트를 필터합니다.
filtered_dataset = dataset.filter(lambda x: tf.strings.length(x) >
                                  MIN_TRAINING_SEQ_LEN)
for i, element in enumerate(filtered_dataset.as_numpy_iterator()):
    print(element)
    if i > NUM_ITERS: break;
```

OUTPUT:

```
b"Readers of that preceding volume will …… of his class."
b'"That scoundrelly Chow …… cleaned out right away."'
b'"Just this," replied …… about my little mishap?"'
```

텍스트 전처리

- tf.data를 이용한 전처리하기
  - batch 함수 활용하기

```
batched_dataset = dataset.filter(lambda x: tf.strings.length(x) >
                                           MIN_TRAINING_SEQ_LEN).batch(2)
for. i, element in enumerate(batched_dataset.as_numpy_iterator()):
    print(element)
    if i > NUM_ITERS: break;
```

OUTPUT:

```
[b"Readers of that Academy, Dave had full …… and respected member of his class.”,
 b'"That scoundrelly Chow made his boast the …… place is cleaned out right away.”’]
[b'"Just this," replied Pennington, hanging his head. …… about my little mishap?”’
 b'"A very excellent reason, Mr. Darrin, and I commend …… members of your class.”’]
```

텍스트 전처리

- 학습 데이터셋 파이프라인 구성하기
  - 앞서 소개한 tf.data 패키지에 있는 모듈을 사용하여 파이프라인을 구성해봅시다.

```
# 학습 데이터셋 파이프라인
raw_train_ds = (
    tf.data.TextLineDataset(dir + "simplebooks-92-raw/train.txt")
            .filter(lambda x: tf.strings.length(x) > MIN_TRAINING_SEQ_LEN)
            .batch(BATCH_SIZE)
            .shuffle(buffer_size=256)
)
```

텍스트 전처리

- 학습 데이터셋 파이프라인 구성하기
  - 앞서 소개한 tf.data 패키지에 있는 모듈을 사용하여 파이프라인을 구성해봅시다.

```
# 평가 데이터셋 파이프라인
raw_val_ds = (
    tf.data.TextLineDataset(dir + "simplebooks-92-raw/valid.txt")
           .filter(lambda x: tf.strings.length(x) > MIN_TRAINING_SEQ_LEN)
           .batch(BATCH_SIZE)
)
```

토큰화와 인덱싱

- 모델에 텍스트 정보를 입력하기 위해선
  - 모델에 입력할 텍스트는 숫자 인덱스로 변환해야 합니다.
  - 인덱스는 텍스트에서 토큰 단위로 쪼개어 숫자로 변환합니다. (토큰화)

"철수는 학교에 갑니다" ➔ ["철수는", "학교에", "갑니다"] ➔ [2, 1, 3]

텍스트 **토큰** **인덱스**

토큰화와 인덱싱

- Wordpiece 토크나이저
  - 토크나이저는 텍스트를 토큰으로 변환하는 역할을 합니다.
    - 일반적으로 띄어쓰기를 이용한 “Space 토크나이저”나 형태소 분석기를 이용한 토크나이저가 있습니다.
  - Wordpiece 토크나이저는 왜 쓰나요?
    - 수많은 텍스트 데이터를 학습하게 되면 인덱싱을 위한 어휘사전의 크기가 굉장히 커집니다.
    - Wordpiece 토크나이저는 기존 어휘들을 쪼개어 어휘사전 수를 줄일 수 있도록 도와줍니다.
    - (참고: Wordpiece 토크나이저 소개)

토큰화와 인덱싱

- Wordpiece 토크나이저 사전 학습하기
  - Keras-nlp에서는 Wordpiece 토크나이저를 만들고 사용할 수 있도록 지원해 줍니다.

```
# 토크나이저 사전을 학습합니다.
vocab = keras_nlp.tokenizers.compute_word_piece_vocabulary(
    raw_train_ds,
    vocabulary_size=VOCAB_SIZE,
    lowercase=True,
    reserved_tokens=["[PAD]", "[UNK]", "[BOS]"],
)
```

토큰화와 인덱싱

- Wordpiece 토크나이저 사전 학습하기
  - Wordpiece 토크나이저를 학습하는데 활용할 토큰들도 정의합니다.
    - [PAD] 토큰: 입력 토큰 길이를 고정했을 때 빈 공간을 채우기 위한 토큰
    - [UNK] 토큰: 사전에 없는 토큰인 경우 채우기 위한 토큰
    - [BOS] 토큰: 문장 또는 텍스트가 시작한다는 신호로 사용하기 위한 토큰

```
# 토크나이저 사전을 학습합니다.
vocab = keras_nlp.tokenizers.compute_word_piece_vocabulary(
    raw_train_ds,
    vocabulary_size=VOCAB_SIZE,
    lowercase=True,
    reserved_tokens=["[PAD]", "[UNK]", "[BOS]"],
)
```

토큰화와 인덱싱

- Wordpiece 토크나이저 사전 학습하기
  - 학습한 토크나이저 사전 결과를 확인할 수 있습니다.

```
print(vocab[:30])
```

OUTPUT:

```
['[PAD]', '[UNK]', '[BOS]', '!', '"', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.',
'/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '=']
```

토큰화와 인덱싱

- Wordpiece 토크나이저 만들기
  - 학습한 Wordpiece 토크나이저 사전을 가지고 토큰화를 할 수 있는 토크나이저로 만들어 봅니다.

```
tokenizer = keras_nlp.tokenizers.WordPieceTokenizer(
    vocabulary=vocab,
    sequence_length=SEQ_LEN,
    lowercase=True
)
```

토큰화와 인덱싱

- Wordpiece 토크나이저 만들기
  - 토크나이저에 텍스트를 입력하면 토큰화된 결과를 확인해 볼 수 있습니다.

```
# 토크나이저를 tokenize만 호출하면 토큰 인덱스 형태로 출력이 됩니다.
print([tokenizer.id_to_token(t) for t in tokenizer.tokenize('hello
       world!').numpy().tolist()])
```

OUTPUT:

```
['he', '##ll', '##o', 'world', '!', '[PAD]', '[PAD]', '[PAD]', '[PAD]’]
```

토큰화와 인덱싱

- 인덱싱 파이프라인 추가하기
  - 앞서 구현한 전처리 파이프라인에 인덱싱을 추가해보도록 합시다.
  - GPT에서는 항상 텍스트 입력 시작에는 [BOS] 토큰을 추가해줍니다.
    - Keras-nlp에서는 StartEndPacker 클래스를 통해서 간단히 처리할 수 있습니다.

```
start_packer = keras_nlp.layers.StartEndPacker(
    sequence_length=SEQ_LEN,
    start_value=tokenizer.token_to_id("[BOS]"),
)
```

토큰화와 인덱싱

- 인덱싱 파이프라인 추가하기
  - 생성한 StartEndPacker 인스턴스를 호출하면 [BOS] 인덱스가 앞에 추가된 것을 확인할 수 있습니다.
    - [BOS] 인덱스 값은 2 입니다.

```
tokens = tokenizer('hello world')
start_packer(tokens)
```

OUTPUT:

```
<tf.Tensor: shape=(128,), dtype=int32, numpy= array([ 2, 103, 1520, 291, 394, 0, 0, 0, 0],
dtype=int32)>
```

토큰화와 인덱싱

- 인덱싱 파이프라인 추가하기
  - 인덱싱 파이프라인으로 추가할 전처리 함수를 만들어 봅시다.
  - 앞서 구현한 토크나이저와 StartEndPacker를 같이 처리할 수 있도록 구현합니다.
  - 함수출력은 모델 학습에 필요한 입력 인덱스와 정답 레이블로 합니다.

```
def preprocess(inputs):
    outputs = tokenizer(inputs) # 토큰화와 인덱싱을 합니다.
    features = start_packer(outputs) # “[BOS]” 인덱스를 추가합니다.
    labels = outputs # 정답 레이블을 구성해줍니다.
    return features, labels
```

토큰화와 인덱싱

- 인덱싱 파이프라인 추가하기
  - 전처리 함수에 텍스트를 입력한 결과는 다음과 같습니다.

```
preprocess("This is test sentence.")
```

OUTPUT:

```
(<tf.Tensor: shape=(128,), dtype=int32, numpy= array([ 2, 139, 124, 56, 410, 375, 1121, 15,
 0, 0, 0, 0, 0, 0], dtype=int32)>,
 <tf.Tensor: shape=(128,), dtype=int32, numpy= array([ 139, 124, 56, 410, 375, 1121, 15, 0,
 0, 0, 0, 0, 0, 0], dtype=int32)>)
```

토큰화와 인덱싱

- 인덱싱 파이프라인 추가하기
  - 앞서 전처리 파이프라인을 구성한 학습, 평가 dataset 객체에 map 함수를 호출하여 인덱싱 파이프라인을 구성합니다.

```
# prefetch 함수는 파이프라인 맨 마지막에 붙혀줍니다.
train_ds = raw_train_ds.map(preprocess,
    num_parallel_calls=tf.data.AUTOTUNE).prefetch(
    tf.data.AUTOTUNE
)
val_ds = raw_val_ds.map(preprocess,
    num_parallel_calls=tf.data.AUTOTUNE).prefetch(
    tf.data.AUTOTUNE
)
```

GPT 모델 구성과 학습

- 모델 구성하기
  - GPT 모델은 3개의 레이어로 구분하여 구현합니다.
    - **임베딩 레이어:** 입력 텍스트에 대해 토큰과 위치에 대한 임베딩을 출력합니다.
    - **디코더 레이어:** 입력 텍스트 뒤에 생성할 다음 토큰 임베딩을 출력합니다.
    - **Logit 레이어:** 입력한 텍스트에 대한 다음 토큰 인덱스의 logit값을 출력합니다.

GPT 모델 구성과 학습

- 모델 입력 레이어 구성하기
  - 먼저 모델을 구성하기 위해 입력에 대해 정의를 합니다.
  - Keras에서는 모델을 구성하기 위해 Input을 반드시 생성해줘야 합니다.

```
inputs = keras.layers.Input(shape=(None,), dtype=tf.int32)
```

GPT 모델 구성과 학습

- 임베딩 레이어 구성하기
  - GPT 모델에서는 토큰에 대한 임베딩과 토큰 위치에 대한 임베딩을 활용합니다.
  - Keras-nlp에서는 TokenAndPositionEmbedding을 통해 두 임베딩을 모두 얻을 수 있습니다.

```
embedding_layer = keras_nlp.layers.TokenAndPositionEmbedding(
    vocabulary_size=VOCAB_SIZE,
    sequence_length=SEQ_LEN,
    embedding_dim=EMBED_DIM,
    mask_zero=True, # [PAD] 토큰에 대해 마스킹을 해줍니다.
)
```

GPT 모델 구성과 학습

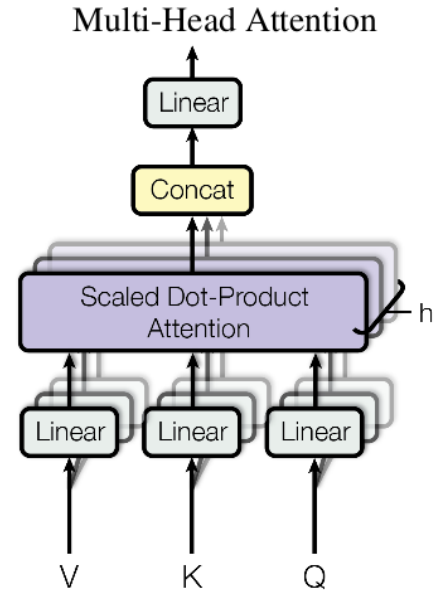
- 임베딩 레이어 구성하기
  - 그리고 앞서 만든 Input을 생성한 임베딩 레이어의 입력으로 넣어줍니다.

```
x = embedding_layer(inputs)
```

GPT 모델 구성과 학습

- 디코더 레이어 구성하기
  - GPT에서 디코더 레이어는 Transformer 모델의 디코더 레이어를 활용 합니다.
  - Keras-nlp에서 제공하는 TransformerDecoder를 이용하여 간편하게 디코더 레이어를 구성할 수 있습니다.

```
for _ in range(NUM_LAYERS):
    decoder_layer = keras_nlp.layers.TransformerDecoder(
            num_heads=NUM_HEADS, # 어텐션 head 수 (h)
            intermediate_dim=FEED_FORWARD_DIM, # 입출력 차원 수
    )
    x = decoder_layer(x)
```



GPT 모델 구성과 학습

- Logit 레이어 구성하기
  - 디코더 레이어에서 출력한 임베딩 벡터를 토큰 인덱스로 변환하기 위해 Logit 레이어를 만듭니다.
  - Logit 레이어는 Dense 레이어를 이용하여 각 토큰 인덱스의 logit값을 출력합니다.

```
outputs = keras.layers.Dense(VOCAB_SIZE)(x)
```

GPT 모델 구성과 학습

- 모델 학습을 위한 준비
  - 앞서 구성한 레이어를 하나의 모델 객체로 구성해줘야 합니다.
  - keras.Model을 생성하여 학습할 GPT 모델 객체를 생성합니다.
    - keras.Model은 입력 레이어와 출력 레이어만 입력해주면 모델 구성이 가능합니다.

```
model = keras.Model(inputs=inputs, outputs=outputs)
```

GPT 모델 구성과 학습

- 모델 학습을 위한 준비
  - 모델 학습을 위해 로스 함수와 평가 지표(metric) 함수를 같이 정의합니다.
    - 로스 함수는 주로 NLL(Negative Log Likelihood) Loss나 CrossEntropy를 사용합니다.
    - 텍스트 생성을 위한 평가 지표 함수는 보통 Perplexity나 BLEU, ROUGE를 이용합니다.

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
# perplexity에서 mask_token_id를 설정하면 [PAD] 토큰을 점수에 반영하지 않습니다.
perplexity = keras_nlp.metrics.Perplexity(from_logits=True,
                                          mask_token_id=0)
```

GPT 모델 구성과 학습

- 모델 학습을 위한 준비
  - 이제 모델 학습을 위해 컴파일을 해봅시다.
    - 모델 학습에서 optimizer는 adam optimizer를 설정합니다.

```
model.compile(optimizer="adam", loss=loss_fn, metrics=[perplexity])
```

GPT 모델 구성과 학습

- 모델 학습을 위한 준비
  - 모델 구조를 확인하기 위해 summary 함수를 호출해봅시다.

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

 token_and_position_embeddin  (None, None, 256)        1312768
 g (TokenAndPositionEmbeddin
 g)

 transformer_decoder (Transf  (None, None, 256)        394749
 ormerDecoder)

 transformer_decoder_1 (Tran  (None, None, 256)        394749
 sformerDecoder)

 dense (Dense)               (None, None, 5000)        1285000

=================================================================
Total params: 3,387,266
Trainable params: 3,387,266
Non-trainable params: 0
_________________________________________________________________
```

GPT 모델 구성과 학습

- 모델 학습
  - model.fit 함수를 통해 간단하게 학습을 진행할 수 있습니다.
    - 지금까지 구현한 학습/평가 dataset 객체를 같이 입력하여 학습을 합니다.

```
model.fit(train_ds, validation_data=val_ds, verbose=2, epochs=EPOCHS)
```

OUTPUT:

```
Epoch 1/6 3169/3169 - 238s - loss: 4.5088 - perplexity: 96.2405 - val_loss: 4.0023 -
val_perplexity: 62.5531 - 238s/epoch - 75ms/step
Epoch 2/6 3169/3169 - 229s - loss: 4.0025 - perplexity: 57.6544 - val_loss: 3.8421 -
val_perplexity: 52.8804 - 229s/epoch - 72ms/step
Epoch 3/6 3169/3169 - 233s - loss: 3.8910 - perplexity: 51.4962 - val_loss: 3.8000 -
val_perplexity: 50.4783 - 233s/epoch - 74ms/step
```

텍스트 생성

- keras_nlp.utils 를 활용하여 텍스트 생성하기
  - 이번 실습에서는 keras-nlp에서 제공하는 함수들을 이용하여 텍스트를 생성하고자 합니다.
  - 텍스트 생성은 그리디 서치서 부터 Top-P 서치까지 다양한 방법으로 접근하여 할 수 있습니다.

텍스트 생성

- 텍스트 생성 로직
  - 학습한 모델을 가지고 텍스트 생성을 하기 위해서 입력 시작 토큰으로 [BOS] 토큰을 준비 합니다.

```
prompt_tokens = tf.convert_to_tensor([tokenizer.token_to_id("[BOS]")])
```

텍스트 생성

- 텍스트 생성 로직
  - 텍스트 생성을 위한 모델 인퍼런스를 하는 과정은 함수로 구현해 보도록 합시다.
    - 함수 출력은 텍스트 입력 토큰 다음에 생성될 토큰을 출력합니다.

```
def token_logits_fn(inputs):
    cur_len = inputs.shape[1]
    output = model(inputs)
    return output[:, cur_len - 1, :]
```

텍스트 생성

- 그리디 서치
  - 그리디 서치는 모델에서 출력된 logit값 중에서 가장 높은 토큰을 출력하는 방법입니다.
  - Keras-nlp에서는 greedy_search 함수를 제공하여 텍스트 생성을 할 수 있도록 해줍니다.

```
output_tokens = keras_nlp.utils.greedy_search(
    token_logits_fn,
    prompt_tokens,
    max_length=NUM_TOKENS_TO_GENERATE, # 지정한 길이만큼 텍스트 생성을 합니다.
)
```

텍스트 생성

- 그리디 서치
  - 그리디 서치를 통해 간단하게 텍스트 생성을 확인해 볼 수 있습니다.
  - 하지만, 이 방식은 금방 생성 토큰이 반복되는 문제가 있습니다.

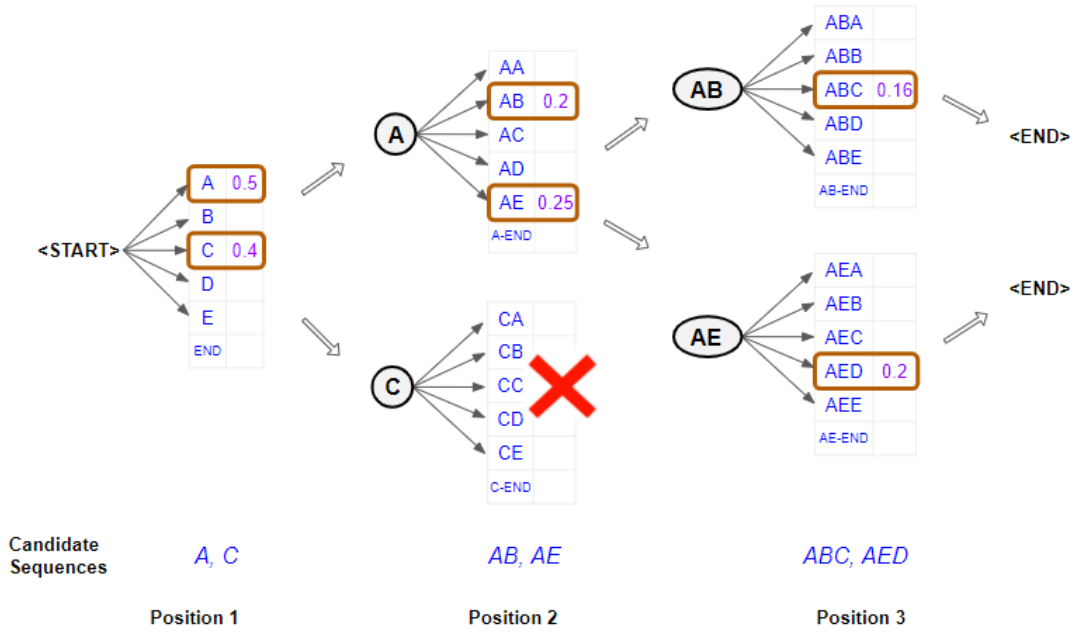
```
txt = tokenizer.detokenize(output_tokens)
print(f"Greedy search generated text: \n{txt}\n")
```

OUTPUT:

```
Greedy search generated text:
b'[BOS] " i \' m not going to be a'
```

텍스트 생성

- 빔 서치
  - 빔 서치는 일정 토큰 시퀀스 길이 안에 있는 토큰 확률들을 조합하여 가장 높은 확률의 생성 토큰을 찾는 방식 입니다.



텍스트 생성

- 빔 서치
  - Keras-nlp의 beam_search 함수를 이용하여 빔 서치 방식으로 텍스트 생성을 할 수 있습니다.
    - 빔 서치를 하고자 하는 사이즈 num_beam이 1이면 그리디 서치와 동일 합니다.

```
output_tokens = keras_nlp.utils.beam_search(
    token_logits_fn,
    prompt_tokens,
    max_length=NUM_TOKENS_TO_GENERATE,
    num_beams=10,
    from_logits=True,
)
```

텍스트 생성

- 빔 서치
  - 빔 서치의 경우 num_beam의 크기가 클수록 그만한 토큰 시퀀스 확률을 가지고 연산을 하기 때문에 연산비용이 커질 수 있습니다.

```
txt = tokenizer.detokenize(output_tokens)
print(f"Beam search generated text: \n{txt}\n")
```

OUTPUT:

```
Beam search generated text:
b'[BOS] " i don \' t want to tell you'
```

텍스트 생성

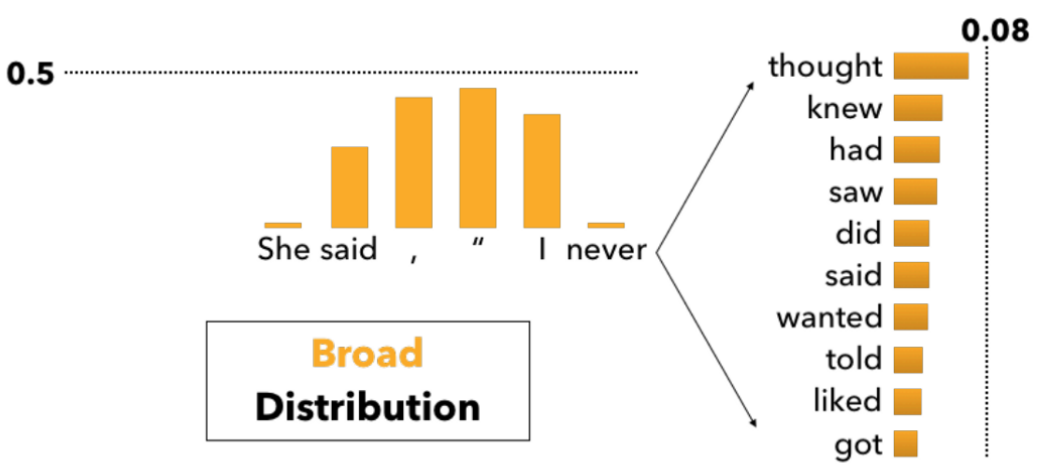
- Top-K 서치
  - 생성한 토큰 확률 분포에서 상위 K 개에 대한 토큰들 중 랜덤으로 샘플링 하는 방식입니다.

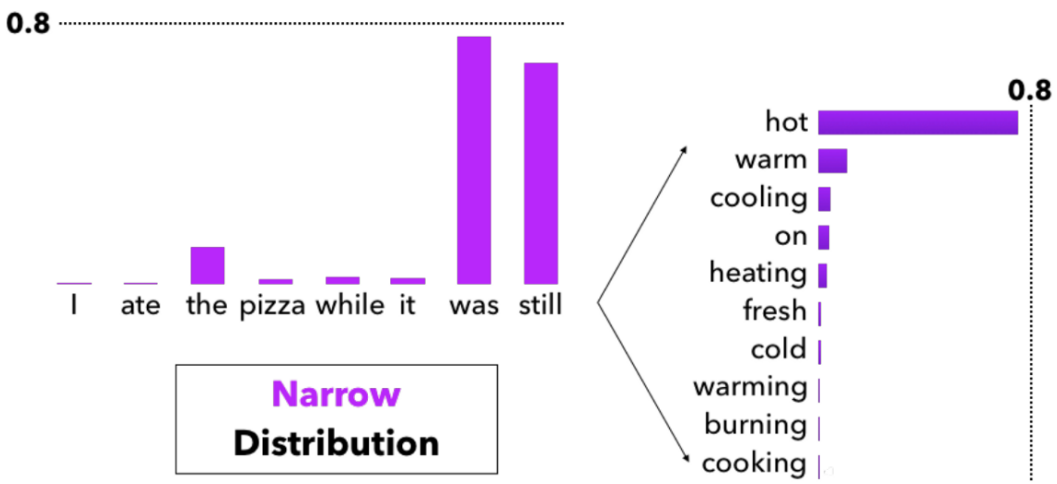
```
output_tokens = keras_nlp.utils.top_k_search(
    token_logits_fn,
    prompt_tokens,
    max_length=NUM_TOKENS_TO_GENERATE,
    k=10,
    from_logits=True,
)
```

이미지 출처: AllenAI AI2 Blog

텍스트 생성

- Top-P 서치
  - 생성한 토큰의 확률 분포는 항상 비슷할 수 없습니다.
  - Top-K 서치를 적용하게 되면 확률 분포의 형태에 따라 텍스트 생성 퀄리티가 영향을 받을 수 있습니다.



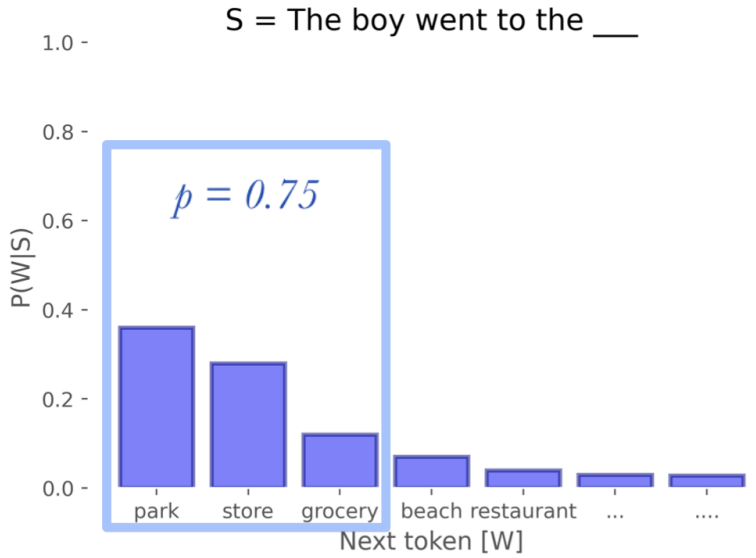


이미지 출처: The Curious Case of Neural Text Degeneration, Holzman et al.

텍스트 생성

- Top-P 서치
  - 생성한 토큰 확률 분포에서 상위 누적 분포가 P인 토큰들 중 랜덤으로 샘플링 하는 방식입니다.

```
output_tokens = keras_nlp.utils.top_p_search(
    token_logits_fn,
    prompt_tokens,
    max_length=NUM_TOKENS_TO_GENERATE,
    p=0.5,
    from_logits=True,
)
```



이미지 출처: AllenAI AI2 Blog

정리

- GPT 모델과 텍스트 생성
- 학습 데이터 전처리를 위한 파이프라인 구성
- GPT 모델 레이어 구성과 학습 방법
- 텍스트 생성을 위한 방식

질의응답

채용소개

<End of Document>
감사합니다.