자연어처리과제#3

20011614 성현주

#3.1 Text Classification with RNN model

- Modeling Bidirectional GRU

모델 구조

1) Embedding Layer(64 dimension)
2) Bidirectional LSTM Layer(32 units)
3) Bidirectional LSTM Layer(16 units)
4) Fully Connected Layer(16 units, relu activation)
5) Dropout Layer(0.5)
6) Fully Connected Layer(2 output layer unit, softmax)

Optimizer: RMSprop(learning rate 0.001)

학습 상세

overfitting 규제: Earlystopping 사용 (validation set 손실 값이 더 이상 증가하지 않을 때 학습중단)

Batch size 128, 5 epochs, validation split=0.2

#3.1 Text Classification with RNN model

```python
from tensorflow.keras import layers
from tensorflow.keras import Sequential

from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.optimizers import Adam, AdamW, RMSprop

model = Sequential()

model.add(layers.Embedding(max_features+1, 64, input_shape=(max_len, )))
'''
Bidirectional 파라미터
merge_mode: mode by which outputs of the forward and backward RNNs will be combined.
One of {sum/mul/concat(default)/ave/None} If None, the outputs will not be combined, they will be returned as a list.
'''
# return_sequence=True : whether to return the last output in the output sequence, or the full sequence
# return_state=True : whether to return the last state in addition to the output.
model.add(layers.Bidirectional(layers.LSTM(32, return_sequences=True), merge_mode='concat'))
model.add(layers.Bidirectional(layers.LSTM(16)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(2, activation='softmax')) #binary classification

model.compile(optimizer=RMSprop(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])
model.summary()

es = EarlyStopping(monitor='val_loss', mode='min', verbose = 1, patience=2)
```

1.7s

```
Model: "sequential_23"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_23 (Embedding)    (None, 40, 64)            640064

 bidirectional_34 (Bidirect  (None, 40, 64)            24832
 ional)

 bidirectional_35 (Bidirect  (None, 32)                10368
 ional)

 dense_27 (Dense)            (None, 16)                528

 dropout_4 (Dropout)         (None, 16)                0

 dense_28 (Dense)            (None, 2)                 34

=================================================================
Total params: 675826 (2.58 MB)
Trainable params: 675826 (2.58 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

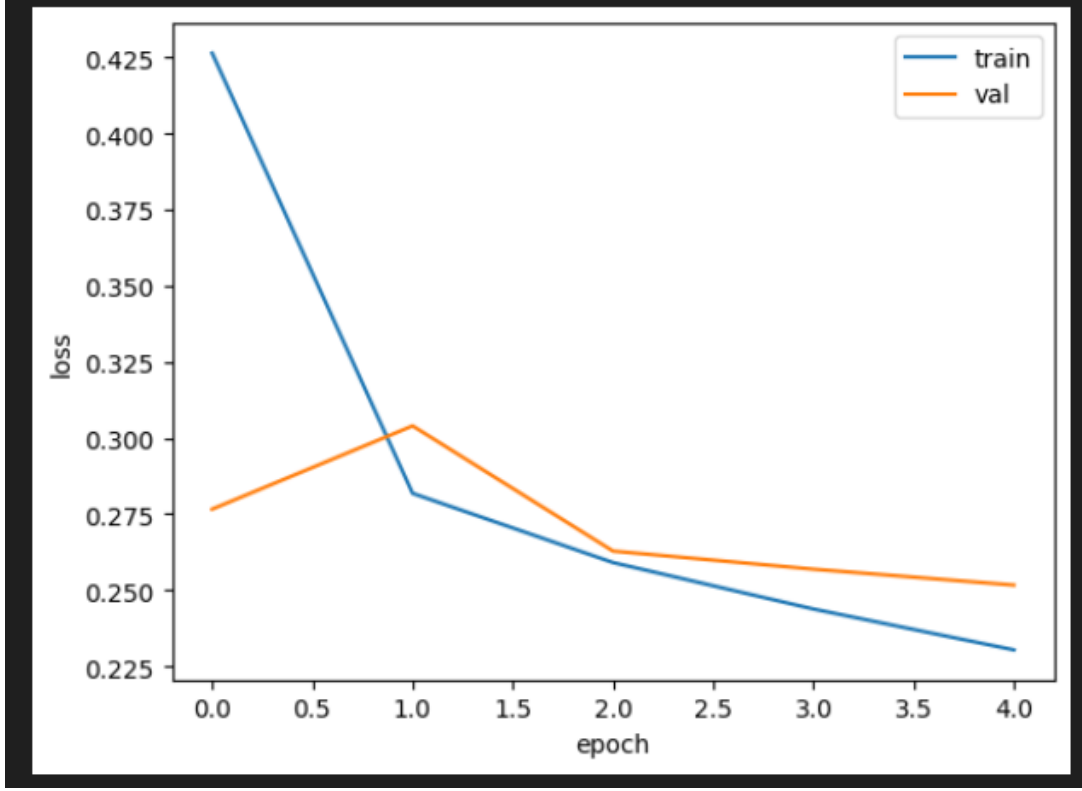
Feature를 다양한 차원으로 학습할 수 있도록 2개의 Bidirectional LSTM 레이어를 사용하는 모델. 더불어 LSTM 레이어가 더 복잡한 시간적 패턴을 포착할 수 있게 하기 위해, **첫 번째 레이어에서** 모든 hidden state들의 아웃풋 시퀀스를 전달할 수 있도록 return_sequences=True로 설정함. 두 번째 레이어는 마지막 레이어이므로, 현재 task가 seq2seq가 아니기 때문에 return_sequences=True를 설정하지 않음

#3.1 Text Classification with RNN model

```python
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train','val'])
plt.show()
```

✓ 0.2s




```python
history = model.fit(X_train, y_train, epochs=5, batch_size=128, validation_split=0.2, callbacks=[es])
```

✓ 7m 11.2s

```
Epoch 1/5
827/827 [==============================] - 85s 95ms/step - loss: 0.4264 - accuracy: 0.7995 - val_loss: 0.2766 - val_accuracy: 0.8858
Epoch 2/5
827/827 [==============================] - 81s 98ms/step - loss: 0.2818 - accuracy: 0.8933 - val_loss: 0.3040 - val_accuracy: 0.8908
Epoch 3/5
827/827 [==============================] - 91s 110ms/step - loss: 0.2591 - accuracy: 0.9030 - val_loss: 0.2628 - val_accuracy: 0.8921
Epoch 4/5
827/827 [==============================] - 85s 103ms/step - loss: 0.2438 - accuracy: 0.9099 - val_loss: 0.2569 - val_accuracy: 0.8947
Epoch 5/5
827/827 [==============================] - 88s 107ms/step - loss: 0.2304 - accuracy: 0.9154 - val_loss: 0.2517 - val_accuracy: 0.8958
```

```python
test_loss, test_acc = model.evaluate(X_test, y_test)
print("Loss:", test_loss)
print("Accuracy:", test_acc)
```

✓ 15.4s

```
1034/1034 [==============================] - 15s 15ms/step - loss: 0.2520 - accuracy: 0.8965
Loss: 0.25201135873794556
Accuracy: 0.8965141773223877
```

에폭 이 진행됨에 따라 val_loss와 train_loss 모두 감소하는 추세를 보였고 최종 validation set에서의 accuracy는 0.897이었다

#3.2 Machine Translation with Seq2Seq model

- 2. BLEU Score

```python
# BLEU score
import collections
import math

def ngram_counts(sentence, n):
  words = sentence.split()
  ngrams = [tuple(words[i:i+n]) for i in range(len(words)-n+1)]
  return collections.Counter(ngrams)
  #ngram 시퀀스마다 갯수를 센 Counter 딕셔너리 반환

#문장길이에 대한 과적합 보정
def brevity_penalty(pred, real):
  p_len = len(pred.split())
  r_len = len(real.split())
  if p_len > r_len:
    return 1
  else:
    return (p_len/r_len)
    #return math.exp(1-r_len/p_len) #min(1, pred len/real len)

def bleu_score(pred, real, weights=(0.25, 0.25, 0.25, 0.25)):
  bp = brevity_penalty(pred, real)
  ng = 1

  for i, weight in enumerate(weights, start=1):
    #i-gram 1부터 4까지
    gram_len = len(pred)-i+1 #분모(예측된 문장에서 n-gram 시퀀스의 길이)
    cnt = 0
    ngram_cnts_pred = ngram_counts(pred, i)
    ngram_cnts_real = ngram_counts(real, i)
    for key,value in ngram_cnts_pred.items():
      if key in ngram_cnts_real:
        if value > ngram_cnts_real[key]:
          cnt += ngram_cnts_real[key] #Clipping
        else:
          cnt += value
    ng *= (cnt / gram_len)**weight

  return bp * ng
```

Bleu_score()함수는 pred(모델이 예측한 문장)와 real(정답 문장)을 받아서 bleu 스코어를 계산한다. 1-gram부터 4-gram까지의 precision 을 기하 평균한 값에 brevity penalty 값을 곱한 값을 반환한다.

Ngram_counts()는 주어진 문장 시퀀스에서 주어진 n값에 따라 n-gram 시퀀스를 만들어서 그 요소들의 등장횟수를 count한 딕셔너리를 반환한다

Bravity_penalty()는 모델이 예측한 문장이 너무 짧을 때 패널티를 부여하는 역할

#3.2 Machine Translation with Seq2Seq model

- 2. BLEU Score

```python
#Test data로 성능 평가해보기(BLUE)
bleu = 0
for i, sample in enumerate(input_tensor_val):

  inp_sent = ''
  for w in sample:
    if w!=0:
      kw = inp_lang.index_word[w]
      if kw != '<start>' and kw != '<end>':
        inp_sent = inp_sent + kw +' '
  #print(inp_sent)

  pred_sent = translate(inp_sent).replace(' <end> ','')

  targ_sent = ''
  for w in target_tensor_val[i]:
    if w!=0:
      ew = targ_lang.index_word[w]
      if ew != '<start>' and ew != '<end>':
        targ_sent = targ_sent + ew + ' '
  #print(targ_sent)
  bleu += bleu_score(pred_sent, targ_sent)

  if i%100 == 0:
    print(f'{i+1}samples sampled')

print(f"총합산한 BLEU스코어: {bleu:.10f}")
```

Dataset 생성 시 테스트를 위해 분할해 두었던 Input_tensor_val과 target_tensor_val을 이용해 성능 평가하는 코드

토큰을 단어로 변환한 Inp_sent를 translate함수에 전달해 모델이 예측한 문장을 받아내고(이때 문장끝을 나타내는 <end>지우기) targ_sent와 비교를 통해 bleu score를 계산해 낸다.

테스트 샘플의 bleu 스코어를 총 합산한 값을 출력한다 (원래 평균으로 하려고 했으나 bleu 스코어들이 현저히 낮아서 합산 방식으로 변경)
결과:0

```
1samples sampled
101samples sampled
201samples sampled
301samples sampled
401samples sampled
501samples sampled
601samples sampled
701samples sampled
801samples sampled
901samples sampled
1001samples sampled
1101samples sampled
총합산한 BLEU스코어: 0.0000000000
```

#3.2 Machine Translation with Seq2Seq model

- 3. Increasing translation performance with hyperparameter tuning

```
WARNING:tensorflow:5 out of the last 5 calls to <function _BaseOptimizer._update_ste
WARNING:tensorflow:6 out of the last 6 calls to <function _BaseOptimizer._update_ste
Epoch 1 / Batch 0 / Loss 0.5508 / Time taken 6.964753150939941 sec
Epoch 1 / Batch 10 / Loss 0.3527 / Time taken 40.098310708999634 sec
Epoch 1 / Batch 20 / Loss 0.3918 / Time taken 71.29074764251709 sec
Epoch 1 / Batch 30 / Loss 0.3767 / Time taken 102.43522763252258 sec
Epoch 1 / Batch 40 / Loss 0.3616 / Time taken 132.72870254516602 sec
Epoch 1 / Batch 50 / Loss 0.3660 / Time taken 162.51793766021729 sec
Epoch 1 / Batch 60 / Loss 0.3374 / Time taken 193.02948570251465 sec
Epoch 1 / Batch 70 / Loss 0.3070 / Time taken 222.32602906227112 sec
Epoch 1 / Loss 0.3769
Time taken for 1 epoch 261.91385650634766 sec

Epoch 2 / Batch 0 / Loss 0.3140 / Time taken 3.2988977432250977 sec
Epoch 2 / Batch 10 / Loss 0.3945 / Time taken 33.160728454589844 sec
Epoch 2 / Batch 20 / Loss 0.3402 / Time taken 62.33522152900696 sec
Epoch 2 / Batch 30 / Loss 0.3320 / Time taken 91.82401037216187 sec
Epoch 2 / Batch 40 / Loss 0.2940 / Time taken 121.99018430709839 sec
Epoch 2 / Batch 50 / Loss 0.2965 / Time taken 151.4120581150055 sec
Epoch 2 / Batch 60 / Loss 0.3165 / Time taken 180.51051902770996 sec
Epoch 2 / Batch 70 / Loss 0.3145 / Time taken 209.36981749534607 sec
Epoch 2 / Loss 0.3242
Time taken for 2 epoch 215.70763111114502 sec

Epoch 3 / Batch 0 / Loss 0.3026 / Time taken 2.9805526733398438 sec
Epoch 3 / Batch 10 / Loss 0.2985 / Time taken 31.8379545211792 sec
Epoch 3 / Batch 20 / Loss 0.2655 / Time taken 61.43540096282959 sec
Epoch 3 / Batch 30 / Loss 0.3075 / Time taken 90.5802993774414 sec
Epoch 3 / Batch 40 / Loss 0.2701 / Time taken 119.67837572097778 sec
Epoch 3 / Batch 50 / Loss 0.2801 / Time taken 148.76279044151306 sec
Epoch 3 / Batch 60 / Loss 0.3014 / Time taken 177.79370856285095 sec
Epoch 3 / Batch 70 / Loss 0.2845 / Time taken 207.14012002944946 sec
Epoch 3 / Loss 0.3037
Time taken for 3 epoch 212.7887580394745 sec
```

초기 모델학습

실험 방식:
1) underfitting을 해소하기 위해 에폭 수를 10으로 늘리고
2) Adam optimizer의 learning rate을 0.01로 올려주었다
* 로컬로 돌리는 데 시간이 오래 걸려 환경을 colab으로 변경 (T4 GPU 사용)

결과: 합산 bleu스코어는 0.4347

```
1samples sampled
101samples sampled
201samples sampled
301samples sampled
401samples sampled
501samples sampled
601samples sampled
701samples sampled
801samples sampled
901samples sampled
1001samples sampled
1101samples sampled
총합산한 BLEU스코어: 0.4347332528
```

#3.2 Machine Translation with Seq2Seq model

- 3. Increasing translation performance with hyperparameter tuning

```
Epoch 7 / Batch 0 / Loss 0.2073 / Time taken 3.5878798961639404 sec
Epoch 7 / Batch 10 / Loss 0.1967 / Time taken 32.73727059364319 sec
Epoch 7 / Batch 20 / Loss 0.2139 / Time taken 62.024208545684814 sec
Epoch 7 / Batch 30 / Loss 0.1859 / Time taken 90.93750405311584 sec
Epoch 7 / Batch 40 / Loss 0.2112 / Time taken 120.82797908782959 sec
Epoch 7 / Batch 50 / Loss 0.2016 / Time taken 149.9278221130371 sec
Epoch 7 / Batch 60 / Loss 0.2204 / Time taken 179.22981023788452 sec
Epoch 7 / Batch 70 / Loss 0.1987 / Time taken 209.0883753299713 sec
Epoch 7 / Loss 0.2017
Time taken for 7 epoch 215.16784954071045 sec

Epoch 8 / Batch 0 / Loss 0.1807 / Time taken 2.717386245727539 sec
Epoch 8 / Batch 10 / Loss 0.1895 / Time taken 31.685171842575073 sec
Epoch 8 / Batch 20 / Loss 0.1885 / Time taken 61.55118203163147 sec
Epoch 8 / Batch 30 / Loss 0.1754 / Time taken 90.44241952896118 sec
Epoch 8 / Batch 40 / Loss 0.2014 / Time taken 119.43006420135498 sec
Epoch 8 / Batch 50 / Loss 0.1815 / Time taken 148.42149829864502 sec
Epoch 8 / Batch 60 / Loss 0.1924 / Time taken 179.36373925209045 sec
Epoch 8 / Batch 70 / Loss 0.2004 / Time taken 208.4376721382141 sec
Epoch 8 / Loss 0.1913
Time taken for 8 epoch 213.85535216331482 sec

Epoch 9 / Batch 0 / Loss 0.1644 / Time taken 3.568786859512329 sec
Epoch 9 / Batch 10 / Loss 0.1736 / Time taken 32.90928936004639 sec
Epoch 9 / Batch 20 / Loss 0.1860 / Time taken 61.943204402923584 sec
Epoch 9 / Batch 30 / Loss 0.1609 / Time taken 91.28462338447571 sec
Epoch 9 / Batch 40 / Loss 0.1613 / Time taken 121.29462480545044 sec
Epoch 9 / Batch 50 / Loss 0.2286 / Time taken 150.66126990318298 sec
Epoch 9 / Batch 60 / Loss 0.1778 / Time taken 180.7201840877533 sec
Epoch 9 / Batch 70 / Loss 0.1850 / Time taken 210.64005184173584 sec
Epoch 9 / Loss 0.1811
Time taken for 9 epoch 216.32001948356628 sec

Epoch 10 / Batch 0 / Loss 0.1515 / Time taken 2.7216358184814453 sec
Epoch 10 / Batch 10 / Loss 0.1650 / Time taken 32.336235761642456 sec
Epoch 10 / Batch 20 / Loss 0.1758 / Time taken 62.3652126789093 sec
Epoch 10 / Batch 30 / Loss 0.1682 / Time taken 91.5924985408783 sec
Epoch 10 / Batch 40 / Loss 0.2220 / Time taken 120.75073504447937 sec
Epoch 10 / Batch 50 / Loss 0.1899 / Time taken 150.09231114387512 sec
Epoch 10 / Batch 60 / Loss 0.1740 / Time taken 180.3912341594696 sec
Epoch 10 / Batch 70 / Loss 0.1673 / Time taken 209.5207085609436 sec
Epoch 10 / Loss 0.1732
Time taken for 10 epoch 261.91244411468506 sec
```

개선 후 학습로그