11 - d 위치 포지셔닝

11 - c 까지는 문맥만 고려하고 단어의 순서는 고려하지 않음 (워드 임베딩 만 사용) → 단어의 순서를 고려하기 위해서 위치 임베딩을 추가로(positional encoding)을 사용한다.

사인, 코사인 등의 주기 함수에서 같은 축에 위치한 값들이 존재하므로 이들의 연관성을 고려하여 학습한다.

page 4 ~ 5

코드 11-24 서브클래싱으로 위치 임베딩 구현하기

```
class PositionalEmbedding(layers.Layer):
    def __init__(self, sequence_length, input_dim, output_dim, **kwargs):
        super().__init__(**kwargs)                    위치 임베딩의 단점은 시퀀스 길이를 미리 알아야 한다는 것입니다.
        self.token_embeddings = layers.Embedding(     토큰 인덱스를 위한 Embedding 층을 준비합니다.
            input_dim=input_dim, output_dim=output_dim)
        self.position_embeddings = layers.Embedding(
            input_dim=sequence_length, output_dim=output_dim)
        self.sequence_length = sequence_length        토큰 위치를 위한 Embedding 층을 준비합니다.
        self.input_dim = input_dim
        self.output_dim = output_dim

    def call(self, inputs):
        length = tf.shape(inputs)[-1]   (None, sequence_length)  [234, 567, 890, ..., 0, 0], # 600개의 Sequence 1
        positions = tf.range(start=0, limit=length, delta=1)
        embedded_tokens = self.token_embeddings(inputs)   (32, 600, 1000), batch=32일 때
        embedded_positions = self.position_embeddings(positions)   (600, 1000)
        return embedded_tokens + embedded_positions
```

imdb를 예시로 생각하자,

input_dim은 **단어 사전의 개수**이다.

output_dim - 트랜스포머에 1000개의 차원이 들어간다는 의미이다.

**kwargs

토큰 임베딩

포지셔닝 임베딩 둘다 1000개의 차원으로 같음

length = 문장 단어의 개수, inputs은 들어오는 단어들(문장)
ft.shape(inputs)[-1]은 실제로는 배치로 들어오기 때문에 (None, sequence_length) 에서 -1번째를 가지고 오기 위함이다.
positions = 0부터 599까지의 포지션이 존재하게 된다. 델타 1은 포지션을 정할 때 단어 하나씩 건너 뛴다는 걸 의미한다.

embedded_positions에는 배치가 없다. 배치가 들어올 때마다 값을 주면 되니까 배치가 필요 없다.

```
        return embedded_tokens + embedded_positions ···· 두 임베딩 벡터를 더합니다.

    def compute_mask(self, inputs, mask=None):
        return tf.math.not_equal(inputs, 0)

    def get_config(self): ···· 모델 저장을 위한 직렬화를 구현합니다.
        config = super().get_config()
        config.update({
            "output_dim": self.output_dim,
            "sequence_length": self.sequence_length,
            "input_dim": self.input_dim,
        })
        return config
```

Embedding 층처럼 이 층은 입력에 있는 0 패딩을 무시할 수 있도록 마스킹을 생성해야 합니다. compute_mask 메서드는 프레임워크에 의해 자동으로 호출되고 만들어진 마스킹은 다음 층으로 전달됩니다.

두 임베딩 벡터 (토큰, 포지션)을 합친다.
단어 인덱스의 임베딩을 학습하는 것처럼 위치 임베딩 벡터를 학습하고 합쳐서 위치를 고려한 단어 임베딩을 만든다.

compute_mask는 제로 패딩 관련 함수

page 6 ~ 7

코드 11-25 트랜스포머 인코더와 위치 임베딩 합치기

```
vocab_size = 20000
sequence_length = 600
embed_dim = 256
num_heads = 2
dense_dim = 32

inputs = keras.Input(shape=(None,), dtype="int64")                    여기가 위치 임베딩입니다!
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(inputs)
x = TransformerEncoder(embed_dim, dense_dim, num_heads)(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dropout(0.5)(x)
```

embed_dim이 1000이 아니라 256인 이유 (Linear Layer를 거쳐 아웃풋이 256개의 차원으로 줄어들 기 때문에 256이다.

dense_dim = 노드의 개수 32개

오버피팅 방지를 위해 드롭아웃을 사용하고 드롭 아웃을 사용한 것중에 max를 구한다?

```
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()
callbacks = [
    keras.callbacks.ModelCheckpoint("full_transformer_encoder.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=20, callbacks=callbacks)
model = keras.models.load_model(
    "full_transformer_encoder.keras",
    custom_objects={"TransformerEncoder": TransformerEncoder,
                    "PositionalEmbedding": PositionalEmbedding})
print(f"테스트 정확도: {model.evaluate(int_test_ds)[1]:.3f}")
```