14주차 RNN LSTM - 카테고리 분류 및 키워드

순환 신경망 RNN : recurrent neural network

지속되는 생각을 하지 못한다 따라 기억된 데이터

순환 신경망 - **여러개의 데이터 순서**대로 입력 → 앞에서 입력 데이터를 기억함

기억된 데이터가 얼마나 중요한지 판단하고 → 별도의 가중치를 주고 다음 데이터로 넘김

순서대로 실행 → 깥은 층을 맴도는 것처럼 보임 해당 hidden layer를 점검

순환한다 → 어떤 공식으로 되어있는 지 확인할 것

오늘 + 주가 → 오늘이라고 오늘의 주가라고 기억함

→ 잠시 앞에서 입력받은 데이터를 잠시 기억

  전체를 다 주면 기울기 소실 (vanishing gredient) 문제가 생김 → 계속

특정만 짤라서 얘만 중요해 → 가중치 ** 영향은 가장 마지막 가중치가 영향을 가짐

→ 개념적, 수학적 요소가 어려운 것

LSTM - Long Short Term Memory

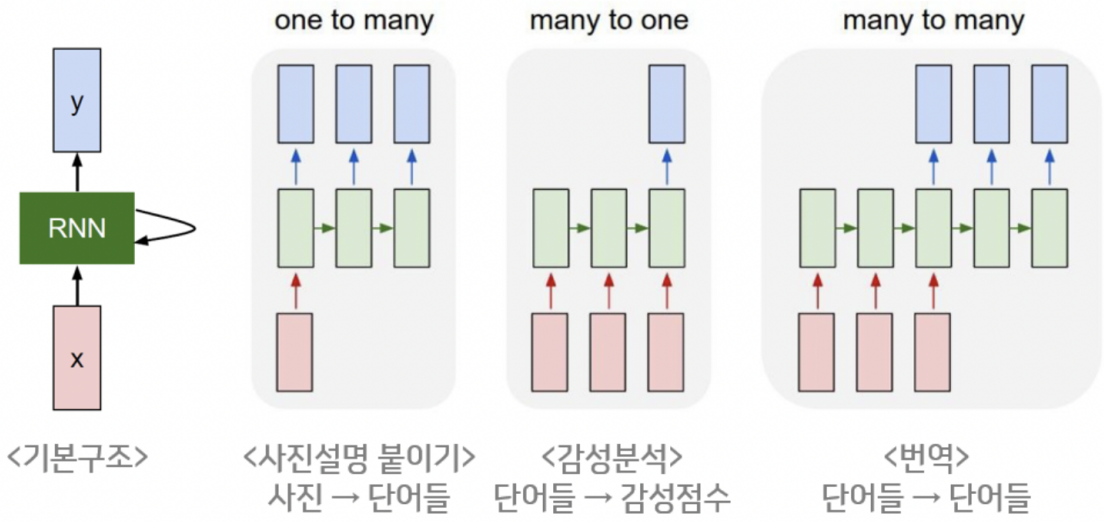
: 반복을 많이 수행하는 RNN → 기울기 소실 문제가 많이 발생함

→ **해결하기 어렵다는 단점을 보완함 → 반복직전, 기억된 값 넘길까?말까~ 추가함**

RNN 적용 방식

RNN 방식의 장점은 입력 값과 출력 값을 어떻게 설정하느냐에 따라 여러 가지 상황에서 이를 적용할 수 있음

- 단일 입력 다수 출력(one-to-many): 사진의 캡션을 만들 때 활용
- 다수 입력 단일 출력(many-to-one): 문장을 읽고 뜻을 파악할 때 활용
- 다수 입력 다수 출력(many-to-many): 문장을 번역할 때 활용



→ DGAN combined with LSTM ?

```
(X_train,y_train),(X_test,y_test)=reuters.load_data(num_words=1000,test_split=0.2)
#num_words=1000 = input data 갯수????
num_words는 이 데이터에서 등장 빈도 순위로 몇 번째에 해당하는 단어까지만
사용할 것인지 조절합니다. 예를 들어서 100이란 값을 넣으면,
등장 빈도 순위로 상위 1~100 등에 해당하는 단어만 사용
category =np.max(y_train)+1
# y_train 분류 - text의 카테고리를 나누기 위함 0부터 레이블링
```

```
category =np.max(y_train)+1

print(np.max(X_train[0]) - 의미 : 384 의 의미는 0기사에서 가장 마지막에 있는 카테고리 번호
print(len(X_train[1000]
```

X_train[0] - 1000개 이상의 빈도를 가진 애들이 나옴

단어의 빈도가 하나의 열이라고 생각

model.add(Embedding(input_dim=1000,output_dim=32,input_length=100))

- input_dim : 1~1000 랭킹 1000개
- **output_dim**: 정수 >= 0. 밀집 임베딩의 차원.

▼ LSTM을 이용한 로이터 뉴스 카테고리 분류하기

- 케라스에서 제공하는 로이터 뉴스 데이터를 LSTM을 이용하여 텍스트 분류.
- 로이터 뉴스 기사 데이터는 총 11,228개의 뉴스 기사가 46개의 뉴스 카테고리로 분류되는 뉴스 기사 데이터
- 텍스트를 토큰(Tokenization)의 단위로 분할하는 작업

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Embedding
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.datasets import reuters
from tensorflow.keras.callbacks import EarlyStopping

import numpy as np
import matplotlib.pyplot as plt

(X_train, y_train), (X_test, y_test) = reuters.load_data(num_words=1000, test_spli
t=0.2)
#(num_words 1000개 상위 1000개 추출)

category=np.max(y_train) +1 #==np.max(y_test)+1

print(category,'카테고리')
print(len(X_train),학습용 뉴스 총 갯수)
print(len(X_test),학습용 뉴스 총 갯수)



output :
46  카테고리
8982 학습용 뉴스 기사
2246 테스트용 뉴스 기사
```

```
[4]  print(X_train[0]) #1000개의 숫자의 ranking : 전처리부터 해보면 이해할 수 있음~~~!!

     [1, 2, 2, 8, 43, 10, 447, 5, 25, 207, 270, 5, 2, 111, 16, 369, 186, 90, 67

[5]  np.max(X_train[0])

     864
```

```
data = reuters.get_word_index()
len(data)

30979
```

→ word index 총 갯수 30979개 : 각 단어와 그 단어에 부여된 인덱스를 리턴

- 데이터 안에서 해당 단어가 몇 번이나 나타나는지 세어 빈도에 따라 번호를 붙였음
- 예를 들어, 2이라고 하면 두 번째로 빈도가 높은 단어라는 의미함
- 이 작업을 위해 tokenizer() 같은 함수를 사용하는데, 케라스에서 제공하는 데이터는 이 작업을 이미 마친 상태임

return sequence

```
print('첫번째 훈련용 뉴스 기사 :', X_train[0]) #토큰화 -> data 단위 : 형태소
print('첫번째 훈련용 뉴스 기사의 레이블 :',y_train[0])
```

데이터전처리

- 주의해야 할 점은 각 기사의 단어 수가 제각각 다르므로 이를 동일하게 맞추어야 함

=> X_train = sequence.pad_sequences(X_train, maxlen=100) =>> input size를 동일하게 넣어주기 위한 전처리 #딥러닝 모델에 입력하려면 학습 데이터의 길이가 동일해야하므로

- 이때는 다음과 같이 ***데이터 전처리 함수 pad_sequences()****를 이용함
- ***maxlen=100: 단어 수를 100개로 맞추라는 의미****
- 입력된 기사의 단어 수가 100보다 크면 100개째 단어만 선택하고 나머지는 버리고, 100보다 작을 경우 모두 0으로 채움

```
#기사의 단어 총 갯수 100으로 고정
X_train=sequences.pad_sequences(X_train,maxlen=100)
X_test = sequence.pad_sequences(X_test, maxlen=100)

#원핫인코딩 처리
```

```
y_train=to_categorical(y_train)
y_test=to_categorical(y_test)
```

모델 구조 설정

- Embedding 층과 LSTM 층을 새로 추가
- Embedding 층은 데이터 전처리 과정을 통해 ***입력된 값을 받아 다음 층이 알 수 있는 형태로 변환하는 역할****을 수행
- ***Embedding(input_dim=10000,output_dim=32,input_length=100)**** 형식으로 사용하며, 모델 설정 부분의 맨 처음에 있어야 함 **‘입력 (상위 1000개 추출) , 출력, 단어 수(기사 속 단어 갯수)’ index 0부터**

=> 해당 내용 숙지

- LSTM은 앞서 설명했듯이 RNN에서 기억 값에 대한 가중치를 제어
- **LSTM(기사당 단어 수, 기타 옵션)** 형식으로 적용
- **LSTM의 활성화 함수로는 tanh을 사용**

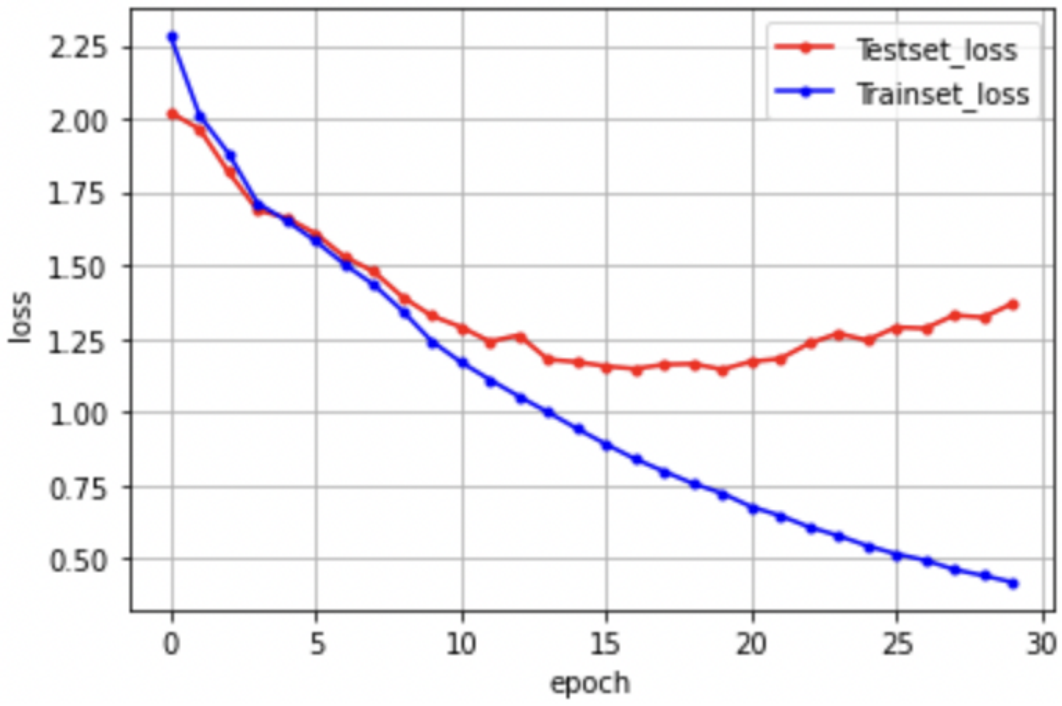
```
model = Sequential() #Embedding - 자연어처리 -> 모델구조 입력층을 받아서 다음으로 넘겨줌 -> on
e hot encoding과 연관
model.add(Embedding(input_dim=1000,output_dim=32,input_length=100)) #num_words=100
0 최대 차원을 넘겨줘야함 하나의 열에 들어갈 차원을 의미함
#input_dim=1000 상위 몇개로 나눴는지
#input_length=100 1개의 데이터당 몇개의 단어가 있는지  매번 입력될 ‘단어 수’ sequence 수대로 적
용

단어 사전 내에 있는 불러올 단어 총 수 1000 -> 하나의 데이터 단어 수 : 기사당 단어 수 100


model.add(LSTM(units=100, activation='tanh')) #sigmoid과의 차이점 숙지 -> -1~1 => 성능
이 우수해짐 => 표준화에 가깝기때문에 기울기 소실 문제는 계속 발생함 -> 이를 보완한 모델 LSTM
model.add(Dense(units=46, activation='softmax')) #다중 분류 len(y_train)
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accurac
y'])
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)

history = model.fit(X_train, y_train, batch_size=20, epochs=200, validation_data=
(X_test, y_test), callbacks=[early_stopping_callback])
```

```
model.evaluate(X_test, y_test)[0]
#[0] loss : [1] acc 아무것도 안쓰면 2개 다 나왕~~
```

```
71/71 [==============================] - 0s 5ms/step - loss: 1.3686 - accuracy: 0.711
1.368633508682251
```

```
# 학습셋과 테스트셋의 오차 저장
y_vloss = history.history['val_loss']
y_loss = history.history['loss']

# 그래프 표현
x_len = np.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')
#과적합이 발생하는 위치 (earlystopping필요) -> patience(과적합이 많이 발생하는 이유)

#할때마다 각각 모델을을 저장해야함 -> 기존 CNN 함수 참고 -> 그래프보고 최적의 모델을 사용하면 됨

plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```



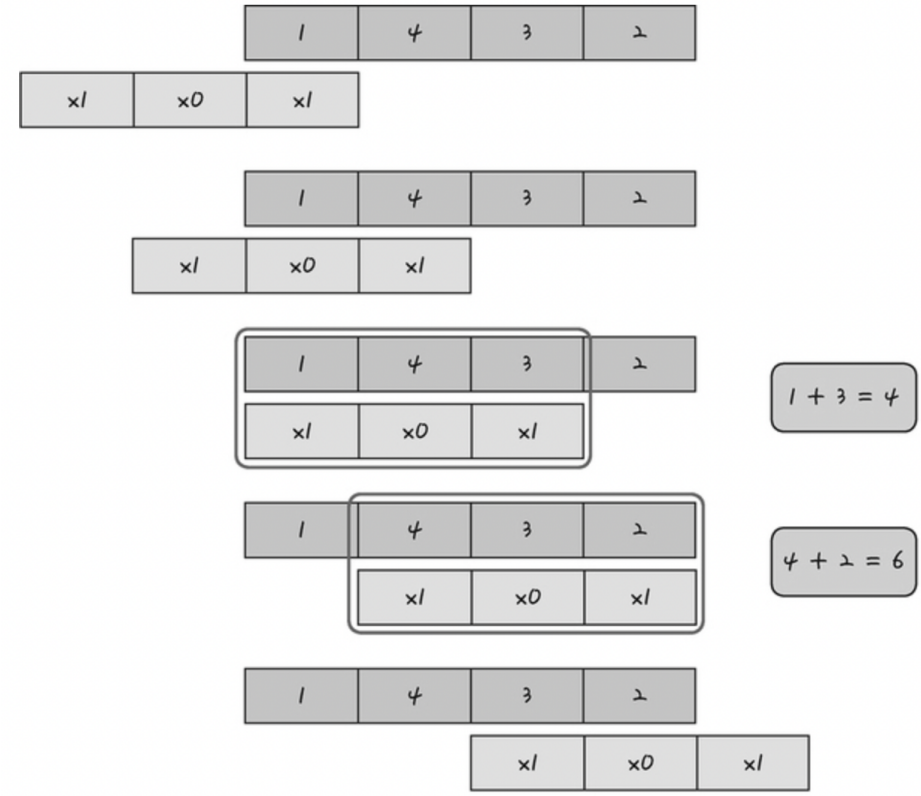
▼ LSTM 과 CNN 조합을 이용한 영화 리뷰 분류하기

- **인터넷 영화 데이터베이스(Internet Movie DataBase, IMDB)**는 영화와 관련된 정보와 출연진 정보, 개봉 정보, 영화 후기, 평점까지 매우 폭넓은 데이터가 저장된 자료임
- 영화에 관해 남긴 2만 5,000여 개의 영화 리뷰가 담겨 있으며, 해당 영화를 긍정적으로 평가했는지 혹은 부정적으로 평가했는지도 담겨 있음
- 각 단어에 대한 전처리를 마친 상태

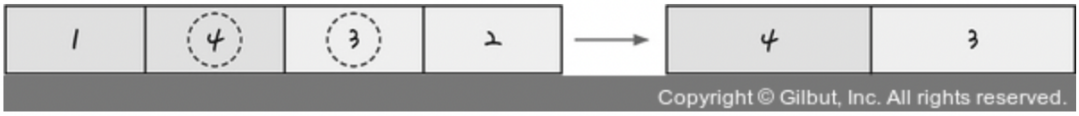
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Embedding, LSTM, C
onv1D, MaxPooling1D
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.callbacks import EarlyStopping

import numpy as np
import matplotlib.pyplot as plt

#CNN 이미지만 쓰는 것이 아니고 -> 특징점을 추출할 수 있음 = text적용 Conv1D MaxPooling1D -> T
EXT에 쓰임 Conv2D= image에 적용

# 데이터를 불러와 학습셋, 테스트셋으로 나눔
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=5000)

# 데이터 전처리: 단어의 수를 맞춤
X_train = sequence.pad_sequences(X_train, maxlen=500)
X_test = sequence.pad_sequences(X_test, maxlen=500)
```

```
#모델의 구조 설정
CNN_LSTM_model=Sequential()
CNN_LSTM_model.add(Embeddig(5000,500) #500차원 벡터 -> 5000 단어에만 관심있음
CNN_LSTM_model.add(Dropout(0.5))
CNN_LSTM_model.add(Conv1D(64,3,padding='valid',activation='relu',strides=1)) #64번
연산 -> Input data : 500
CNN_LSTM_model.add(MaxPooling1D(pool_size=4))
CNN_LSTM_model.add(LSTM(50,activation='tanh'))
CNN_LSTM_model.add(Dense(1))
CNN_LSTM_model.add(Activation('sigmoid'))
CNN_LSTM_model.summary()
```

CNN_LSTM_model.add(Conv1D(64, 3, padding='valid', activation='relu',strides=1)) 해당 코드 내용 시험 출제



Maxpooling 4,3




```
CNN_LSTM_model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accur
acy'])
early_stopping_callback=EarlyStopping(monitor='val_loss',patience=5)
history=CNN_LSTM_model.fit(X_train,y_train,batch_size=40,epochs=100,validation_spl
it=0.25,callbacks=[early_stopping_callback])


CNN_LSTM_model.evaluate(X_test,y_test)
```

```
y_vloss=history.history['val_loss']
y_loss=history.history['loss']
```

```
x_len=np.arange(len(y_loss))

plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')

plt.legend(loc="uppder right")
plt.grid()
plt.xlabel("epoch")
plt.ylabel("loss")
plt.show()
```