[Assignment#1, #2]

RNN, LSTM, SelfAttention을 원하는 문제에 활용하여 문제 해결하고 결과에 대한 고찰 및 분석하기**.**

소프트웨어학부 **20195298** 박준용

목표

예제를 통해 **NLP(RNN, LSTM, SelfAttention)** 모델에 대해서 이해하기

- **RNN**

  RNN은 순차적인 데이터를 처리하기에 적합한 모델로 각 단어를 순차적으로 입력으로 받고**,** 내부의 순환 구조를 통해 이전 단계의 정보를 현재 단계로 전달한다**.** 이전 단계의 정보가 현재 단계에 영향을 미치기 때문에 문맥을 파악하는 데 유용하지만 장기 의존성 문제가 있다**.**

- **LSTM**

  LSTM은 **RNN**의 변형 모델로**, RNN**의 단점인 장기 의존성 문제를 해결하기 위해 고안되었다**. LSTM**은 게이트 메커니즘을 도입하여**,** 어떤 정보를 기억하고 어떤 정보를 잊을지를 결정한다**.** 이를 통해 장기 의존성을 유지하면서 필요한 정보를 기억할 수 있다**.**

- **SelfAttention**

  Self-Attention은 입력 시퀀스의 각 단어 간의 상호 관계를 고려하여 중요한 단어를 강조하는 방식으로 작동한다**.IMDB** 예제에서 **Self-Attention** 모델은 각 단어의 임베딩을 입력으로 받고**, Self-Attention** 계산을 통해 통해 중요한 단어에 더 큰 가중치를 부여하고**,** 이를 바탕으로 긍정 또는 부정을 예측하는 출력층을 추가할 수 있다**.**

Task

IMDB(Internet Movie Data Base)데이터를 이용하여 영화에 대한 리뷰 텍스트를 학습하고 해당 리뷰가 긍정적인지 부정적인지 판별

- **IMDM** 데이터 구조

정보	값
훈련용 데이터	**25000**
테스트 데이터	**25000**
카테고리**(**긍정**,** 부정**)**	**2**
리뷰의 최대 길이	**2494**
리뷰의 평균 길이	약 **238**

각 리뷰는 텍스트 형식으로 구성되어 있으며**,** 영화 리뷰의 내용을 담고 있다**.** 리뷰 내용은 자연어로 작성된 문장 또는 문단으로 이루어져 있다**.** 각 리뷰에는 해당하는 라벨**(**긍정 또는 부정**)**이 지정되어 있으며**.** 긍정적인 리뷰는 **1**로 라벨링되고**,** 부정적인 리뷰는 **0**으로 라벨링되어 있다**.**

모델 정의

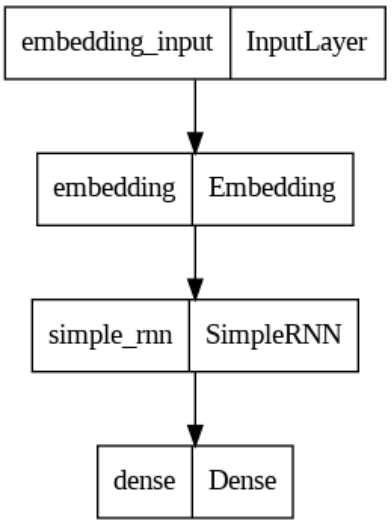
RNN

```
def rnn_model(): #-- RNN Model
   model = Sequential()
   model.add(Embedding(vocab_size, embedding_dim))
   model.add(SimpleRNN(hidden_units)) #-- SimpleRNN 레이어: RNN의 한 종류로, 시계열 데이터 처리에 사용됨
   model.add(Dense(1, activation='sigmoid')) #-- Dense 레이어: 이진 분류를 위한 출력층
      …
```

RNN모델은 **TensorFlow**의 **Keras**에서 제공하는 **SimpleRNN** 모델을 사용했다**. SimpleRNN**은 순환 신경망**(Recurrent Neural Network, RNN)**의 한 종류로**,** 시퀀스 데이터를 처리하는 데 사용되며 이 모델은 이전 상태를 저장하고 현재 입력과 함께 사용하여 다음 상태를 생성하는 반복적인 구조를 가지고 있다**. SimpleRNN** 레이어를 통과한 이후 최종적으로 이진분류 모델이므로 **simgmoid**를 사용하여 출력할 수 있도록 모델을 정의했다**.**

—# SimpleRNN

1. 입력 데이터**:** **SimpleRNN**은 시퀀스 데이터를 입력으로 받는다**.**
2. 상태**(**은닉 상태**, Hidden State): SimpleRNN**은 각 타임스텝에서 상태**(**은닉 상태**)**를 유지하며 이전 타임스텝의 출력 값을 기반으로 갱신되며**,** 현재 타임스텝의 정보를 저장하는 역할을 한다**.**
3. 순차적 계산**: SimpleRNN**은 타임스텝마다 입력 데이터와 이전 타임스텝의 상태를 조합하여 새로운 상태를 생성 하고 선형 변환한 후**,** 활성화 함수**(tanh)**를 통과시키며 활성화 함수를 거친 결과는 새로운 상태**(**은닉 상태**)**가 된다**.**
4. 출력**:** 각 타임스텝에서 상태**(**은닉 상태**)**를 출력한다**.** 이 출력은 다음 타임스텝으로 전달된다
5. 시퀀스의 마지막 타임스텝**:** 시퀀스의 마지막 타임스텝에서 최종 출력을 반환한다**.**



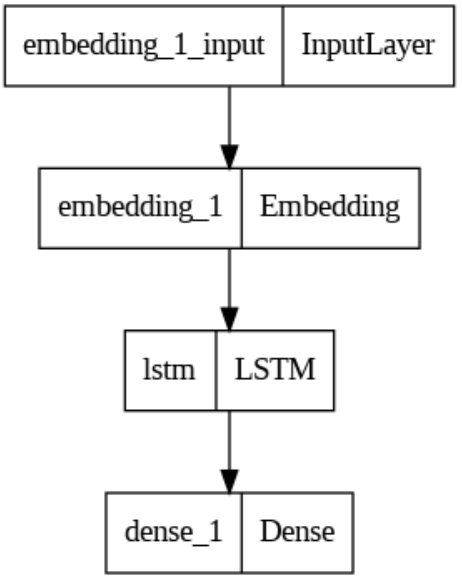
LSTM

```
def lstm_model():
    model = Sequential() #-- LSTM 모델 생성
    model.add(Embedding(vocab_size, embedding_dim)) #-- Embedding 층 추가: 입력 데이터를 임베딩 벡터로 변환
    model.add(LSTM(hidden_units))#-- LSTM 층 추가: LSTM 레이어를 모델에 추가
    model.add(Dense(1, activation='sigmoid'))#-- Dense 층 추가: 이진 분류를 위한 출력층
```

LSTM 모델은 **TensorFlow**에서 제공하는 **LSTM** 모델을 사용했다.
LSTM(Long Short-Term Memory)은 **RNN(Recurrent Neural Network)**의 한 종류로, 시퀀스 데이터를 처리하는 데 사용된다. **LSTM**은 게이트를 이용하여 일반적인 **RNN**보다 긴 시퀀스에서 장기 의존성을 더 잘 학습할 수 있도록 설계되어 있다

—# LSTM

1. **LSTM**의 구조:
    - **LSTM**은 기본적으로 입력 게이트**(input gate)**, 삭제 게이트**(forget gate)**, 출력 게이트**(output gate)**로 구성된 셀**(cell)**로 이루어져 있다.
    - 입력 게이트는 현재 입력을 얼마나 반영할지 결정
    - 삭제 게이트는 이전 상태로부터 얼마나 많은 정보를 삭제할지 결정
    - 출력 게이트는 현재 상태를 다음 타임스텝으로 얼마나 많이 출력할지 결정
2. 셀의 동작:
    - **LSTM**은 이전 상태를 기억하고 새로운 정보를 기반으로 상태를 갱신
    - 각 타임스텝에서, 현재 입력과 이전 타임스텝의 상태가 셀에 입력으로 제공
    - 입력 게이트는 현재 입력과 이전 상태를 조합하여 새로운 정보를 생성
    - 삭제 게이트는 이전 상태로부터 얼마나 많은 정보를 유지할지를 결정
    - 출력 게이트는 현재 상태를 다음 타임스텝으로 출력할지를 결정
3. 역전파와 학습:
    - **LSTM**은 역전파 알고리즘을 사용하여 손실 함수의 기울기를 계산하고 가중치를 업데이트
    - 역전파를 통해 시퀀스의 각 타임스텝에 걸친 기울기가 전파되며, 모델이 학습 데이터에 적합하도록 조정



Self Attention

```
class MultiHeadAttention(tf.keras.layers.Layer):

class TransformerBlock(tf.keras.layers.Layer):

class TokenAndPositionEmbedding(tf.keras.layers.Layer):
```

Self-Attention은 입력의 서로 다른 위치 간의 의존성을 파악하는 데에 효과적인 메커니즘이다. **MultiHead Attention** 클래스와 **Transformer Block** 클래스를 통해 **Self-Attention**이 구현되고, **TokenAndPositionEmbedding** 클래스를 통해 입력에 토큰과 위치 임베딩이 조합한다. 이렇게 구성된 모델은 입력 데이터의 시퀀스 내에서 의미 있는 상호작용을 찾고, 그에 따라 출력을 예측하게 된다.

—# Self Attention

Self Attention Class

- **MultiHead Attention** 클래스

MultiHead Attention 클래스는 어텐션 메커니즘을 사용하여 입력 시퀀스 내의 상호 의존성을 파악하는 역할을 수행한다. 클래스 내의 **scaled_dot_product_attention** 메서드는 입력된 쿼리**(Query)**, 키**(Key)**, 값**(Value)**에 대해 어텐션 가중치를 계산한다. 어텐션 가중치는 쿼리와 키 사이의 유사도를 계산하고, 유사도에 소프트맥스 함수를 적용하여 정규화한다.이렇게 계산된 가중치를 사용하여 값에 가중합을 적용하여 어텐션 출력을 계산한다.**split_heads** 메서드는 입력 텐서를 헤드 수로 분할하여 어텐션 연산을 수행하기 위한 형태로 변환하고 **call** 메서드는 입력을 받고, 쿼리, 키, 값으로 변환한 후 멀티 헤드 어텐션을 수행하여 결과를 반환한다.

- **Transformer Block** 클래스

TransformerBlock 클래스는 트랜스포머의 핵심 블록을 구현한다. 클래스 내에는 멀티 헤드 어텐션**(MultiHeadAttention)**과 포지션 와이즈 피드 포워드 신경망**(Feed-Forward Neural Network)**가 포함된다. **call** 메서드에서는 입력을 받고, 멀티 헤드 어텐션과 포지션 와이즈 피드 포워드 신경망을 순차적으로 적용하고, 최종적으로 레이어 정규화를 적용한 결과를 반환한다

- **TokenAndPositionEmbedding** 클래스

TokenAndPositionEmbedding 클래스는 입력 시퀀스의 토큰과 위치 정보를 임베딩하여 결합하는 역할을 수행한다. 클래스 내의 **call** 메서드는 입력을 받고, 토큰 임베딩과 위치 임베딩을 계산하여 결합한 후 반환한다. 토큰 임베딩은 임베딩 레이어를 통해 입력 토큰을 고차원 벡터로 변환하고, 위치 임베딩은 입력 시퀀스의 위치 정보를 포지션 임베딩 레이어를 통해 임베딩하고 결과로 토큰 임베딩과 위치 임베딩을 더한 값을 반환한다.

클래스	역할
MultiHead Attention	입력 시퀀스 내의 상호 의존성을 파악하고 가중치를 계산하는 역할
Transformer Block	멀티 헤드 어텐션과 포지션 와즈피드 포워드 신경망을 결합하여 입력 시퀀스의 표현을 변환
TokenAndPositionEmbedding	입력 시퀀스의 토큰과 위치 정보를 임베딩하여 결합하는 역할

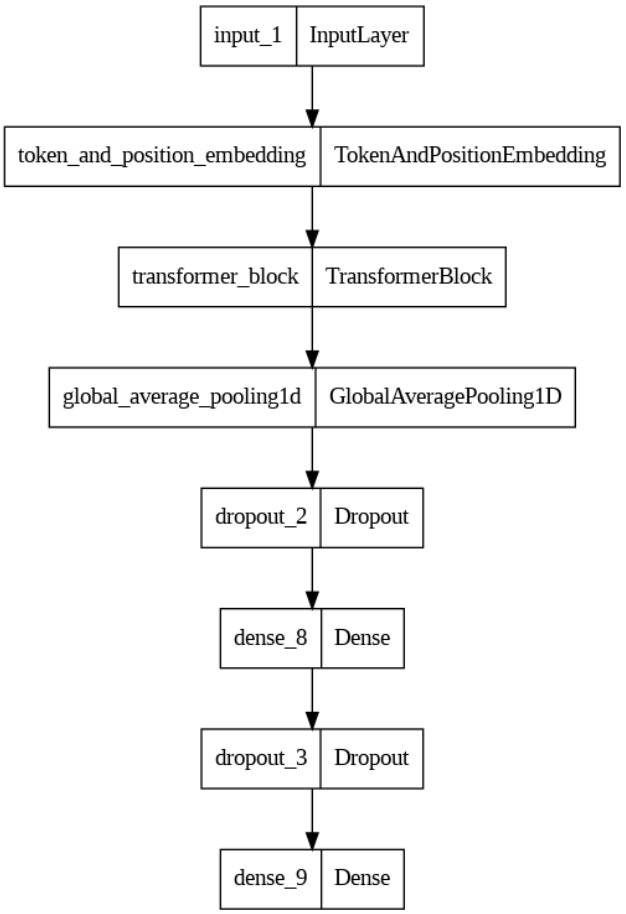
Self Attention Function

1. 입력 시퀀스
   - 입력 시퀀스는 임베딩 과정을 통해 고차원 벡터로 변환하고 이 임베딩 벡터는 단어의 의미적인 특성을 포착한 표현

2. **Query, Key, Value** 생성:
   - 입력 시퀀스의 임베딩된 벡터를 이용하여 **Query, Key, Value**를 생성
   - **Query** : 각 위치(단어)에서의 쿼리 벡터로, 해당 위치와 다른 모든 위치들 간의 유사도를 측정하는 데 사용
   - **Key** : 각 위치(단어)에서의 키 벡터로, 유사도 측정 시 쿼리와 비교되는 대상이 됨
   - **Value** : 각 위치(단어)에서의 값 벡터로, 어텐션 가중치를 적용하여 최종 출력을 계산하는 데 사용

3. 어텐션 가중치 계산:
   - **Query**와 **Key** 벡터 간의 유사도를 계산한다. 일반적으로는 내적**(dot product)**을 사용
   - 유사도를 측정한 후 소프트맥스 함수를 적용하여 어텐션 가중치를 얻음.
   - 어텐션 가중치는 각 단어의 중요도를 나타내며, 값 벡터에 가중합을 적용하여 단어 간 상호 작용을 반영한 결과를 얻음

4. 출력 계산:
   - 어텐션 가중치를 값 벡터에 적용하여 최종 출력을 계산
   - 각 단어의 값 벡터에 어텐션 가중치를 곱한 후, 이를 합산하여 최종 출력 벡터 결정
   - 최종 출력 벡터는 입력 시퀀스의 모든 단어 간의 상호 작용을 반영한 결과



모델 학습

RNN

Epoch	손실값(Training)	정확도(Training)	손실값(Validation)	정확도(Validation)
1	0.6692	0.5687	0.6345	0.6106
2	0.5006	0.7600	0.4403	0.8120
3	0.4402	0.8005	0.6201	0.6360
4	0.4305	0.8083	0.4994	0.7570
5	0.3888	0.8371	0.4788	0.7976
6	0.3470	0.8557	0.4513	0.8210

RNN 학습 결과 분석

Epoch 1에서부터 **Epoch 6**까지의 훈련 및 검증 정확도를 살펴보면 훈련 정확도가 점차 상승하고 있음을 알 수 있다. 이는 모델이 훈련 데이터에 대해 점진적으로 더 잘 맞추고 있다는 것을 의미한다.

검증 정확도도 **Epoch 1**부터 상승하고 있지만, **Epoch 3** 이후에는 큰 변화가 없고 일정하게 유지되고 있다.

Early Stopping이 **Epoch 6**에서 작동하여 훈련이 조기 종료되었고, 이는 모델이 더 이상 향상되지 않고 과적합되기 시작한 시점을 감지한 것이다.

최종 검증 정확도는 **0.8210**으로 훈련 데이터와는 다소 차이가 있지만, 상당히 좋은 성능을 나타냄을 의미한다.

결과적으로, **SimpleRNN** 모델은 **IMDB** 데이터에 약 **80**퍼 이상의 정확도를 보여주고 싶다.

LSTM

Epoch	손실값(Training)	정확도(Training)	손실값(Validation)	정확도(Validation)
1	0.5229	0.7314	0.3452	0.8594
2	0.3319	0.8661	0.4959	0.7704
3	0.2798	0.8932	0.3910	0.8350
4	0.2382	0.9097	0.4140	0.8310
5	0.2228	0.9173	0.3092	0.8666
6	0.1926	0.9299	0.3392	0.8620
7	0.1677	0.9394	0.3751	0.8540
8	0.1520	0.9464	0.3514	0.8740
9	0.1296	0.9561	0.3503	0.8784

LSTM 학습 결과 분석

훈련 정확도와 검증 정확도를 살펴보면 훈련 정확도가 점진적으로 상승하고 있다. 이는 모델이 훈련 데이터를 잘 학습하고 있다는 것을 의미한다.

검증 정확도도 **Epoch 1**에서부터 상승하고 있지만, **Epoch 3** 이후에는 큰 변화가 없고 일정하게 유지되고 있다.

Early Stopping이 **Epoch 9**에서 작동하여 훈련이 조기 종료되었다.

최종 검증 정확도는 **0.8784**로 나타났다. 이는 훈련 데이터와는 다소 차이가 있지만, 상당히 좋은 성능을 나타냄을 의미한다.

결과적으로, **LSTM** 모델은 **IMDB** 데이터에 대해 **RNN**보다 좋은 성능을 보여주고 있습니다.

Self Attention

Epoch	손실값(Training)	정확도(Training)	손실값(Validation)	정확도(Validation)
1	0.3881	0.8129	0.2703	0.8894
2	0.2133	0.9199	0.2850	0.8807
3	0.1622	0.9415	0.3077	0.8791
4	0.1262	0.9566	0.3514	0.8597
5	0.1007	0.9662	0.4111	0.8642

Self-Attention 학습 결과 분석

Self-Attention 모델은 **5**번의 에포크 후 조기 종료되었다.

훈련 정확도는 **0.8129**에서 시작하여 점진적으로 상승하였고, 검증 정확도는 **0.8894**로 상당히 높은 값을 나타냈습니다.

하지만 검증 손실이 **0.2703**에서 **0.4111**까지 상승한 것으로 보아, 모델이 훈련 데이터에 과적합되었을 가능성이 있다 이는 훈련 데이터와는 다소 차이가 있지만, 상당히 좋은 성능을 나타냄을 의미한다

결과적으로, **Self-Attention** 모델은 **IMDB** 데이터에 대해 **RNN**보다 훨씬 더 좋은 성능을 보여주고 **LSTM** 모델보다도 학습데이터에 있어서는 더 작은 **epoch**로 좋은 성능을 보여주고 있다.

	Epoch	손실값(Training)	정확도(Training)	손실값(Validation)	정확도(Validation)
RNN	6	0.3470	0.8557	0.4513	0.8210
LSTM	9	0.1296	0.9561	0.3503	0.8784
Self-Attention	5	0.1007	0.9662	0.4111	0.8642

LSTM과 **Self-Attention** 모델이 **RNN**보다 정확도가 높은걸 확인할 수 있었고 **LSTM**과 **Self-Attention** 모델은 비슷하지만 **Attention**모델이 더 빠르게 학습한걸 확인할 수 있다.

모델 평가 및 예측

모델 평가

RNN

- 테스트 정확도 : **0.8407**

LSTM

- 테스트 정확도 : **0.8764**

Self Attention

- 테스트 정확도 : **0.8642**

모델 예측

긍정적인 리뷰

```
# 긍정적인 리뷰
positive_review = "I watched this film last night and it was an amazing experience. \
The performances by the actors were top notch and the storyline was gripping. \
The cinematography was beautiful, it really transported me into the movie's world. \
I highly recommend this film to anyone who enjoys quality cinema."
```

```
어젯밤에 이 영화를 봤는데 정말 놀라운 경험이었어요.
배우들의 연기는 최고 수준이었으며 스토리 라인은 흥미진진했습니다.
영화 촬영이 아름다웠고 영화 속 세계로 나를 데려다 주었어요.
수준 높은 영화를 즐기는 모든 분들께 이 영화를 강력히 추천합니다.
```

부정적인 리뷰

```
# 부정적인 리뷰
negative_review = "I really wanted to like this movie but it just didn't hit the mark for me. \
The plot was full of holes and the character development was almost non-existent. \
Despite having a strong cast, the performances were lackluster due to the weak script. \
I wouldn't recommend wasting your time on this one."
```

```
이 영화를 정말 좋아하고 싶었지만 제게는 맞지 않았습니다.
줄거리에 구멍이 많았고 캐릭터 전개가 거의 존재하지 않았다.
탄탄한 출연진에도 불구하고 각본이 약해서 연기가 형편없었다.
시간을 낭비하지 않는 것이 좋습니다.
```

RNN

- 긍정적인 리뷰

`96.70%` 확률로 긍정 리뷰입니다.

- 부정적인 리뷰

`92.79%` 확률로 부정 리뷰입니다.

LSTM

- 긍정적인 리뷰

`98.44%` 확률로 긍정 리뷰입니다.

- 부정적인 리뷰

`97.79%` 확률로 부정 리뷰입니다.

Self Attention

- 긍정적인 리뷰

`99.60%` 확률로 긍정 리뷰입니다.

- 부정적인 리뷰

`99.56%` 확률로 부정 리뷰입니다.

```
========================= 긍정적인 리뷰 ==============================
============================RNN model 예측============================
1/1 [==============================] - 0s 56ms/step
96.70% 확률로 긍정 리뷰입니다.
============================LSTM model 예측===========================
1/1 [==============================] - 0s 28ms/step
98.44% 확률로 긍정 리뷰입니다.
============================트랜스포머 model 예측====================
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
99.60% 확률로 긍정 리뷰입니다.
0.40% 확률로 부정 리뷰입니다.
========================= 부정적인 리뷰 ==============================
============================RNN model 예측============================
1/1 [==============================] - 0s 67ms/step
92.79% 확률로 부정 리뷰입니다.
============================LSTM model 예측===========================
1/1 [==============================] - 0s 28ms/step
97.79% 확률로 부정 리뷰입니다.
============================트랜스포머 model 예측====================
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 22ms/step
0.44% 확률로 긍정 리뷰입니다.
99.56% 확률로 부정 리뷰입니다.
```

모델 별 예측결과

모델 별 긍정적인 리뷰 예측 결과

모델	예측결과	확률(%)
RNN	긍정 리뷰	**96.70**
LSTM	긍정 리뷰	**98.44**
Self-Attention	긍정 리뷰	**99.60**

모델 별 부정적인 리뷰 예측 결과

모델	예측결과	확률(%)
RNN	부정 리뷰	**92.79**
LSTM	부정 리뷰	**97.79**
Self-Attention	부정 리뷰	**99.56**

RNN

시퀀셜 데이터에서 순차적인 정보를 고려하는 장점이 있지만, 장기 의존성 문제로 인해 긴 문장에서 다른 모델에 비해 성능이 떨어지는걸 확인할 수 있다.

LSTM

RNN의 장기 의존성 문제를 개선하여 긴 문장에서도 **RNN**보다 더 좋은 성능을 보여주는걸 확인할 수 있다.

Self-Attention

Self-Attention 모델은 **LSTM**과 달리 순환 구조를 사용하지 않고, 어텐션 메커니즘을 활용하여 문장의 모든 단어를 동시에 처리하며 학습과 추론 속도를 향상시켜 더 빠르게 학습하고 예측할 수 있으며 가장 높은 정확도가 나왔다.

결론

예측 결과를 통해 **LSTM**과 **Self-Attention**모델이 **RNN** 모델보다 더 높은 신뢰도로 리뷰를 분류하는 것으로 나타났다. 특히, **Self-Attention** 모델의 신뢰도가 가장 높은 것으로 보인다. 이는 트랜스포머 모델이 순차적인 정보 처리를 넘어선 문맥 파악과 어텐션 메커니즘을 활용하여 더 효과적인 텍스트 분류를 가능케 한다는 것을 보여준다.