

AI Programming

Lab 05: LAB_05_GRU_dist

소속: 컴퓨터정보공학부

학번: 2019202103

이름: 이은비

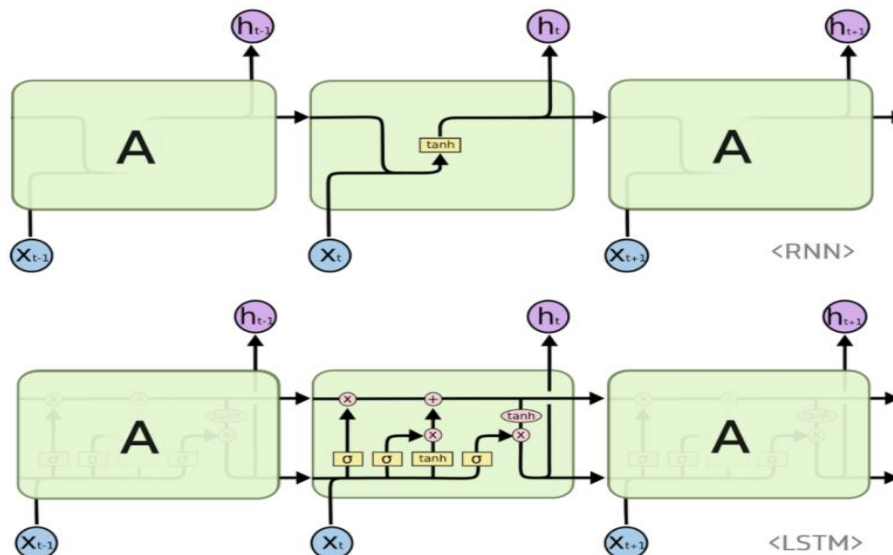
<Lab Objective and Whole simulation program flow>

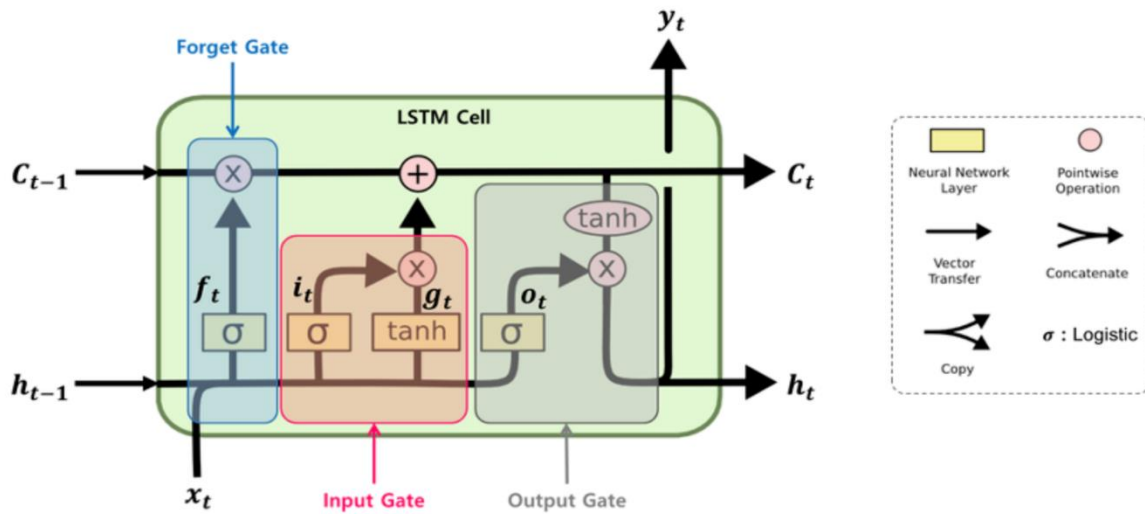
RNN 과 비교해서 다른 모델인 LSTM 과 GRU 에 대한 개념을 이해하고 그에 대한 코드내용을 실행해 봄으로써 그 결과를 확인하는 작업을 진행합니다. 기본적인 LSTM 과 GRU 의 구조와 진행과정을 RNN 과 대조해서 확인하기 위해 각각을 보면 RNN - Recurrent Neural

Network 는 입력 개수 출력 개수에 따라서 one to many, many to one, many to many 로 나뉘지만 핵심적으로 대조되는 특징은 sequential 한 계산을 진행한다는 것인데, 이전 hidden state 의 아웃풋과 현시점의 인풋이 함께 연산 된다는 것 입니다. 문제 입력 시퀀스의 길이가 너무 길면 (Long sequence) gradient 를 주는 편미분 과정에서 (Back Propagation Through Time) gradient 가 vanishing 하거나 exploding 하는 문제가 생깁니다. 결국 다른 딥러닝 모델들과 마찬가지로 Back propagation 과정을 통해 Gradient 를 편미분하는데, 이때 발생하는 문제를 보면 RNN 의 구조상 입력 데이터의 길이가 길어지면 error 를 계산한 후, 처음 hidden state 까지 back propagation 하며 gradient 주는 과정이 너무 길기 때문에 gradient 값이 연산 과정에서 아주 작아져 버립니다. 결국 gradient 가 vanishing 하는 형태가 되고 맙니다. 그 결과 학습도 제대로 진행되지 않는 것을 알 수 있습니다.

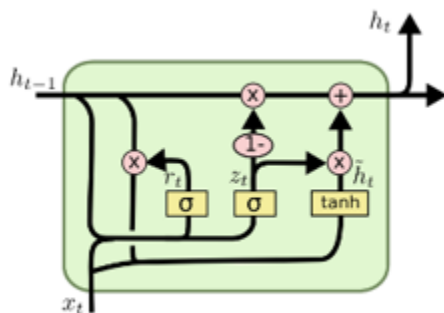
이에 대한 해결방안으로 이전 기억들도 계속 remind 하는 학습 방법이 필요한데, 이것이 LSTM 입니다.

LSTM 의 구조는 아래와 같고,





특징은 2 개의 벡터가 존재하며 단기 상태를 표현하는 h_t 와 장기 상태를 표현하는 C_t . 그리고, 3 개의 게이트 (input gate, output gate 그리고 forget gate)를 가지고 있다는 점이다. LSTM 셀에서는 상태가 h_t (단기 상태), C_t (장기 상태)로 2 개의 벡터로 나누어 진다는 것을 알 수 있습니다. 각 게이트를 정말 간단하게만 정리하자면 input gate 는 이번 입력을 얼마나 반영할지를 의미하고, output gate 는 이번 정보를 얼마나 내보낼지를 의미합니다. forget gate 는 과거 정보를 얼마나 까먹을지를 의미합니다. 그리고 이 모든 장기기억이 C_t 에 담김으로써 RNN 의 문제를 해결하는 것을 알 수 있습니다. GRU 는 위의 LSTM 의 복잡한 구조를 보다 간단한 구조로 만든 것으로 GRU 는 게이트가 2 개, LSTM 은 3 개입니다. GRU 에서는 reset gate, update gate 2 개의 gate 만을 사용합니다. 또한 cell state, hidden state 가 합쳐져 하나의 hidden state 로 표현하고 있습니다.



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \quad \text{---(1)}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \quad \text{---(2)}$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t]) \quad \text{---(3)}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \quad \text{---(4)}$$

reset gate 를 구하는 공식은 위 그림에서 (2)식에 해당됩니다. 이전 시점의 hidden state 와 현 시점의 x 를 활성화함수 시그모이드를 적용하여 구하는 방식입니다. 결과값은 0~1 사이의 값을 가질 것이며 이전 hidden state 의 값을 얼마나 활용할 것인지에 대한 정보로 해석할 수

있을 것입니다. reset gate 에서 나온 값은 그대로 사용되는 것이 아니라 (3) 식으로 다시 활용합니다. (3)식에서는 전 시점의 hidden state 에 reset gate 를 곱하여 계산합니다.

update gate 는 LSTM 의 input, forget gate 와 비슷한 역할을 하며 과거와 현재의 정보를 각각 얼마나 반영할지에 대한 비율을 구하는 것이 핵심입니다. (1)식을 통해서 구한 결과 z 는 현재 정보를 얼마나 사용할지 반영합니다. 그리고 $(1-z)$ 는 과거 정보에 대해서 얼마나 사용할지 반영합니다. 그래서 각 역할을 LSTM 의 input, forget gate 로 볼 수 있고 최종적으로 (4)식을 통해 현 시점의 출력값 hidden state 를 구할 수 있습니다.

GRU 는 기존 LSTM 에 비해 더 간단한 구조를 가지고 있습니다. 그리고 마지막 출력값에 활성화함수를 적용하지 않습니다. 성능 면에서는 LSTM 과 비교해서 우월하다고 할 수 없지만 학습할 파라미터가 더 적은 것이 장점이라고 할 수 있습니다.

둘사이의 차이점을 표로 보면 다음과 같습니다,

LSTM	GRU
gate 수 3 개(forget, input, output)	gate 수 2 개(reset, update)
Control the exposure of memory content (cell state)	Expose the entire cell state to other units in the network
Has separate input and forget gates	Performs both of these operations together via update gate
More parameters	Fewer parameters

그리고 또한 Whole simulation program flow를 보면 Keras dataset class를 사용한 IMDB standard dataset를 이용한 실습으로 num_words = 10000은 데이터 집합에 대해 10000개의 고유 단어만 사용됨을 의미합니다.

dataset을 load하고 이후에 tokenizer 와 Integer encoding을 진행합니다. Tokenizing 하는 preprocessing을 한 후에 texts_to_sequences를 진행하고 이후, pad_sequences 하여 x_train, x_test: 영화 리뷰 텍스트 데이터 목록. 길이가 일정하지 않음.

y_train, y_test : 정수 대상 레이블 목록 (1 또는 0) 과 같이 preprocessing하는 과정을 거칩니다. 이후에 input_length = maxlen 이미 pad_sequence를 사용하여 데이터 세트의 모든 문장의 길이를 200으로 동일하게 만들었기 때문에 임베딩 계층은

n_unique_words를 이미 10000으로 선언한 데이터 세트의 어휘 크기로 사용합니다.

Embedding 레이어 다음으로 양방향 LSTM 유닛을 추가하고 있습니다.

시그모이드 활성화를 사용한 다음 모델을 컴파일합니다. 이후 training하는 layer들의

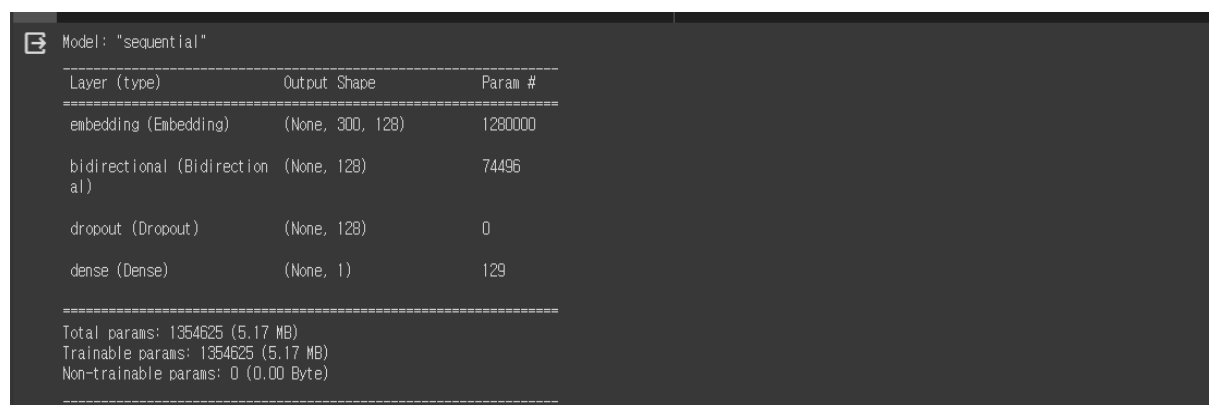
순서는 embedding layer -> bi-directional GPU -> dropout -> output layer의 순서로 layer를 쌓습니다. 그리고 조기 중단을 위한 모니터링과 검증 정확도를 기반으로 최상의 모델을 체크포인트로 하여 IMDb 데이터 세트에서 모델을 훈련시킵니다.

그리고 loss와 accuracy를 plot하는 과정을 거치는 Convergence Graph를 plot합니다. 이후 model을 evaluate하며 random값에 대한 리뷰와 함께 model을 test합니다. 그리고 model layer 쌓아서 training하기부터 random값에 대한 리뷰와 함께 model을 test하는 것까지의 과정을 GRU, LSTM 각각으로 다르게 비교해 볼 수 있도록 합니다.

<Explain the number of the parameters in each layers for model.summary()>

아래 result에서도 중복되는 결과화면 일 수 있지만 이용하여 설명하면 다음과 같습니다.

GRU)



```

Model: "sequential"
-----
Layer (type)                Output Shape              Param #
-----
embedding (Embedding)        (None, 300, 128)         1280000
bidirectional (Bidirectional) (None, 128)               74496
dropout (Dropout)            (None, 128)               0
dense (Dense)                 (None, 1)                 129
-----
Total params: 1354625 (5.17 MB)
Trainable params: 1354625 (5.17 MB)
Non-trainable params: 0 (0.00 Byte)
  
```

1번째 layer로는 Embedding layer로 Input Shape는 (None, 300)이며, 300은 `max_length`을 의미합니다. 그런데 출력되는 내용은 Output Shape으로 (None, 300, 128)로, 128 is the `embedding_size`를 의미합니다. Parameters의 개수는 `vocab_size * embedding_size` = 1,280,000 parameters이며 2번째 layer로는 Bidirectional GRU Layer로 Input Shape는 각각 (batch_size, input_length, output_dim) = (None, 300, 128)로 대응되며, Output Shape는 각각 (batch_size, 2 * hidden_states) = (None, 128 * 2)입니다. 그리고 parameters의 개수는 GRU $3 * \text{hidden_states} * (\text{output_dim} + \text{hidden_states} + 1) = 3 * 64 * (128 + 64 + 1) = 74,496$

Dropout Layer에서는 가중치가 없기 때문에 드롭아웃 계층에 매개변수가 없습니다. 단순히 입력에 드롭아웃을 적용합니다.

왜냐하면 드롭아웃은 훈련 중 입력 단위의 일부를 임의로 0으로 설정하여 과적합을 방지하는 정규화 기법으로 새로운 파라미터를 도입하지 않기 때문입니다. Output Shape은 (None, 1)이나 이진 분류 작업이므로 Parameters는 (128 (input_dim) + 1

(bias)) * 1 = 129 parameters입니다. dense layer는 이진 분류를 담당하는 출력 레이어이며, 각 입력 특징(이 경우 128)에 대한 가중치와 바이어스 항을 갖습니다.

LSTM)

```

Model: "sequential_1"
-----
Layer (type)                Output Shape              Param #
-----
embedding_1 (Embedding)     (None, 300, 128)         1280000
bidirectional_1 (Bidirecti  (None, 128)              98816
onal)
dropout_1 (Dropout)         (None, 128)              0
dense_1 (Dense)              (None, 1)                129
-----
Total params: 1378945 (5.26 MB)
Trainable params: 1378945 (5.26 MB)
Non-trainable params: 0 (0.00 Byte)
-----

```

1번째 layer로는 Embedding layer로 Input Shape는 (batch_size, input_length) = (None, 300)가 각각 대응됩니다. 그리고 출력되는 내용은 Output Shape으로 Output Shape: (batch_size, input_length, output_dim) = (None, 300, 128). Parameters의 개수는 `vocab_size * embedding_size` = 1,280,000 parameters이며 2번째 layer로는 Bidirectional LSTM Layer로 Input Shape는 각각 (batch_size, input_length, output_dim) = (None, 300, 128)로 대응되며, Output Shape는 각각 (batch_size, 2 * hidden_states) = (None, 128 * 2) 입니다. 그리고 parameters의 개수는 LSTM이므로 $4 * \text{hidden_states} * (\text{output_dim} + \text{hidden_states} + 1) = 4 * 64 * (128 + 64 + 1) = 74,496$

Dropout Layer 에서는 가중치가 없기 때문에 드롭아웃 계층에 매개변수가 없습니다. 단순히 입력에 드롭아웃을 적용합니다.

왜냐하면 드롭 아웃은 훈련 중 입력 단위의 일부를 임의로 0으로 설정하여 과적합을 방지하는 정규화 기법으로 새로운 파라미터를 도입하지 않기 때문입니다. Output Shape은 (None, 1) 이나 이진 분류 작업이므로 Parameters는 $(128 (\text{input_dim}) + 1 (\text{bias})) * 1 = 129 \text{ parameters}$ 입니다. dense layer는 이진 분류를 담당하는 출력 레이어이며, 각 입력 특징(이 경우 128)에 대한 가중치와 바이어스 항을 갖습니다.

<Simulation results (including Results of 5.)>

공통인 부분으로 data set 을 load 해오는 부분입니다.

```
1 #from keras.datasets import imdb
  #loading IMDB standard dataset using the Keras dataset class.
  (ds_train, ds_test), ds_info = tfds.load('imdb_reviews',
                                          split=['train', 'test'], # + 'unsupervised'
                                          shuffle_files=True,
                                          as_supervised=True,
                                          with_info=True)

  print(ds_info.features)

[2] Downloading and preparing dataset 80.23 MiB (download: 80.23 MiB, generated: Unknown size, total: 80.23 MiB) to /root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0...
  DI Completed.: 100% 1/1 [00:02<00:00, 2.72s/url]
  DI Size.: 100% 80/80 [00:02<00:00, 55.83 MiB/s]

Dataset imdb_reviews downloaded and prepared to /root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0. Subsequent calls will reuse this data.
FeaturesDict({
  'label': ClassLabel(shape=(), dtype=int64, num_classes=2),
  'text': Text(shape=(), dtype=string),
})
```

그리고 dataset 을 맞는 형태로 preprocessing 합니다.

x_train, x_test: 영화 리뷰 텍스트 데이터 목록. 길이가 일정하지 않음.

y_train, y_test : 정수 대상 레이블 목록 (1 또는 0) 과 같이 preprocessing 하는 과정을 거칩니다.

```
[4] # pre-processing the dataset
  x_train = []
  y_train = []

  x_test_str = []
  y_test = []
  # Extract training data and store it in the list
  for sentence, label in ds_train:
    x_train.append(sentence.numpy().decode('utf8'))
    y_train.append(label.numpy())
  # Extract test data and store it in the list
  for sentence, label in ds_test:
    x_test_str.append(sentence.numpy().decode('utf8')) # x_test_str is used at the test stage
    y_test.append(label.numpy())
  # Convert labels to NumPy arrays
  y_train = np.array(y_train)
  y_test = np.array(y_test)

[5] # print the sample data, x_train[0]
  print(x_train[0])
  print('The review is', 'Positive' if y_train[0]==1 else 'Negative')

This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Both are great actors, but this must simply be their worst role in history. Even their great acting could not redeem this movie's ridiculous storyline. This
The review is Negative
```

GRU)

Embedding layer, Bidirectional layer, Dropout layer, Dense layer의 순서로 layer를 쌓고 model.summary()를 통해 그 결과를 확인합니다.

```
# This cell uses the GRU model.
# Import the packages required for the job.
from keras.layers import Embedding, Bidirectional, Dense
import tensorflow as tf

hidden_states = 64
dropout_rate = 0.5

model = tf.keras.Sequential()
# Embedded layer: transforming words into embedded vectors
model.add(tf.keras.layers.Embedding(input_dim=vocab_size, output_dim=embedding_size, input_length=max_length)) # embedding layer
# Bidirectional GRU layer: Bidirectional recurrent neural networks
model.add(tf.keras.layers.Bidirectional(tf.keras.layers.GRU(hidden_states))) # bidirectional GRU
# Dropout layer: Apply dropout to prevent overfitting
model.add(tf.keras.layers.Dropout(dropout_rate)) # dropout
# Output layer: Using the sigmoid activation function for binary classification
model.add(tf.keras.layers.Dense(1, activation='sigmoid')) # output layer
# Compilation of models: setting loss functions, optimizers, and evaluation metrics
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Model Summary Output
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 300, 128)	12800000
bidirectional (Bidirectional)	(None, 128)	74496
dropout (Dropout)	(None, 128)	0
dense (Dense)	(None, 1)	129

Total params: 1354625 (5.17 MB)
Trainable params: 1354625 (5.17 MB)
Non-trainable params: 0 (0.00 Byte)

training 하는데 최적 accuracy 일때를 찾기 위해 breakpoint 등의 장치를 통해서 결과를 확인합니다.

```
print(results.history['loss'])
print(results.history['accuracy'])
```

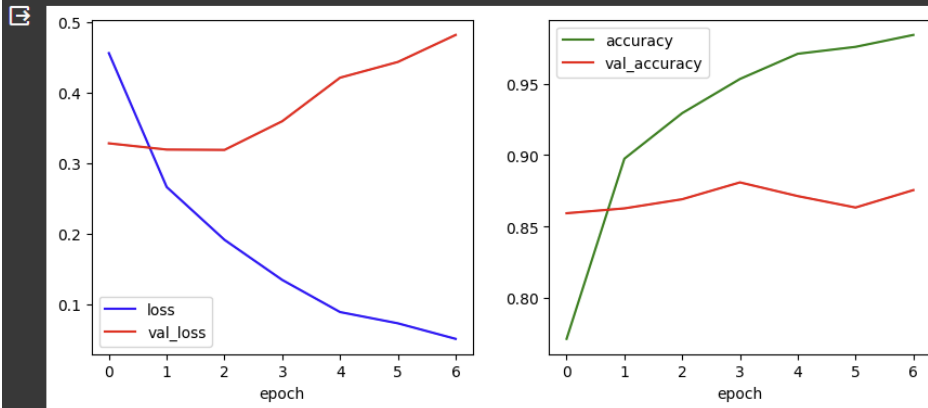
```
Epoch 1/10
313/313 [=====] - ETA: 0s - loss: 0.4555 - accuracy: 0.7712
Epoch 1: val_accuracy improved from -inf to 0.85920, saving model to GRU_ind.h5
313/313 [=====] - 42s 100s/step - loss: 0.4555 - accuracy: 0.7712 - val_loss: 0.3278 - val_accuracy: 0.8592
Epoch 2/10
1/313 [.....] - ETA: 34s - loss: 0.3390 - accuracy: 0.8281/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via 'model.save()'. This file format is obso
saving_api.save_model(
313/313 [=====] - ETA: 0s - loss: 0.2660 - accuracy: 0.8974
Epoch 2: val_accuracy improved from 0.85920 to 0.86260, saving model to GRU_ind.h5
313/313 [=====] - 20s 63ms/step - loss: 0.2660 - accuracy: 0.8974 - val_loss: 0.3190 - val_accuracy: 0.8626
Epoch 3/10
313/313 [=====] - ETA: 0s - loss: 0.1912 - accuracy: 0.9293
Epoch 3: val_accuracy improved from 0.86260 to 0.89300, saving model to GRU_ind.h5
313/313 [=====] - 13s 41ms/step - loss: 0.1912 - accuracy: 0.9293 - val_loss: 0.3185 - val_accuracy: 0.8930
Epoch 4/10
313/313 [=====] - ETA: 0s - loss: 0.1343 - accuracy: 0.9532
Epoch 4: val_accuracy improved from 0.89300 to 0.88080, saving model to GRU_ind.h5
313/313 [=====] - 11s 35ms/step - loss: 0.1343 - accuracy: 0.9532 - val_loss: 0.3591 - val_accuracy: 0.8808
Epoch 5/10
313/313 [=====] - ETA: 0s - loss: 0.0889 - accuracy: 0.9709
Epoch 5: val_accuracy did not improve from 0.88080
313/313 [=====] - 10s 33ms/step - loss: 0.0889 - accuracy: 0.9709 - val_loss: 0.4208 - val_accuracy: 0.8712
Epoch 6/10
313/313 [=====] - ETA: 0s - loss: 0.0729 - accuracy: 0.9759
Epoch 6: val_accuracy did not improve from 0.88080
313/313 [=====] - 10s 31ms/step - loss: 0.0729 - accuracy: 0.9759 - val_loss: 0.4430 - val_accuracy: 0.8632
Epoch 7/10
313/313 [=====] - ETA: 0s - loss: 0.0509 - accuracy: 0.9841
Epoch 7: val_accuracy did not improve from 0.88080
313/313 [=====] - 9s 30ms/step - loss: 0.0509 - accuracy: 0.9841 - val_loss: 0.4813 - val_accuracy: 0.8754
Epoch 7: early stopping
[0.4555075764566067, 0.26598814123929407, 0.19119088351726532, 0.13434424966376038, 0.08887707442045212, 0.073497238593285, 0.05032321642756452]
[0.7712000012397766, 0.857400215530936, 0.9230000102043152, 0.9532499940094, 0.970999991416931, 0.9757500030087067, 0.984099984163063]
```

Loss 와 accuracy 를 graph 를 통해 plot 합니다.


```

plt.xlabel('epoch')
plt.legend()
# Second subplot: Accuracy Graph
plt.subplot(1,2,2)
plt.plot(results.history['accuracy'], 'g-', label='accuracy')
plt.plot(results.history['val_accuracy'], 'r-', label='val_accuracy')
plt.xlabel('epoch')
plt.legend()
# print the graph
plt.show()

```



Model 을 평가한 것을 출력합니다.

```

# model = tf.keras.models.load_model('GRU_imdb.h5')
model.evaluate(X_test, y_test)

782/782 [=====] - 7s 9ms/step - loss: 0.5468 - accuracy: 0.8631
[0.5468241572380066, 0.8631200194358826]

```

random 값을 통해 test 한 결과를 출력합니다.

```

# select the random index
idx = np.random.randint(X_test.shape[0])
# Reshaping the selected test data to enter the model
X_inout = X_test[idx].reshape(1,-1)
# Use the model to predict the positivity score of the selected review
score = float(model.predict(X_inout))
# Decide positively or negatively for reviews based on predicted scores
decision = 1 if score>0.5 else 0
# Use the score if positive and subtract the score from 1 if negative to calculate the opposite score
rate = score if decision==1 else (1-score)
# print the selected review text
print(X_test_str[idx])
# print the decision and score
print('The review is', 'Positive' if decision==1 else 'Negative', 'in {:.2f}%'.format(rate*100))

#print(X_inout.shape)

1/1 [=====] - 1s 584ms/step
Engaging entry from Europe about Czech fighter pilots flying for the RAF during WW2. It's always interesting as an American to see a new point of view on familiar events in history. There's nothing terribly original or revolutionary about the style in which
The review is Positive in 99.82%

```

LSTM)

위에서 layers쌓기부터 random값을 통해 test한 결과를 출력하는 point까지 LSTM을 통해서 출력합니다.

```
# This cell uses the LSTM model.
# Import the packages required for the job.
from keras.layers import Embedding, Bidirectional, Dense
import tensorflow as tf

hidden_states = 64
dropout_rate = 0.5

model = tf.keras.Sequential()
# Embedded layer: transforming words into embedded vectors
model.add(tf.keras.layers.Embedding(input_dim=vocab_size, output_dim=embedding_size, input_length=max_length)) # embedding layer
# Bidirectional LSTM layer: Bidirectional recurrent neural networks
model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(hidden_states))) # bidirectional GRU
# Dropout layer: Apply dropout to prevent overfitting
model.add(tf.keras.layers.Dropout(dropout_rate)) # dropout
# Output layer: Using the sigmoid activation function for binary classification
model.add(tf.keras.layers.Dense(1, activation='sigmoid')) # output layer
# Compilation of models: setting loss functions, optimizers, and evaluation metrics
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Model Summary Output
model.summary()
```

Model: "sequential_1"

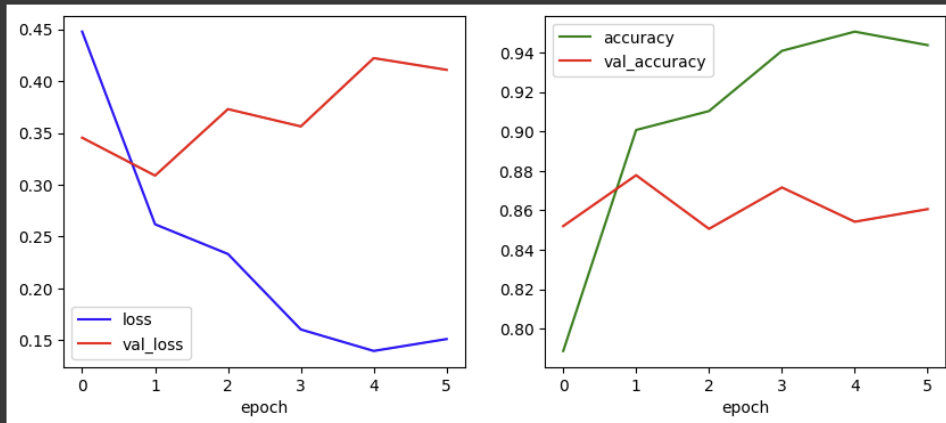
Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 300, 128)	1280000
bidirectional_1 (Bidirectional)	(None, 128)	98816
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

=====
 Total params: 1378945 (5.26 MB)
 Trainable params: 1378945 (5.26 MB)
 Non-trainable params: 0 (0.00 Byte)
 =====

```
# tf.keras.callbacks.ModelCheckpoint('GPU_Leb_16', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)
# Model training
results = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=10,
                    callbacks=[es, mc],
                    validation_split=0.2)
# Outpuitng Training Results
print(results.history['loss'])
print(results.history['accuracy'])
```

Epoch 1/10
 313/313 [=====] - ETA: 0s - loss: 0.4478 - accuracy: 0.7886
 Epoch 1: val_accuracy improved from -inf to 0.8500, saving model to GPU_Leb_16
 313/313 [=====] - 39s 111ms/step - loss: 0.4478 - accuracy: 0.7886 - val_loss: 0.3454 - val_accuracy: 0.8500
 Epoch 2/10
 /usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an H5 file via 'model.save()'. This file format is considered legacy. We recommend using instead the native Keras format, e.g. 'model.save(format='keras')'.
 saving_model.save_model(
 313/313 [=====] - ETA: 0s - loss: 0.2618 - accuracy: 0.9007
 Epoch 2: val_accuracy improved from 0.8500 to 0.8780, saving model to GPU_Leb_16
 313/313 [=====] - 19s 59ms/step - loss: 0.2618 - accuracy: 0.9007 - val_loss: 0.3088 - val_accuracy: 0.8778
 Epoch 3/10
 313/313 [=====] - ETA: 0s - loss: 0.2332 - accuracy: 0.9104
 Epoch 3: val_accuracy did not improve from 0.8780
 313/313 [=====] - 13s 43ms/step - loss: 0.2332 - accuracy: 0.9104 - val_loss: 0.3730 - val_accuracy: 0.8505
 Epoch 4/10
 313/313 [=====] - ETA: 0s - loss: 0.1603 - accuracy: 0.9409
 Epoch 4: val_accuracy did not improve from 0.8780
 313/313 [=====] - 11s 35ms/step - loss: 0.1603 - accuracy: 0.9409 - val_loss: 0.3553 - val_accuracy: 0.8716
 Epoch 5/10
 313/313 [=====] - ETA: 0s - loss: 0.1306 - accuracy: 0.9506
 Epoch 5: val_accuracy did not improve from 0.8780
 313/313 [=====] - 10s 33ms/step - loss: 0.1306 - accuracy: 0.9506 - val_loss: 0.4222 - val_accuracy: 0.8542
 Epoch 6/10
 313/313 [=====] - ETA: 0s - loss: 0.1511 - accuracy: 0.9438
 Epoch 6: val_accuracy did not improve from 0.8780
 313/313 [=====] - 10s 33ms/step - loss: 0.1511 - accuracy: 0.9438 - val_loss: 0.4109 - val_accuracy: 0.8605
 Epoch 6: early stopping
 [0.44773247957236, 0.2618422210216522, 0.2331787794828415, 0.16030624508857727, 0.13959068059621365, 0.15106603176787231]
 [0.7886499762535095, 0.9007499814033508, 0.9103500247001648, 0.9409000277519226, 0.9506499767303467, 0.9437999725341797]

```
# First subplot: Loss Graph
plt.subplot(1,2,1)
plt.plot(results.history['loss'], 'b-', label='loss')
plt.plot(results.history['val_loss'], 'r-', label='val_loss')
plt.xlabel('epoch')
plt.legend()
# Second subplot: Accuracy Graph
plt.subplot(1,2,2)
plt.plot(results.history['accuracy'], 'g-', label='accuracy')
plt.plot(results.history['val_accuracy'], 'r-', label='val_accuracy')
plt.xlabel('epoch')
plt.legend()
# print the graph
plt.show()
```



```
# model = tf.keras.models.load_model('GRU_imdb.h5')
model.evaluate(X_test, y_test)

782/782 [=====] - 12s 16ms/step - loss: 0.4251 - accuracy: 0.8580
[0.4250820279121399, 0.8579599857330322]
```

```
# select the random index
idx = np.random.randint(X_test.shape[0])
# Refreshing the selected test data to enter the model
X_input = X_test[idx].reshape(1,-1)
# Use the model to predict the positivity score of the selected review
score = float(model.predict(X_input))
# Decide positively or negatively for reviews based on predicted scores
decision = 1 if score>0.5 else 0
# Use the score: if positive and subtract the score from 1 if negative to calculate the opposite score
rate = score if decision==1 else (1-score)
# print the selected review text
print(X_test_str[idx])
# print the decision and score
print('The review is', 'Positive' if decision==1 else 'Negative', 'in {:.2f}%'.format(rate*100))

# print(X_input.shape)

1/1 [=====] - 1s 66ms/step
When the movie first started I thought cheesy. The first ten minutes were really boring. After the slow beginning and some of the soap opera antics, I started liking it. The plot was different than anything I had ever seen. Now, was it a horror? Not really.
The review is Negative in 99.27%
```

<Discussion (Compare Results of 5. and Training)>

LSTM 과 GRU 를 비교하면 LSTM 에서 보면 loss 와 accuracy 가 각각 0.4251 과 0.8580 이고, 최종 random 값에 대한 99.27%이었고 GRU 는 loss 와 accuracy 가 각각 0.5468 과 0.8631 이고, 최종 random 값에 대한 99.82%으로 둘 다 random 값의 test data 에 대해서 좋은 accuracy 값을 가지는 것을 알 수 있었습니다. 그렇다면 둘의 차이를 대조할 만한 내용을 보면 GRU 와 LSTM 은 gradient vanishing 문제를 방지하기 위해 정보 gating 에 대해 서로 다른 접근 방식을 활용하는데 두 가지를 비교하는 주요 포인트는 다음과 같습니다.

GRU 장치는 LSTM 장치와 같이 정보의 흐름을 제어하지만 메모리 장치를 사용할 필요가 없습니다. 아무런 통제 없이 숨겨진 콘텐츠 전체를 노출할 뿐입니다. GRU 는 상대적으로 새로운 것이며 내 경험에 따르면 성능은 LSTM 과 비슷하지만 계산상 더 효율적입니다. (구조가 덜 복잡합니다.) 따라서 이러한 이유로 GRU 가 보다 더 많이 사용되고 있는 것으로 나타났습니다.