**GRU 활용 및 RNN활용 언어 모형 생성**

**1. GRU 활용 분류**

- 데이터 가져 오기

영화 사이트 IMDB의 리뷰 데이터입니다.

이 데이터는 리뷰에 대한 텍스트와 해당 리뷰가 긍정인 경우 1을 부정인 경우 0으로 표시한 레이블로 구성된 데이터입니다.

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**from** tensorflow.keras.datasets **import** imdb

**영화 리뷰 데이터를 갖고오겠습니다.**

(X\_train, y\_train), (X\_test, y\_test) = imdb.load\_data()

imdb.data\_load()의 파라미터로 num\_words를 사용하면 이 데이터에서 등장 빈도 순위로 몇 번째에 해당하는 단어까지를 사용할 것인지를 의미합니다.

예를 들어서 10,000을 넣으면, 등장 빈도 순위가 1~10,000에 해당하는 단어만 사용하게 됩니다. 즉, 단어 집합의 크기는 10,000이 됩니다. 지금은 별도로 제한하지 않겠습니다.

**데이터 셋 길이 및 카테고리 확인**

**print**('훈련용 리뷰 개수 : {}'.**format**(len(X\_train)))

**print**('테스트용 리뷰 개수 : {}'.**format**(len(X\_test)))

num\_classes = len(set(y\_train))

**print**('카테고리 : {}'.**format**(num\_classes))

훈련용 리뷰는 25,000개, 테스트용 리뷰는 25,000개, 카테고리는 2개입니다.

**첫번째 배열 프린트**

**print**(**X\_train**[0])

**print**(**y\_train**[0])

**학습 데이터 전체 길이 리스트 생성 및 분포 확인**

25,000개의 훈련용 리뷰의 각 길이는 전부 다른데, 리뷰의 길이 분포를 그래프로 시각화해보겠습니다.

len\_result = [len(**s**) **for** **s** in X\_train]

**print**('리뷰의 최대 길이 : {}'.**format**(np.max(len\_result)))

**print**('리뷰의 평균 길이 : {}'.**format**(np.mean(len\_result)))

plt.subplot(1,2,2)

plt.hist(len\_result, bins=50)

plt.show()

대체적으로 1,000이하의 길이를 가지며, 특히 100~500길이를 가진 데이터가 많은 것을 확인할 수 있습니다. 반면, 가장 긴 길이를 가진 데이터는 길이가 2,000이 넘는 것도 확인할 수 있습니다. 레이블의 분포를 확인해보겠습니다.

**긍정/부정 분포 즉 0과 1의 분포 확인**

unique\_elements, counts\_elements = np.unique(y\_train, return\_counts=**True**)

**print**("각 레이블에 대한 빈도수:")

**print**(np.asarray((unique\_elements, counts\_elements)))

동일하게 12500개씩 존재 함을 확인

**숫자와 단어 매핑**

imdb.get\_word\_index()에 각 단어와 맵핑되는 정수가 저장되어져 있습니다. 주의할 점은 imdb.get\_word\_index()에 저장된 값에 +3을 해야 실제 맵핑되는 정수입니다. 이것은 IMDB 리뷰 데이터셋에서 정한 규칙입니다.

word\_to\_index = imdb.get\_word\_index()

index\_to\_word = {}

**for** key, **value** **in** word\_to\_index.items():

index\_to\_word[**value**+3] = key

IMDB 리뷰 데이터셋에서는 0, 1, 2, 3은 특별 토큰으로 취급하고 있습니다. 그래서 정수 4부터가 실제 IMDB 리뷰 데이터셋에서 빈도수가 가장 높은 실제 영단어입니다.

**첫번째 리뷰 단어 매핑**

**for** **index**, token in enumerate(("<pad>", "<sos>", "<unk>")):

index\_to\_word[**index**] = token

# <pad>: padding, 길이를 맞출때 사용하는 비어있는(사용x) 토큰

# <sos>: start of sentence, 문장의 시작을 알리는 토큰

# <unk>: unknown, 모델이 인식할 수 없는 토큰

**print**(' '.**join**([index\_to\_word[**index**] **for** **index** in X\_train[0]]))

첫번째 훈련용 리뷰의 X\_train[0]이 인덱스로 바뀌기 전에 어떤 단어들이었는지 확인해보겠습니다.

**모델 만들기**

우선 필요한 패키지를 가져옵니다.

**import** re

**from** tensorflow.keras.datasets **import** imdb

**from** tensorflow.keras.preprocessing.sequence **import** pad\_sequences

**from** tensorflow.keras.models **import** Sequential

**from** tensorflow.keras.layers **import** Dense, GRU, Embedding

**from** tensorflow.keras.callbacks **import** EarlyStopping, ModelCheckpoint

**from** tensorflow.keras.models **import** load\_model

**리뷰는 1,000으로 제한합니다**. 시간이 너무 오래 걸림 1000개만

vocab\_size = 1000

(X\_train, y\_train), (X\_test, y\_test) = imdb.load\_data(num\_words = vocab\_size)

리뷰 최대 길이는 500으로 제한합니다.

**패딩하기**

pad\_sequences() 활용. 길이는 max\_len에 넣는 값으로 정해집니다. 훈련 데이터가 정한 길이를 초과하면 초과분을 삭제하고, 부족하면 0으로 채웁니다.

max\_len = 500

X\_train = pad\_sequences(X\_train, maxlen=max\_len)

X\_test = pad\_sequences(X\_test, maxlen=max\_len)

**단어 임베딩 설정 및 모델 구성**

Embedding()은 두 개의 인자를 받는데, 첫번째 인자는 단어 집합의 크기이며 두번째 인자는 임베딩 후의 벡터 크기입니다. 여기서는 100을 선택했습니다. 즉, 입력 데이터에서 모든 단어는 100차원의 임베딩 벡터로 표현됩니다.

embedding\_dim = 100

hidden\_units = 128

model = Sequential()

model.add(Embedding(vocab\_size, embedding\_dim))

model.add(GRU(hidden\_units))

model.add(Dense(1, activation='sigmoid'))

**얼리스톱과 체크 포인터 설정**

검증 데이터의 손실(loss)이 증가하면, 과적합 징후이므로 검증 데이터 손실이 4회 증가하면 학습을 중단하는 조기 종료(EarlyStopping)를 사용합니다. 또한, ModelCheckpoint를 사용하여 검증 데이터의 정확도가 이전보다 좋아질 경우에만 모델을 저장하도록 합니다.

es = EarlyStopping(monitor='val\_loss', mode='min', verbose=1, patience=4)

mc = ModelCheckpoint('GRU\_model.h5', monitor='val\_acc', mode='max', verbose=1, save\_best\_only=**True**)

**옵티마이저 및 로스펑션 설정**

model.compile(optimizer='rmsprop', loss='binary\_crossentropy', metrics=['acc'])

history = model.fit(X\_train, y\_train, epochs=15, callbacks=[es, mc], batch\_size=60, validation\_split=0.2)

긍정인지 부정인지에 대한 이진 판별값이 출력이 되기 때문에, 손실 함수는 binary\_crossentropy를 사용합니다. 최적화 함수는 rmsprop를 사용하였습니다. 또한, 에포크마다 정확도를 구하기위해 accuracy를 추가해줍니다. 에포크는 총 10회를 수행하겠습니다.

**조기종료 모델 불러 오기**

. 훈련이 다 되었다면 이제 테스트 데이터에 대해서 정확도를 평가할 차례입니다. 훈련 과정에서 검증 데이터의 정확도가 가장 높았을 때 저장된 모델인 'GRU\_model.h5'를 로드합니다.

loaded\_model = load\_model('GRU\_model.h5')

**print**("\n 테스트 정확도: %.4f" % (loaded\_model.evaluate(X\_test, y\_test)[1]))

25000/25000 [==============================] - 61s 2ms/sample - loss: 0.3440 - acc: 0.8893

테스트 정확도: 0.8893

테스트 데이터에 대해서 정확도 88.93%를 얻습니다.

임의의 문장에 대해서 리뷰의 긍, 부정을 예측하고자 합니다. 이를 위해서는 모델에 넣기 전에 임의의 문장에 대해서 전처리를 해주어야 합니다. sentiment\_predict은 입력된 문장에 대해서 기본적인 전처리와 정수 인코딩, 패딩을 한 후에 모델의 입력으로 사용하여 예측값을 리턴하는 함수입니다.

**2. 언어 모형 만들기**

**- 데이터 가져 오기**

**import** **keras**

**import** **pandas** **as** **pd**

df = pd.read\_csv('amazon\_cells\_labelled.txt', sep='**\t**', header=None)

df.head()

**일단은 단어 단위로 토큰화를 해준다**.

tok = keras.preprocessing.text.Tokenizer()

**이제 df[0] 즉 0번열 실제로 토큰화 및 텍스트를 단어 번호의 리스트로 변환**

tok.fit\_on\_texts(df[0])

seq = tok.texts\_to\_sequences(df[0])

**아마존 리뷰 데이터 활용해서 문장 생성 모델 실습**

아마존 리뷰 데이터에서 첫번째 글을 보면 문장의 시작에는 'So'가 나오고, 'So' 다음에는 'there'이 나오며, 'So there' 다음에는 'is'가 나온다. 이러한 관계를 순환신경망에 학습시키면 된다.

df.iloc[0, 0]

'So there is no way for me to plug it in here in the US unless I go by a converter.'

**텍스트의 시작과 끝을 나타내는 단어를 사전에 추가**

tok.word\_index['<START>'] = start = len(tok.word\_index) + 1

tok.index\_word[start] = '<START>'

tok.word\_index['<END>'] = end = len(tok.word\_index) + 1

tok.index\_word[end] = '<END>'

모든 텍스트의 앞과 뒤에 시작과 끝 표시를 붙여 prev\_seq를 만들고, 끝 표시만 붙은 next\_seq를 만든다.

이렇게 하면 prev\_seq와 next\_seq는 한 단어씩 어긋나게 된다. 순환신경망에 prev\_seq를 입력으로, next\_seq를 출력으로 넣어줄 것이다.

prev\_seq = []

next\_seq = []

**for** s **in** seq:

prev\_seq.append([start] + s + [end])

next\_seq.append(s + [end])

**패딩을 위해 텍스트의 최대 길이를 구한다.**

MAXLEN = max(len(s) **for** s **in** prev\_seq)

**패딩을 하는데 이전과 달리 뒤에 0을 넣어 채워준다.**

**from** **keras.preprocessing.sequence** **import** pad\_sequences

prev\_pad = pad\_sequences(prev\_seq, MAXLEN, padding='post')

next\_pad = pad\_sequences(next\_seq, MAXLEN, padding='post')

prev\_pad[0]

next\_pad[0]

* **padding**: 문자열, 'pre' 혹은 'post': 각 시퀀스의 처음 혹은 끝을 패딩합니다.

**데이터 분할**

**from** **sklearn.model\_selection** **import** train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(prev\_pad, next\_pad, test\_size=.2, random\_state=1234)

**모형 만들기**

NUM\_WORDS = len(tok.index\_word) + 1

**from** **keras.models** **import** Sequential

**from** **keras.layers** **import** Dense, Embedding, LSTM, TimeDistributed

rnn = Sequential()

rnn.add(Embedding(input\_dim=NUM\_WORDS, output\_dim=8, input\_length=MAXLEN, mask\_zero=True))

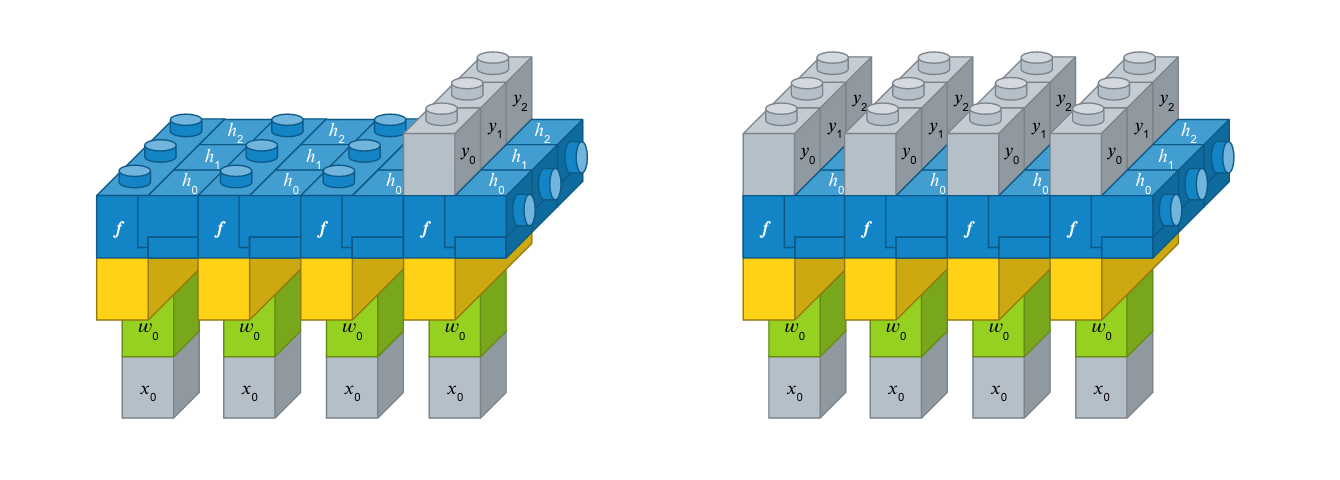
return\_sequences=True로 **모든 입력에 대해 출력을 내놓게 한다**.

rnn.add(LSTM(16, return\_sequences=True))

#### 출력 형태

* return\_sequences : 시퀀스 출력 여부

LSTM 레이어는 return\_sequences 인자에 따라 마지막 시퀀스에서 한 번만 출력할 수 있고 각 시퀀스에서 출력을 할 수 있습니다. many to many 문제를 풀거나 LSTM 레이어를 여러개로 쌓아올릴 때는 return\_sequence=True 옵션을 사용합니다. 자세한 것은 뒤에서 살펴보겠습니다. 아래 그림에서 왼쪽은 return\_sequences=False일 때, 오른쪽은 return\_sequence=True일 때의 형상입니다.



이제 LSTM은 입력된 단어 수만큼 출력을 반복해서 내놓게 된다. 각각의 출력에서 Dense레이어는 다음에 나올 단어를 예측한다. 출력마다 Dense를 반복해서 덧붙이고자 하기 때문에 TimeDistributed를 사용한다.

rnn.add(TimeDistributed(Dense(NUM\_WORDS, activation='softmax')))

엑티베이션 함수도 시그모이드가 아닌 여러 개 확률 출력 softmax사용

**모델 구성 확인**

rnn.summary()

모형의 요약을 보면 마지막 레이어의 형태가 (None, 1879)가 아닌 (None, 32, 1879)이다. 입력이 최대 32개이므로 출력도 최대 32개인데 Dense가 반복되기 때문에 형태가 달라진 것이다.

**출력의 형태 맞추기**

y\_train.shape

(800, 32)

출력의 형태가 (None, 32, 1881)인데 데이터는 (800, 32)의 형태이므로 차원이 맞지 않는다. 끝에 1차원을 덧붙여서 형태를 맞춰준다.

y\_train\_dims = numpy.expand\_dims(y\_train, 2)

y\_train\_dims.shape

(800, 32, 1)

**학습 하기**

**from** **keras.optimizers** **import** Adam

rnn.compile(optimizer=Adam(lr=.1), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'], sample\_weight\_mode='temporal')

rnn.fit(x\_train, y\_train\_dims, epochs=10)

<keras.callbacks.History at 0x14912613dd8>

y\_train.shape

(800, 32)

**sample\_weight\_mode**: 시간 단계별로 샘플 가중치를 주어야 하는 경우 (2D 가중치), 이 인수를 "temporal"로 설정해야 합니다. 디폴트 값은 None으로 (1D) 샘플별 가중치를 적용합니다. 모델이 다중 아웃풋을 갖는 경우, 모드의 딕셔너리 혹은 모드의 리스트를 전달하여 각 아웃풋마다 별도의 sample\_weight\_mode를 사용할 수 있습니다

언어 모형을 이용해 문장의 다음 단어를 예측해보자. 예를 들어 첫번째 리뷰의 앞 10단어는 다음과 같다.

[tok.index\_word[i] **for** i **in** prev\_seq[0][:10]]

['<START>', 'so', 'there', 'is', 'no', 'way', 'for', 'me', 'to', 'plug']

이어서 나올 단어는 'it'이다.

i = prev\_seq[0][10]

tok.index\_word[i]

'it'

이제 RNN으로 예측을 해보자.

new\_sentence = [prev\_seq[0][:10]]

패딩을 하고

new\_pad = pad\_sequences(new\_sentence, MAXLEN, padding='post')

예측을 한다.

next\_words = rnn.predict(new\_pad)

1개의 텍스트에 대해 32단어 길이로 1881종의 단어에 대한 예측이 나왔다.

next\_words.shape

(1, 32, 1881)

가장 확률이 높은 단어는 4번이다.

next\_words[0, 10].argmax()

tok.index\_word[숫자] 로 단어 확인

이를 처음부터 끝까지 반복하게 해서 완전 문장 생성.

new\_sentence = [[start]]

new\_pad = pad\_sequences(new\_sentence, MAXLEN, padding='post')

**for** i **in** range(MAXLEN - 1):

next\_words = rnn.predict(new\_pad) *# 예측*

word = next\_words[0, i].argmax() *# 가장 확률이 높은 단어 선정*

**print**(tok.index\_word[word]) *# 단어 출력*

new\_pad[0, i + 1] = word *# 선정 단어를 추가*

**if** word == end: *# 문장이 끝나면 중단*

**break**

위의 방식으로 하면 매번 같은 문장이 만들어지므로 다양성이 부족하다. 확률이 가장 높은 단어를 선택하는 대신, 단어를 확률에 따라 무작위로 추출하게 하자.

**import** **numpy.random**

new\_sentence = [[start]]

new\_pad = pad\_sequences(new\_sentence, MAXLEN, padding='post')

**for** i **in** range(MAXLEN - 1):

next\_words = rnn.predict(new\_pad)

*# 확률에 따라 단어를 무작위로 추출*

word = numpy.random.choice(NUM\_WORDS, p=next\_words[0, i])

**print**(tok.index\_word[word])

new\_pad[0, i + 1] = word

**if** word == end:

**break**