2019150445 통계학과 신백록

**1.**

우선, 11.3장의 attention 모델을 초기 모델로 하여 학습을 시켰고, train set의 acc는 0.91, validation set의 acc는 0.87정도로 약간의 overfitting이 발생하였다. start와 end를 제거한 상태로 predict를 해보면, 0.425정도의 정확도를 보인다. 출력층의 노드 수가 359개인데에 반해 sample은 50000개이기에 필연적으로 accuracy가 낮을 수 밖에 없다.

Epoch 4/5

1485/1485 [==============================] - 19s 13ms/step - loss: 0.3390 - accuracy: 0.9074 - val\_loss: 0.5161 - val\_accuracy: 0.8655

Epoch 5/5

1485/1485 [==============================] - 19s 13ms/step - loss: 0.3097 - accuracy: 0.9133 - val\_loss: 0.5051 - val\_accuracy: 0.8685

correct\_count/count

0.425890064054819

Overfitting을 해결하기 위해서, encoder와 decoder의 LSTM 층에 Dropout(0.2)를 주었고, 출력층 직전 층에 Dropout(0.3)을 주었다. validation set의 accuracy는 초기 모델과 비슷한 정도의 성능을 보였고, 위와 똑같이 start와 end를 제거한 상태로 predict를 해보았을 때, accuracy가 0.434정도로 초기 모델에 비해 약간의 성능 향상이 있었다.   
Epoch 4/5

1485/1485 [==============================] - 19s 13ms/step - loss: 0.3661 - accuracy: 0.9018 - val\_loss: 0.5181 - val\_accuracy: 0.8653

Epoch 5/5

1485/1485 [==============================] - 19s 13ms/step - loss: 0.3375 - accuracy: 0.9075 - val\_loss: 0.5123 - val\_accuracy: 0.8691

correct\_count/count

0.4336362282139133

하지만 출력층 직전 층의 dropout(0.3)만을 가지고 출력층 직전 층의 100만개에 달하는 모수를 해결할 수 없을 것이라 생각했고, Dense층의 노드 수를 반인 1000개로 줄여보았다. 출력층의 모수와 출력층 전의 Dense층에서 유의미한 모수 숫자의 감소가 있었다. (또한 dense층의 activation function을 ‘tanh’ 대신 ‘relu’ 함수를 사용해보았다.)

dense\_5 (Dense) (None, 17, 1000) 513000 concatenate[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_8 (Dropout) (None, 17, 1000) 0 dense\_5[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_6 (Dense) (None, 17, 359) 359359 dropout\_8[0][0]

Train set의 acc는 0.916이고, validation set의 acc는 0.88로 전 모델보다 성능 향상이 있는 것으로 보았고, 이 모델을 start와 end를 제외하고 predict시켜보았더니 acc가 0.46으로 초기 모델과 전의 모델에 비해 많은 성능 향상이 있었다.

Epoch 4/5

1485/1485 [==============================] - 19s 13ms/step - loss: 0.3274 - accuracy: 0.9104 - val\_loss: 0.4746 - val\_accuracy: 0.8771

Epoch 5/5

1485/1485 [==============================] - 19s 13ms/step - loss: 0.2985 - accuracy: 0.9160 - val\_loss: 0.4609 - val\_accuracy: 0.8799

0.46074780277074334

추가적으로 100차원의 Glove.6B 이전학습을 통해 성능을 향상시킬 수 있지 않을까 해서 시도해보았다. 이전학습을 통해 encoder input에 word embedding을 실행하였고, 이전학습을 시행한 layer를 non-trainable로 바꾸었다. 또한 곳곳에 dropout을 주었고, 모델을 학습시켜보았다. 이를 predict했을 때의 accuracy는 0.431으로 단순히 dropout을 주었을 때보다 성능이 떨어졌다. 이유를 잘 모르겠다..

Epoch 4/5

1485/1485 [==============================] - 18s 12ms/step - loss: 0.3571 - accuracy: 0.9027 - val\_loss: 0.5323 - val\_accuracy: 0.8624

Epoch 5/5

1485/1485 [==============================] - 18s 12ms/step - loss: 0.3290 - accuracy: 0.9082 - val\_loss: 0.5216 - val\_accuracy: 0.8663

0.4316996871741397

**2.**

Eng\_word와 fr\_word의 비율을 0.9로 바꾸고, max\_length를 측정하였다. Max\_length는 위와 동일하게 8과 17이어서 위 모델의 input을 변경할 필요가 없었고, attention 모델을 학습시켰다. 비율이 0.8일때보다 val accuracy가 낮았고, start(1254)와 end(422)를 제거하고predict해봤더니 0.3890213019514375의 acc를 보였다. 0.8 비율의 word를 사용했을 때보다 성능이 현저하게 떨어졌고, 출력층에서 추정해야할 fr\_word가 359개에서 1489개가 되었으니 성능이 떨어지는 건 당연하다.

final\_eng\_words = df[df['cum\_perc']<0.9]['word'].values

final\_fr\_words = df[df['cum\_perc']<0.9]['word'].values

Epoch 4/5

1485/1485 [==============================] - 25s 17ms/step - loss: 0.4191 - accuracy: 0.8951 - val\_loss: 0.6512 - val\_accuracy: 0.8505

Epoch 5/5

1485/1485 [==============================] - 25s 17ms/step - loss: 0.3806 - accuracy: 0.9016 - val\_loss: 0.6365 - val\_accuracy: 0.8554

Overfitting을 해결하기 위해 위 1번과 마찬가지로 출력층 직전 Dense 층의 노드를 1000개로 줄이고 모델을 학습시켰을 때에는 0.39766125428273497의 accuracy가 나왔고, 성능향상이 있었다.