인공지능프로그래밍
Lab 05: Sequence Model of GRU

학 번 : 2021202058

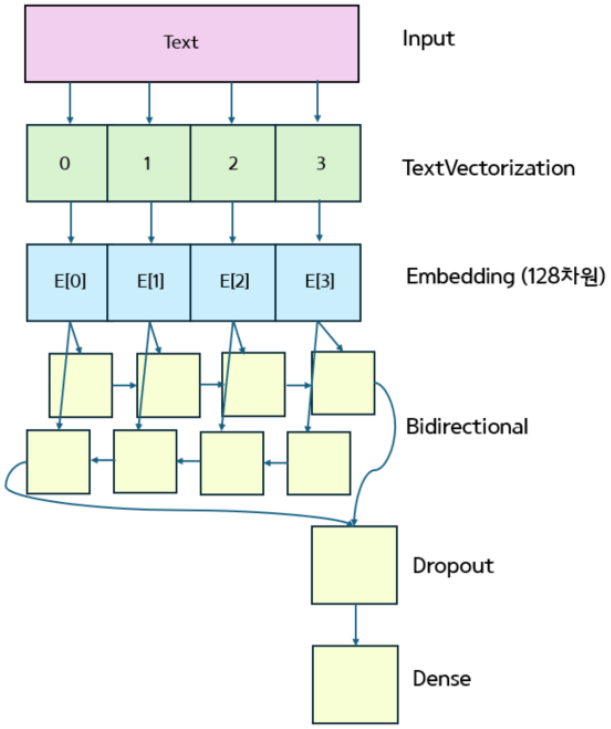
성 명 : 송채영

날 짜 : 2024.10.31

Lab Objective

이번 과제에서는 TensorFlow의 Keras 라이브러리를 사용하여 GRU(Gated Recurrent Unit) 모델을 구현해보는 것이다. GRU는 순차 데이터를 처리하는 데 있어 LSTM과 유사하지만 구조가 더 간단하다. GRU 모델을 만들기 위해 레이어를 쌓고 텍스트 전처리를 위한 TextVectorization과 단어표헌을 하기위한 Embedding을 구현해보고, Keras의 IMDb reiview 데이터세트를 사용하여 영화 리뷰가 긍정적인지 부정적인지를 예측해본다.

Program flow



전체적인 코드의 흐름이다. 영화 리뷰 한 문장인 텍스트를 입력하면 각 단어를 정수 시퀀스로 변환하여 텍스트로 토큰화한다. 이 부분이 TextVectorization에 해당한다. 각 토큰을 고정된 크기, 이번 프로젝트에서는 128에 해당하며 밀집된 벡터(워드 임베딩)에 매핑하는 Embedding 과정을 거친다. 이어서 양방향 GRU를 사용하여 앞 뒤에서 컨텍스트를 캡처한 후 정규화를 위한 드롭아웃을 추가하고 마지막으로 dense layer를 거친다. Dense layer에서는 이진 분류 확률을 나타내기 위해 sigmoid 활성화 함수를 사용하여 0과 1 사이의 단일 값을 출력하도록 한다. 더 자세한 설명은 아래 코드에서 진행하도록 하겠다.

Result

```
[ ]  import os
     # Set the backend of Keras to TensorFlow
     os.environ["KERAS_BACKEND"] = "tensorflow"

     import numpy as np
     import tensorflow as tf
     import tensorflow_datasets as tfds
     import keras
     import matplotlib.pyplot as plt
```

Loading IMDB standard dataset using the Keras dataset class.

num_words = 10000 signifies that only 10000 unique words will be taken for our dataset.

x_train, x_test: List of movie reviews text data. having an uneven length.

y_train, y_test: Lists of integer target labels (1 or 0).

필요한 모듈을 import 하고 os.environ["KERAS_BACKEND"] = "tensorflow"를 설정해주며 keras의 백엔드를 명시적으로 TensorFlow로 설정하는 코드이다.

```
#from keras.datasets import imdb
# Import the IMDB reviews dataset from TensorFlow Datasets
(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)
```

```
Downloading and preparing dataset 80.23 MiB (download: 80.23 MiB, generated: Unknown size, total: 80.23 MiB) to /root/t
Dl Completed...: 100% ████ 1/1 [00:06<00:00,  6.58s/ url]
Dl Size...: 100% ████ 80/80 [00:06<00:00, 12.75 MiB/s]
Dataset imdb_reviews downloaded and prepared to /root/tensorflow_datasets/imdb_reviews/plain_text/1.0.0. Subsequent cal
FeaturesDict({
    'label': ClassLabel(shape=(), dtype=int64, num_classes=2),
    'text': Text(shape=(), dtype=string),
})
```

TensorFlow 데이터 세트에서 IMDb review 데이터세트를 로드하고 이를 train set와 test set로 분할하고 이에 대한 정보를 print 하는 코드이다. 클래스 라벨과 텍스트 데이터의 정보가 출력되었으며 클래스의 경우 2개가 있음을 알 수 있다.

```
# Initialize lists to store training and testing data
X_train = []
y_train = []

X_test_str = []
y_test = []

# Process the training dataset
for sentense, label in ds_train:
    X_train.append(sentense.numpy().decode('utf8'))
    y_train.append(label.numpy())

# Process the testing dataset
for sentense, label in ds_test:
    X_test_str.append(sentense.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)
```

```
# Print the first training review
print(X_train[0])
# Print whether the first review is positive or negative based on its label
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 acto
The review is Negative
```

Train, test 데이터를 처리하는 코드이다. Train과 test를 각각 처리 후 numpy 배열로 변환하였다. 이후 train 데이터의 첫 번째 리뷰에 대한 정보를 출력한 것으로 감정 레벨이 0, Negative인 것을 알 수 있다.

```
# hyperparameter for word embeddings
vocab_size = 10000    # Vocabulary size
embedding_size = 128  # Dimension of each word embedding
max_length = 300      # Maximum length of each input sequence

### START CODE HERE ###
# Initialize tokenizer to vectorize text into integer sequences
tokenizer = keras.layers.TextVectorization(max_tokens=vocab_size,output_mode='int',output_sequence_length=max_length)
# Adapt tokenizer to training data vocabulary
tokenizer.adapt(X_train)

# Transform texts into integer sequences
X_train = tokenizer(X_train)
X_test  = tokenizer(X_test_str)

### END CODE HERE ###
```

input_length = maxlen Since we have already made all sentences in our dataset have an equal length of 200 using pad_sequence. The Embedding layer takes n_unique_words as the size of the vocabulary in our dataset which we already declared as 10000. After the Embedding layer, we are adding Bi-directional LSTM units. Using sigmoid activation and then compiling the model

우선 단어 임베딩을 위한 하이퍼파라미터를 정의하였다. Vacab_size는 토크나이저가 데이터세트에서 보유할 단어의 최대 수를 정의하며 10,000으로 설정되어 있다. Embedding_size의 경우 단어 임베딩의 크기, 차원을 설정하며 128로 정의되어 있다. Max_length는 각 입력 시퀀스를 최대 300개의 토큰 길이로 제한한다. 따라서 300보다 큰 경우 자르고 작은 경우 0으로 패딩 됨을 알 수 있다. 이후 TextVectorization 레이어를 토크나이저로 초기화한다. 토크나이저의 단어 크기를 10,000로 제한하고 int로 변환한다. 이후 각 시퀀스가 300개의 토큰으로 채워지거나 잘리는지를 확인한 후 토크나이저를 x_train의 단어에 맞춘다. 마지막으로 x_train과 x_test를 정수 시퀀스로 토큰화한다.

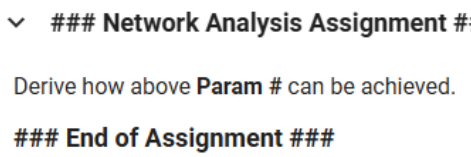
```
[6] # Hyperparameter for model
    hidden_states = 64
    dropout_rate = 0.5

    model = keras.Sequential() # Initialize a sequential model

    ### START CODE HERE ###

    model.add(keras.layers.InputLayer(shape=(max_length, ))) # input layer
    model.add(keras.layers.Embedding(input_dim=vocab_size, output_dim=embedding_size))  # embedding layer
    model.add(keras.layers.Bidirectional(keras.layers.GRU(hidden_states)))  # bidirectional GRU layer
    model.add(keras.layers.Dropout(dropout_rate))  # dropout layer
    model.add(keras.layers.Dense(1, activation='sigmoid'))  # output layer

    ### END CODE HERE ###
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy']) # Compile the model

    model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 300, 128)	1,280,000
bidirectional (Bidirectional)	(None, 128)	74,496
dropout (Dropout)	(None, 128)	0
dense (Dense)	(None, 1)	129

Total params: 1,354,625 (5.17 MB)
Trainable params: 1,354,625 (5.17 MB)
Non-trainable params: 0 (0.00 B)

이어서 sequential모델을 정의하고 컴파일하는 코드에 대한 설명이다. Hidden sate는 64로 설정되었고 dropout_rate는 0.5로 설정되었다. 즉 훈련 중에 뉴런의 50%가 무작위로 삭제됨을 의미한다. 이어서 모델을 초기화 한 후 주석의 조건에 맞추어 필요한 레이어를 추가하였다. 먼저 입력 레이어는 max_length, 300개 토큰의 시퀀스가 된다. 이어서 임베딩 레이어는 정수로 인코딩된 단어를 고정 크기의 벡터(128차원 벡터)로 변환한다. 이때 input_dim은 가능한 토큰 수이고 output_dim은 임베딩 크기로 각각 10,000, 128이다. 이어서 양방향 GRU 계층을 추가하였다. 과거 단어와 미래 단어 모두에서 컨텍스트를 캡처하기 위해 양방향으로 시퀀스를 읽으며 이때 hidden sate가 장치 수를 결정한다. 이어서 dropout레이어와 dense레이어를 거친다. Dense 레이어에서는 이진 분류를 위한 sigmoid 활성화 함수가 있는 단일 뉴런 계층으로 0과 1 사이의 확률을 출력한다. 이어서 모델을 컴파일한다. 손실함수인 이진 교차 엔트로피, 최적화로 Adam, 메트릭인 정확도로 컴파일된다. 이후 모델의 레이어, 출력 모양, 매개변수 수에 대한 요약을 확인하였고, expected output과 동일함을 확인하였다.



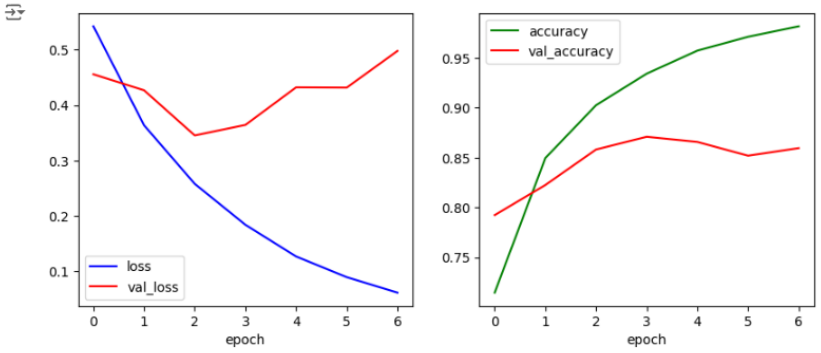
이어서 위 출력결과의 파라미터를 얻는 과정을 설명해보겠다. 우선 임베딩 레이어에서 input_dim = 10000, output_dim = 128이므로 출력 형태는 (none, 300, 128)이 된다. Vocab_size, 10000개는 각각의 128차원 벡터에 매핑되며 총 매개변수는 vocab_size x embedding_size = 10000 x 128 = 1280000이 된다. 양방향 GRU 레이어에서는 bidirectional(GRU(단위=64), 출력형태는 (none, 128)이 된다. 이때 128이 되는 이유는 양방향 레이어이기 때문에 출력 형태가 2배가 된다. GRU 셀에는 3개의 게이트(update, reset, next hidden state)가 있다. 따라서 매개변수를 구하면 (input_dim + hidden sate) * hidden_sate * 3 *2가 된다. 이때 x3을 하는 이유는 GRU셀에는 3개의 게이트가 존재하기 때문이며 x2를 하는 이유는 bidirectional이기 때문이다. 그리고 이때 bias term도 각각 존재하므로 이를 다 계산하면 74496이 된다. 이어서 dropout 레이어는 출력형태가 (none, 128)이 된다. 해당 레이어에서는 학습 가능한 매개변수가 없기 때문에 매개변수가 0이다. 다음으로 dense 레이어에서 출력 형태는 (none, 1)이다. 이전 GRU 계층의 128개 출력을 각각 sigmoid 활성화를 통해 단일 출력 장치에 연결하며, 가중치는 128x1로 128, bias는 1이므로 129가 나오게 된다. 이를 다 합산하면 전체 매개변수가 나오게 된다.

```
n_batch = 64

# Set up early stopping
es = keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
# Set up model checkpointing
mc = keras.callbacks.ModelCheckpoint('GRU_imdb.keras', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)

# Fit the model
results = model.fit(X_train, y_train,
            batch_size=n_batch,
            epochs=10,
            callbacks=[es, mc],
            validation_split=0.2)

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

```
Epoch 1/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.6210 - loss: 0.6199
Epoch 1: val_accuracy improved from -inf to 0.79240, saving model to GRU_imdb.keras
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 29ms/step - accuracy: 0.6213 - loss: 0.6196 - val_accuracy: 0.7924 - val_loss: 0.4556
Epoch 2/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.8504 - loss: 0.3692
Epoch 2: val_accuracy improved from 0.79240 to 0.82260, saving model to GRU_imdb.keras
313/313 ━━━━━━━━━━━━━━━━━━━━ 10s 33ms/step - accuracy: 0.8504 - loss: 0.3691 - val_accuracy: 0.8226 - val_loss: 0.4267
Epoch 3/10
311/313 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.9019 - loss: 0.2592
Epoch 3: val_accuracy improved from 0.82260 to 0.85820, saving model to GRU_imdb.keras
313/313 ━━━━━━━━━━━━━━━━━━━━ 19s 27ms/step - accuracy: 0.9019 - loss: 0.2592 - val_accuracy: 0.8582 - val_loss: 0.3453
Epoch 4/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.9375 - loss: 0.1809
Epoch 4: val_accuracy improved from 0.85820 to 0.87100, saving model to GRU_imdb.keras
313/313 ━━━━━━━━━━━━━━━━━━━━ 10s 27ms/step - accuracy: 0.9375 - loss: 0.1809 - val_accuracy: 0.8710 - val_loss: 0.3643
Epoch 5/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.9614 - loss: 0.1190
Epoch 5: val_accuracy did not improve from 0.87100
313/313 ━━━━━━━━━━━━━━━━━━━━ 9s 29ms/step - accuracy: 0.9614 - loss: 0.1190 - val_accuracy: 0.8658 - val_loss: 0.4320
Epoch 6/10
311/313 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.9770 - loss: 0.0735
Epoch 6: val_accuracy did not improve from 0.87100
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 44ms/step - accuracy: 0.9769 - loss: 0.0736 - val_accuracy: 0.8520 - val_loss: 0.4316
Epoch 7/10
312/313 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.9854 - loss: 0.0546
Epoch 7: val_accuracy did not improve from 0.87100
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 27ms/step - accuracy: 0.9853 - loss: 0.0547 - val_accuracy: 0.8596 - val_loss: 0.4977
Epoch 7: early stopping
[0.542168140411377, 0.363614261503601, 0.2579622566699981 7, 0.184075787663459 78, 0.126943916082382 2, 0.089582122862339 02, 0.061710536479949 95]
[0.714649975299835 2, 0.849650025367736 8, 0.902649998664856, 0.934300005435943 6, 0.957400023937225 3, 0.971300005912780 8, 0.981800019741058 3]
```

배치 크기는 64로 설정하였고 validation loss를 살펴보며 4회 연속 epoch 동안 개선되지 않으면 early stopping을 설정하였다. 또한 validation accuracy가 향상될 때 마다 모델을 파일에 저장하며 이때 정확도가 높아질 때만 저장하도록 설정하였다. 에폭 7에서 loss가 개선되지 않아 early stopping한 것을 확인할 수 있다.

```
# plot loss and accuracy
plt.figure(figsize=(10,4))
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()

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()

plt.show()
```



위 코드는 loss와 accuracy를 matplotlib의 plt를 사용해 그래프로 출력하는 코드로 accuracy와 validation accuracy을 시각화하였다. Validation loss의 경우 train loss에 비해 낮아지지 않는 것을 확인할 수 있고 validation accuracy의 경우 train accuracy에 비해서는 좋게 나오지는 않았다.

Evaluate the model

```
[9]  # model = keras.models.load_model('GRU_imdb.keras')
     model.evaluate(X_test, y_test) # Evaluate the model
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - accuracy: 0.8289 - loss: 0.5871
[0.5812024474143982, 0.8278800249099731]
```

위에서 사전 훈련한 모델을 불러와 test 데이터 세트에서 평가를 진행하는 코드이다. Loss의 경우 0.5871이 나오고 accuracy의 경우 0.8239가 나오는 것을 확인하였다.

```
[10] idx = np.random.randint(X_test.shape[0]) # Select a random index

     X_input = tf.reshape(X_test[idx], shape=(1,-1)) # Reshape the selected test sample for model input

     score = float(tf.squeeze(model.predict(X_input))) # Predict the score

     # Determine the decision based on the score
     decision = 1 if score>0.5 else 0
     # Calculate the rate of confidence for the prediction
     rate = score if decision==1 else (1-score)

     print(X_test_str[idx])

     print('The review is', 'Positive' if decision==1 else 'Negative', 'in {:.2f}% '.format(rate*100))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 160ms/step
Idiots go camping and act like idiots before they finally die like idiots, yes Camp Blood (or if you're wanting an awful,
The review is Negative in 99.94%
```

X_test 배열의 인덱스를 무작위로 선택하고 선택된 테스트 샘플의 모양을 reshape 한 후 훈련된 모델을 사용하여 예측하는 코드이다. Score를 바탕으로 decision과 rate를 구하고 이를 출력하여 확인하였을 때 결과가 잘 나오는 것을 확인할 수 있다.

Discussion

RNN, LSTM, GRU를 개념적으로만 공부하다가 이번 기회를 통해 더욱 자세히 알 수 있었다. 수업을 들을 때는 다른 수업에서도 이미 배웠던 내용이므로 다 이해했다고 생각했는데 이를 직접 구현해보려고 하니, 헷갈리는 부분이 있었던 것 같다. 이를 이해하기 위해 여러 블로그와 티스토리를 찾아보며 이해하려고 했던 것 같다. 특히 parameter를 계산해서 왜 그렇게 나왔는지 생각하는데 시간을 가장 많이 썼던 것 같다. 결론적으로는 Bias term을 더해주지 않아 계산한 값이 맞지 않았던 것이었는데, 이를 찾아내는 점이 어려웠다. 구하는 법은 구글링을 통해 얻게 된 것인데, 강의자료의 수식으로도 구할 수 있을 것 같으나 방법을 찾지는 못하였다. 또한 토크나이저를 하는 과정에서 최대 길이를 기준으로 더 길면 자르고 더 짧으면 0으로 패딩한다는 개념도 이해하는데 조금 어려움이 있었지만 코드를 구현하고 보고서를 작성하는 과정에서 완벽하게 이해할 수 있었다.