기존 문제 해결 - f1 score, loss 함수, model 구조(층 구성의 원리)

<해볼 것>

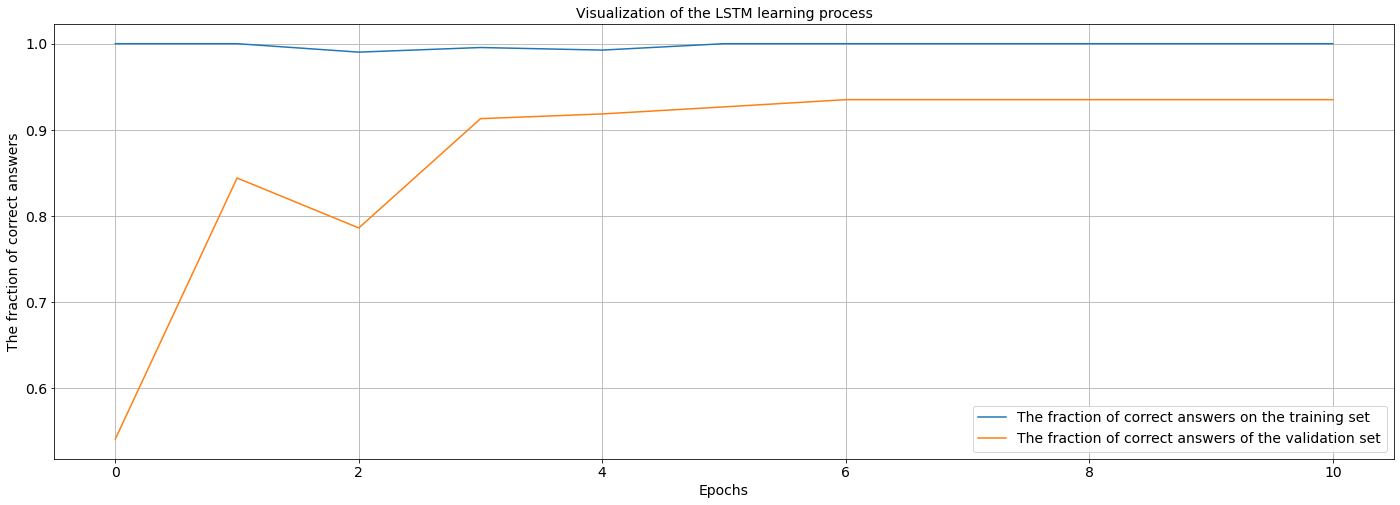
* 다른 레퍼런스들과 성능 비교 (이유 찾아보기)
* +
* transformer 사용

<완료 상황>

* 모델 스태킹 확인
* batchsize, epoch 해결
* f1\_score 해결
* loss 함수 해결
* 성능 평가 해결
* 1개 은닉층으로 LSTM 했던 것을 그대로 GRU에 적용
* unit을 늘린다 : 데이터의 차원을 늘리는 것, layer를 늘린다 : 더 복잡한 모델을 만든다(모델이 더 많은 패턴을 학습할 수 있음. lstm cell의 수는 time step(length)이 결정

<참고사이트>

<https://deep-deep-deep.tistory.com/56> - ReduceLROnPlateau정확도는 전체 데이터에서 맞게 예측한 수



P : 실행을 할 때마다 f1 score 점수가 일정하지 않음.

P : loss 함수가 가끔 튀는 경우가 발생하는데 그 이유를 파악해야함.

P :

정밀도는 예측이 참인 것 중에 실제도 참인 것의 비율 (TP/TP+FP)

재현율은 실제 참인 것 중에 예측도 참인 것의 비율 (TP/TP+FN)

조화평균 논리에 따라 정밀도와 재현율이 비슷할 수록 F1 score도 높아진다.

f1 score는 정밀도, 재현율 둘 중 하나가 0에 가깝게 낮을때 accuracy와 달리 지표에 잘 반영

=> FP(실제 거짓, 예측 참)값과 FN(실제 참, 예측 거짓)값이 비슷하면서 클 때, f1 score는 높아진다.

=> 잘못된 판단들이 비슷하면서 낮을 때, 높아진다.

정확도는 전체 데이터에서 맞게 예측한 수

f1 score 처음 시도에 너무 높은 수치 => 과적합 의심하여 batch\_size 감소

lstm 층(3)을 제거했을 때와 달리 있을 때 더 좋은 손실함수 그래프와 학습과정 그래프를 보여줌. 이때, 성능 0.90까지 상승. 그 이유는~

등장 빈도가 1번 이하인 희귀 단어의 수: 5001

단어 집합(vocabulary)에서 희귀 단어의 비율: 55.54198134162595

전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 6.18438137636802

등장 빈도가 2번 이하인 희귀 단어의 수: 6297

단어 집합(vocabulary)에서 희귀 단어의 비율: 69.93558418480676

전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 9.38972361342979

등장 빈도가 3번 이하인 희귀 단어의 수: 6899

단어 집합(vocabulary)에서 희귀 단어의 비율: 76.62150155486451

전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 11.623075496197366

등장 빈도가 4번 이하인 희귀 단어의 수: 7239

단어 집합(vocabulary)에서 희귀 단어의 비율: 80.3976010661928

전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 13.304890867495208

등장 빈도가 5번 이하인 희귀 단어의 수: 7474

단어 집합(vocabulary)에서 희귀 단어의 비율: 83.00755219902265

전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 14.757929883138566

등장 빈도가 6번 이하인 희귀 단어의 수: 7641

단어 집합(vocabulary)에서 희귀 단어의 비율: 84.86228342958685

전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 15.997032090521241

등장 빈도가 7번 이하인 희귀 단어의 수: 7805

단어 집합(vocabulary)에서 희귀 단어의 비율: 86.6836961350511

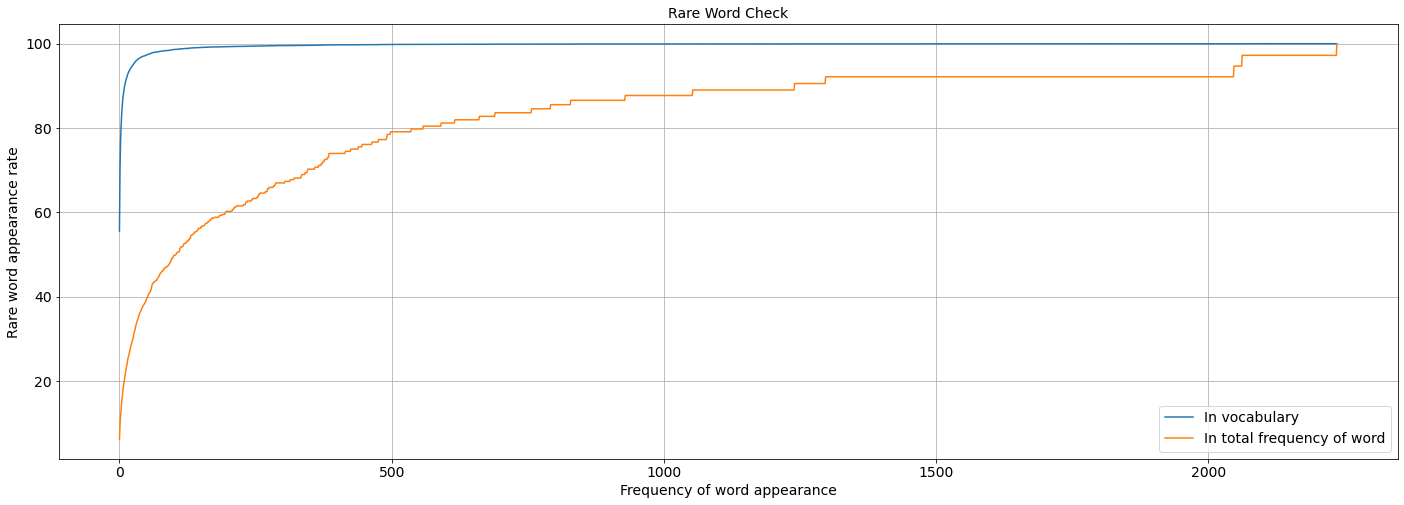
전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 17.416682124528535

.

.

.

그래프화 해본 결과



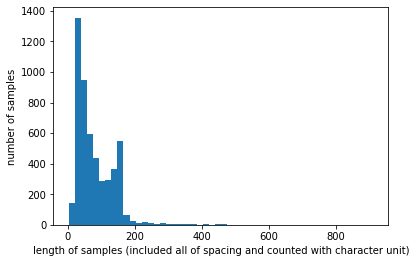
파란 선 : 단어 집합에서 희귀 단어의 비율

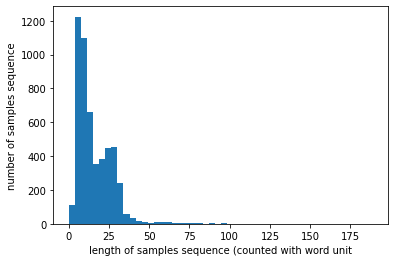
주황 선 : 전체 등장 빈도에서 희귀 단어 등장 빈도 비율

**정제 및 정규화 진행**

re모듈을 활용하여 진행

**패딩을 위한 최대 길이 확인**





=> 띄어쓰기와 문자 단위로 구한 길이와 단어 단위로 구한 길이의 차이가 존재.

=> 단어 단위의 최대 길이 : 189, 평균 길이 : 15.68

Q : 최대 길이 패딩이 아닐 경우 성능 변화는?

**모델 층 변화**

model = Sequential()

model.add(Embedding(NUM\_WORDS, 64, input\_length=MAX\_TEXT\_LEN))

model.add(LSTM(3, return\_sequences=True))

model.add(LSTM(5, return\_sequences=True))

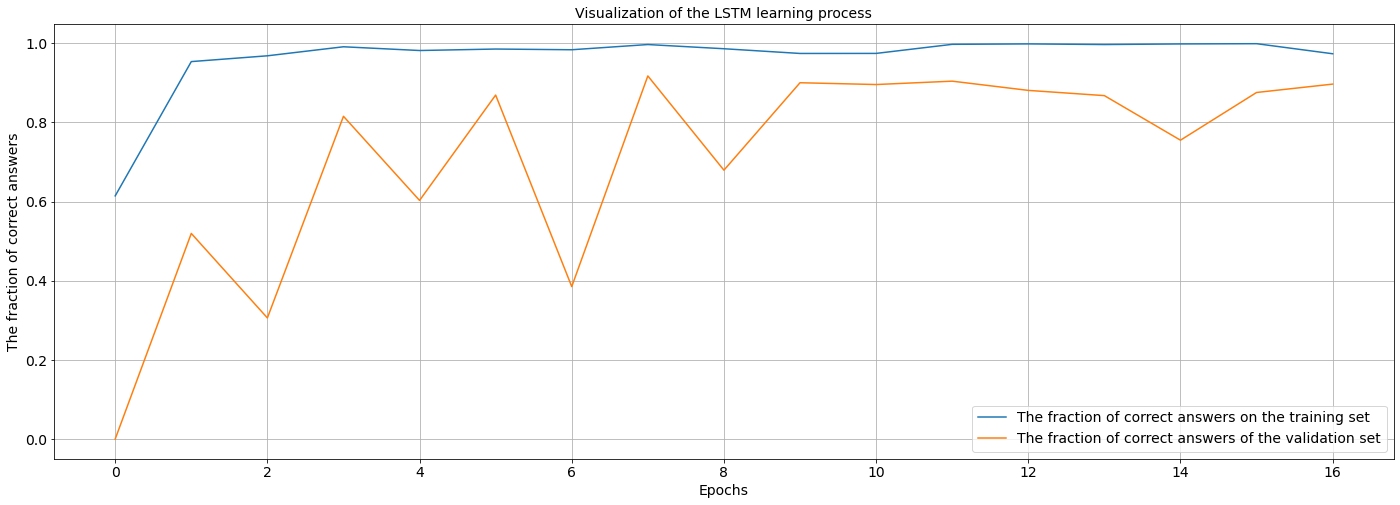
model.add(LSTM(7, return\_sequences=True))

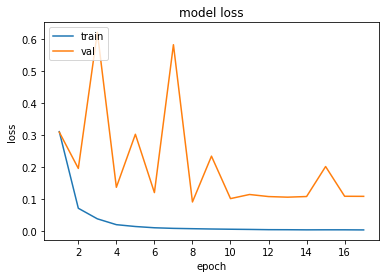
model.add(BatchNormalization())

model.add(LSTM(12))

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

일 때,





심각함… 이유가 뭘까???

lstm 코드를 이번에는 하나만 넣고 dropout 코드도 추가했다.

이전 loss의 간격이 벌어지는 것이 과적합을 의미할 수 있어서 dropout을 적용했다.

model = Sequential()

model.add(Embedding(NUM\_WORDS, 128, input\_length=MAX\_TEXT\_LEN))

model.add(LSTM(128, dropout=0.2, recurrent\_dropout= 0.2, return\_sequences=True))

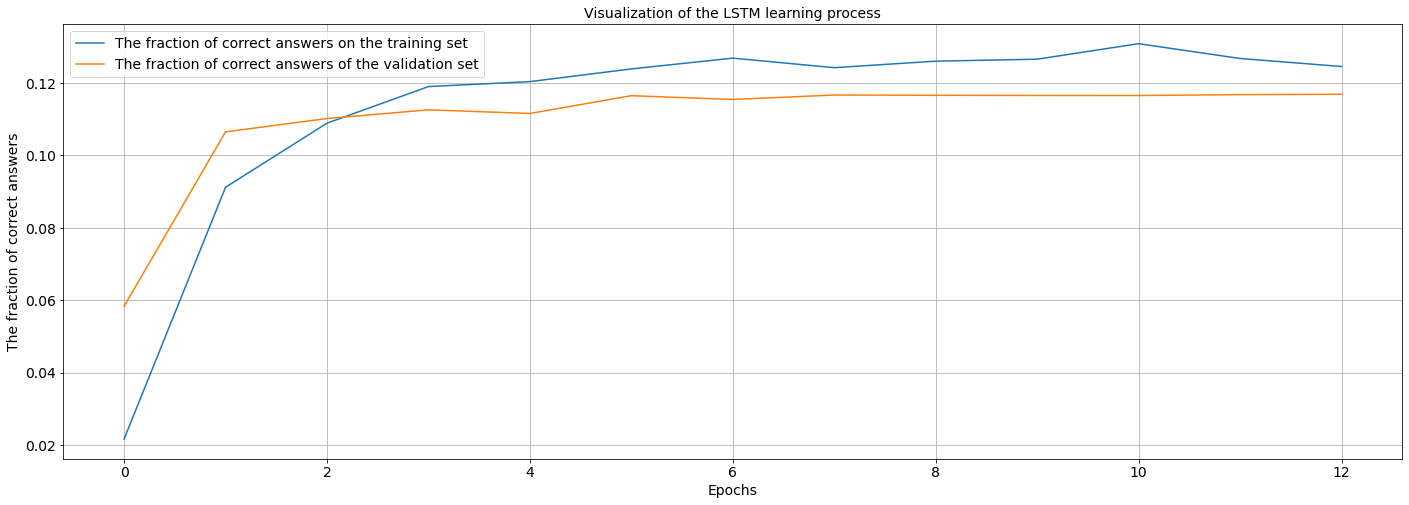
"""model.add(LSTM(5, return\_sequences=True))

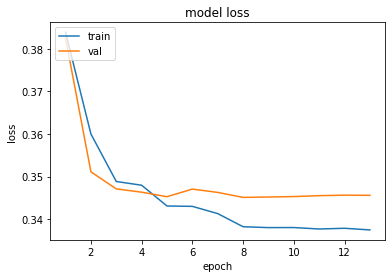
model.add(LSTM(7, return\_sequences=True))

model.add(BatchNormalization())

model.add(LSTM(12))"""

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





문제가 predict하면 X\_test크기가 그래도 반영된다. sequence 전체가 아니라 단어 하나하나에 적용이 된다. shpae이 (1032, 189,1) 층 구조를 하나로 했을 때의 문제인가?

model = Sequential()

model.add(Embedding(NUM\_WORDS, 128, input\_length=MAX\_TEXT\_LEN))

model.add(LSTM(128, dropout=0.2, recurrent\_dropout= 0.2, return\_sequences=True))

"""model.add(LSTM(5, return\_sequences=True))

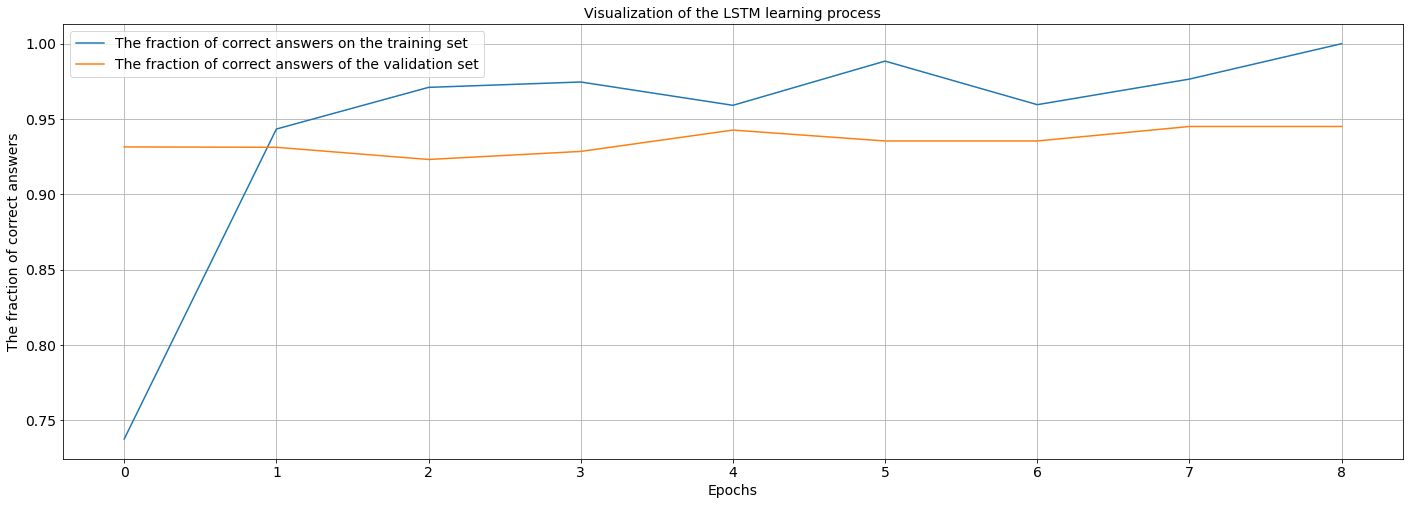
model.add(LSTM(7, return\_sequences=True))

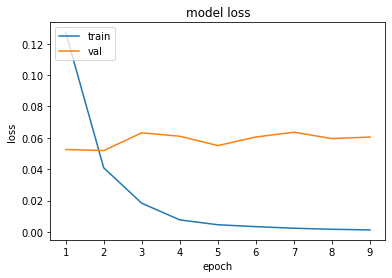
model.add(BatchNormalization())"""

model.add(LSTM(12))

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

lstm(12)를 추가했더니 학습속도가 많이 늦어짐.





망함. 완전 과적합 (유닛 조합 때문에 망한것인가??)(어쩌면 층에 따른 유닛 조합이 곱으로 돼서 학습시간이 늦어졌을 수도)

그래도 pred shape이 (1032,1)로 바뀌게 됨. 층의 개수에 따라서 달라지는게 맞는 듯함.

array([[7.5708842e-04],

[1.5432643e-03],

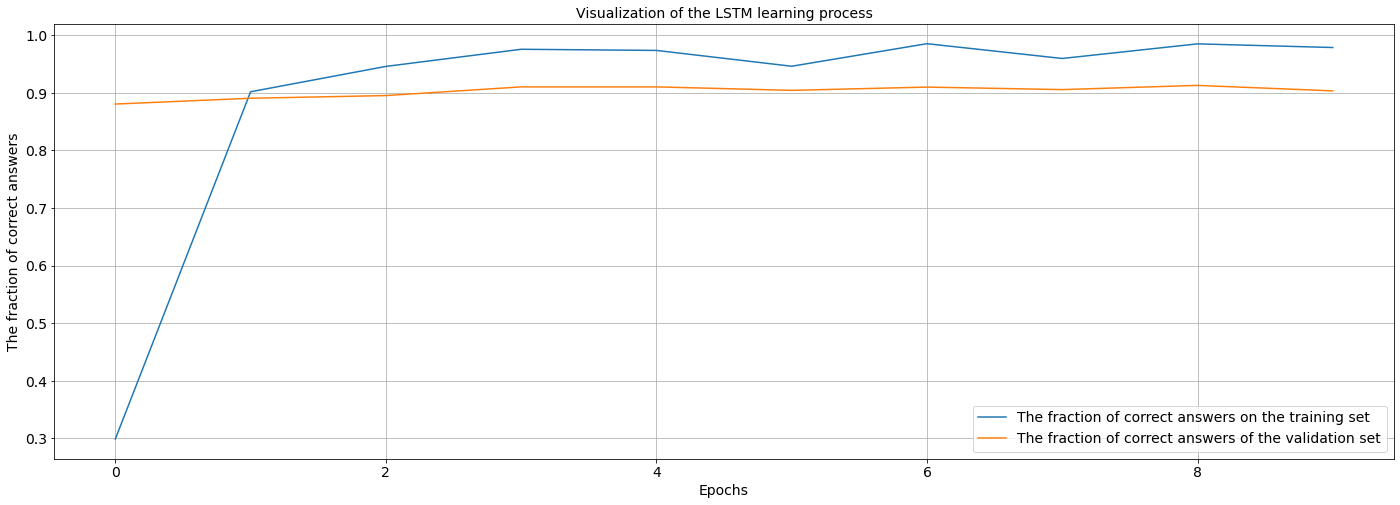
[1.8080625e-03],

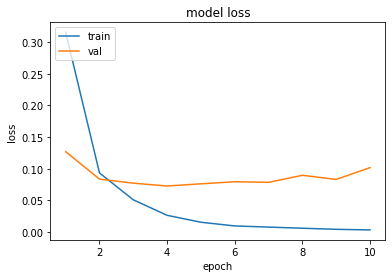
...,

[9.9830663e-01],

[2.2604095e-03],

[3.3937970e-03]], dtype=float32)





model = Sequential()

model.add(Embedding(vocab\_size, 128, input\_length=MAX\_TEXT\_LEN))

model.add(LSTM(5, dropout=0.2, recurrent\_dropout= 0.2, return\_sequences=True))

model.add(LSTM(8, return\_sequences=True))

"""model.add(LSTM(7, return\_sequences=True))

model.add(BatchNormalization())"""

model.add(LSTM(12))

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

위와 같이 val\_loss가 많이 문제가 있어보인다…

그럼 다시 근본으로 돌아가서 lstm(3), lstm(5)로 dropout을 추가해서 돌려봄.

model = Sequential()

model.add(Embedding(vocab\_size, 128, input\_length=MAX\_TEXT\_LEN))

model.add(LSTM(3, dropout=0.2, recurrent\_dropout= 0.2, return\_sequences=True))

model.add(LSTM(5, return\_sequences=True))

"""model.add(LSTM(7, return\_sequences=True))

model.add(BatchNormalization())"""

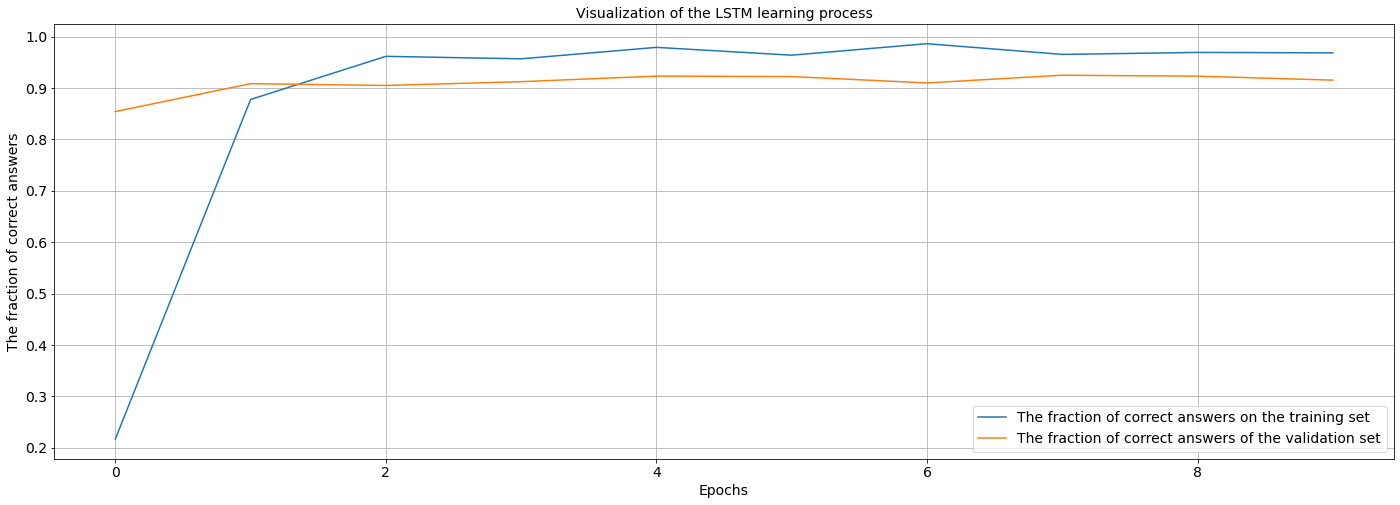
model.add(LSTM(12))

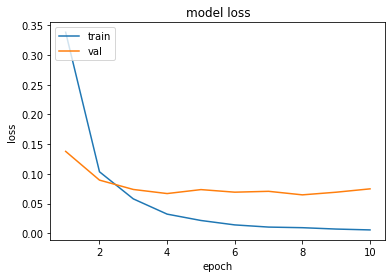
model.add(Dense(1, activation='sigmoid'))

=> vocab\_size를 9004로 실제 사이즈에 맞췄지만, fit이 돌아가지 않음.

=> 10000으로 다시 바꿔서 돌렸을 때, 잘 돌아감. 그래프도 좋아짐.

=> 처음부터 val\_loss값이 낮은데 괜찮은 것인가…

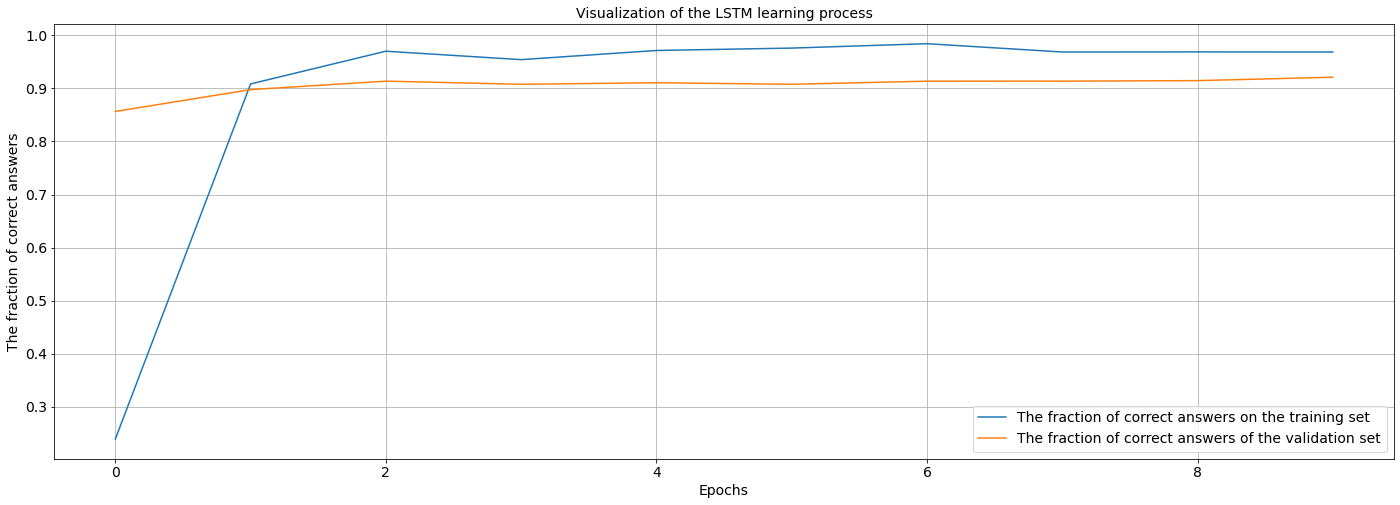


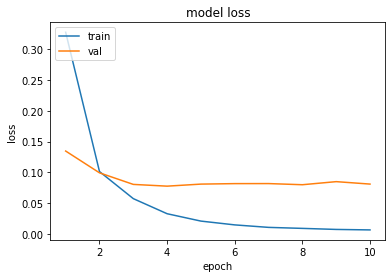


lstm에서 dropout값을 제거해서 진행해봤다. recurrent\_dropout은 유지 시켰음.

dropout : 입력 값에 대한 드롭아웃, recurrent\_dropout : 순환 값에 대한 드롭아웃

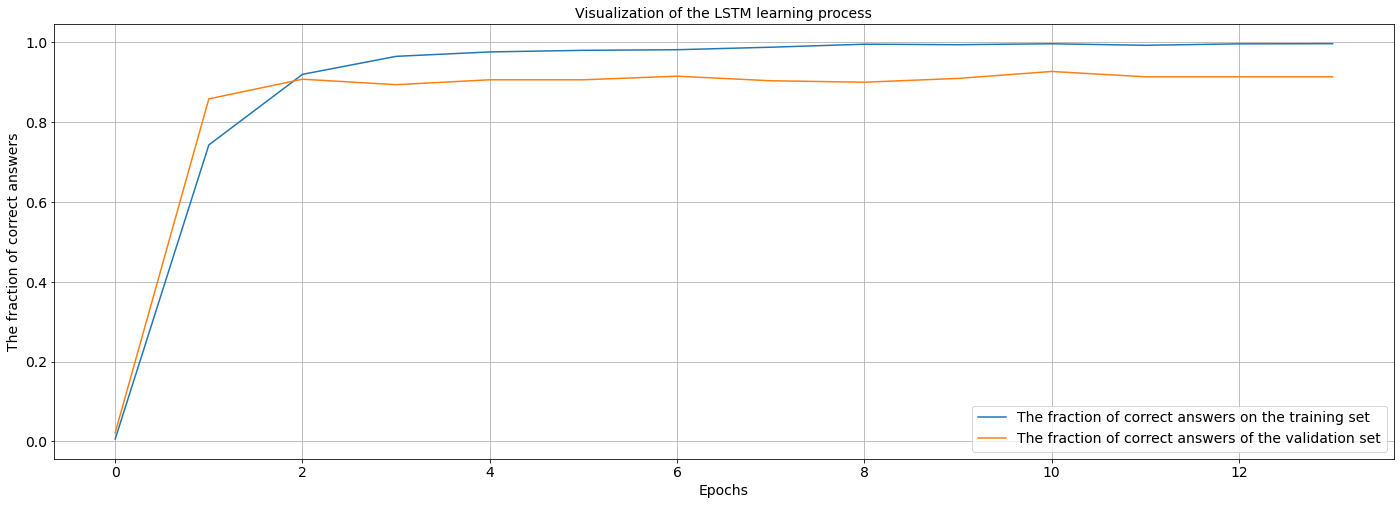
=> 지속적인 문제인 val의 loss값이 떨어지는 한계가 존재함.

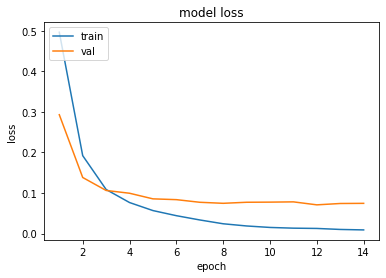




반대로도 진행

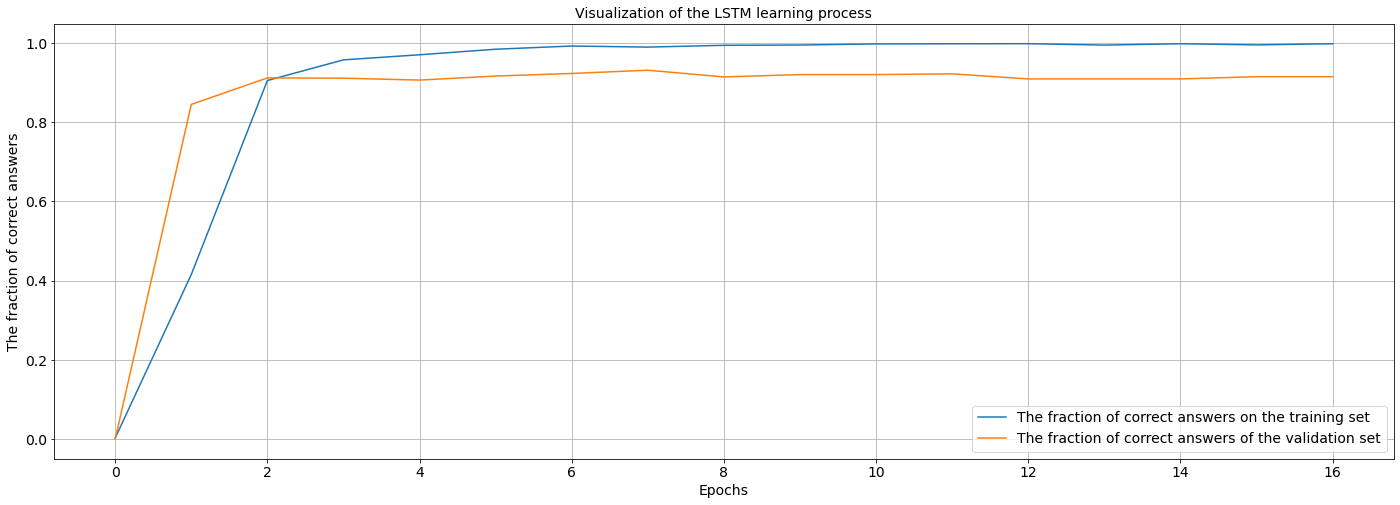
recurrent를 제거, batch\_size, epoch도 두배로 늘림. (1epoch당 과정이 많아도 과적합에 영향을 준다고 생각함)

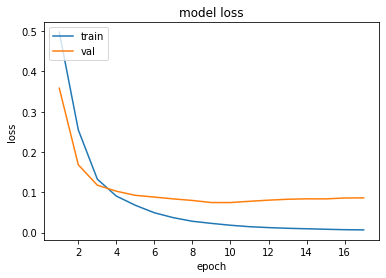




괜찮은 곡선이 만들어짐.

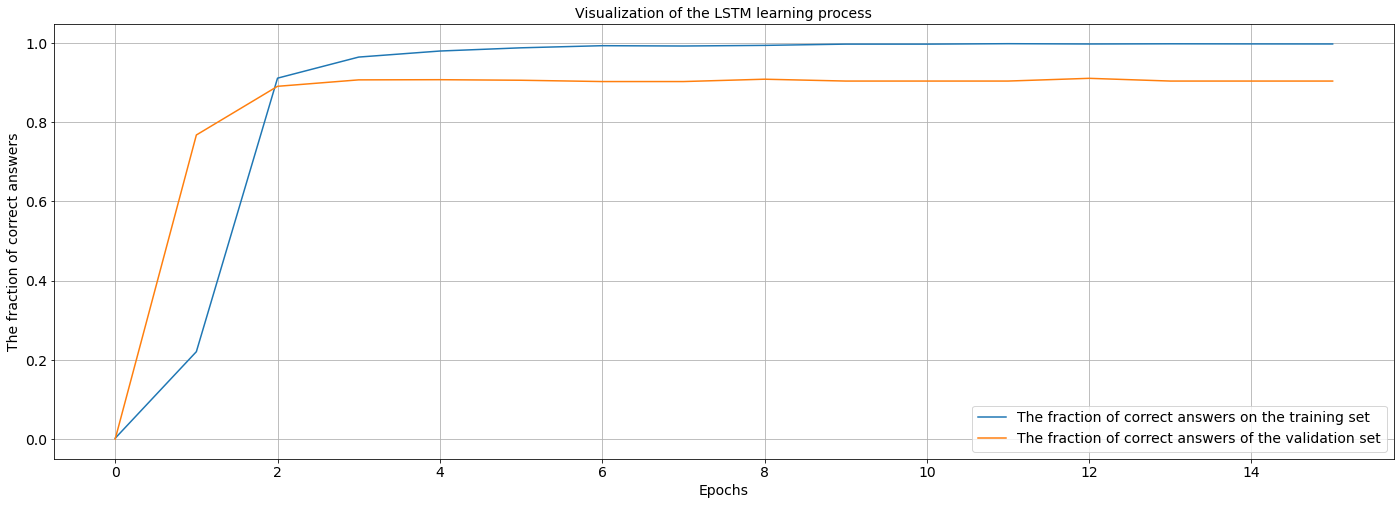
batch와 epoch 때문인지 recurrent\_dropout을 다시 추가해봄

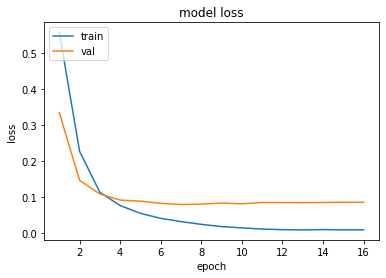




=> 큰 차이를 찾아볼 수는 없었음.

다음은 콜백함수의 하이퍼 파라미터 이해를 높이고 monitor 값을 이해봤다.

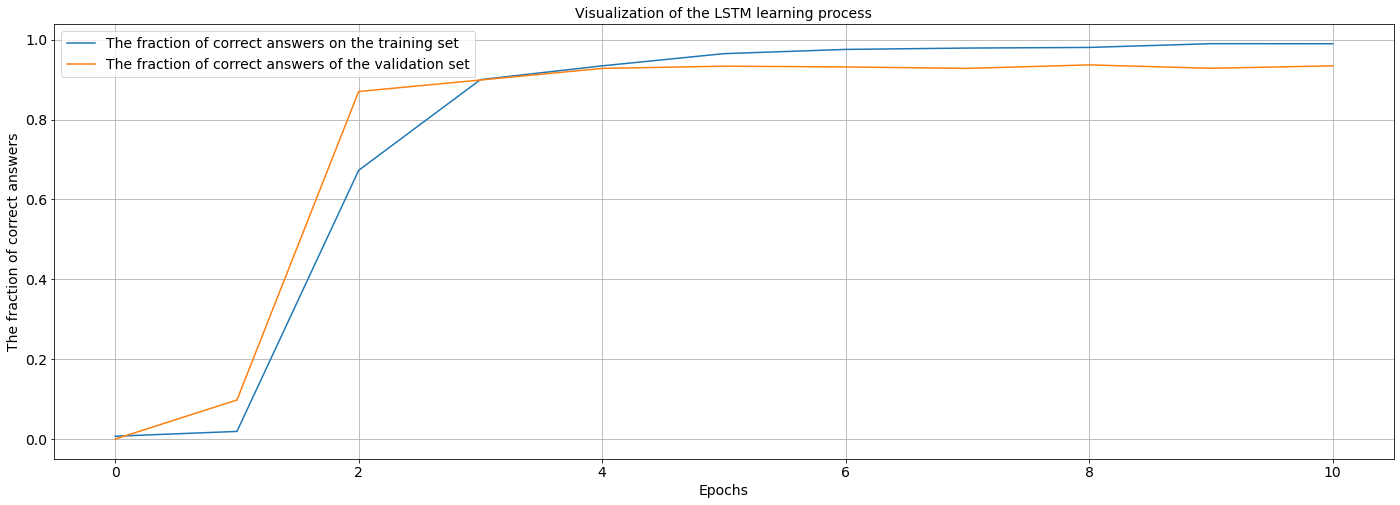


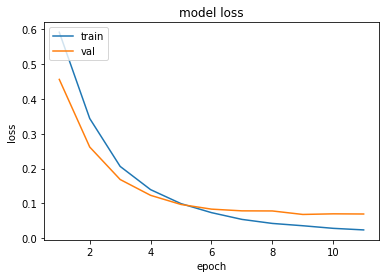


=> 큰 차이가 없었음.

dropout층을 추가하여 학습을 해볼까

val\_loss를 낮출 수 있다는 batch\_size를 더 증가시켜 봤다.



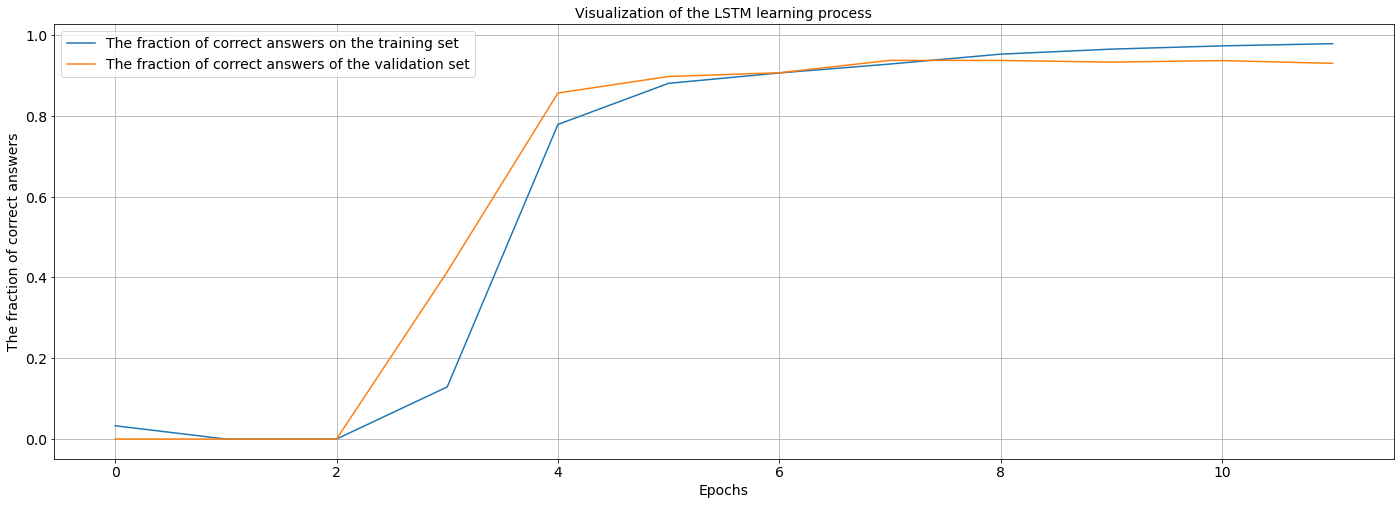


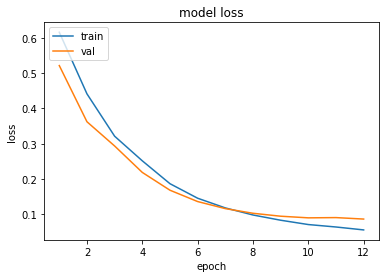
WOW~~~

테스트 f1\_score : 0.8934426229508198

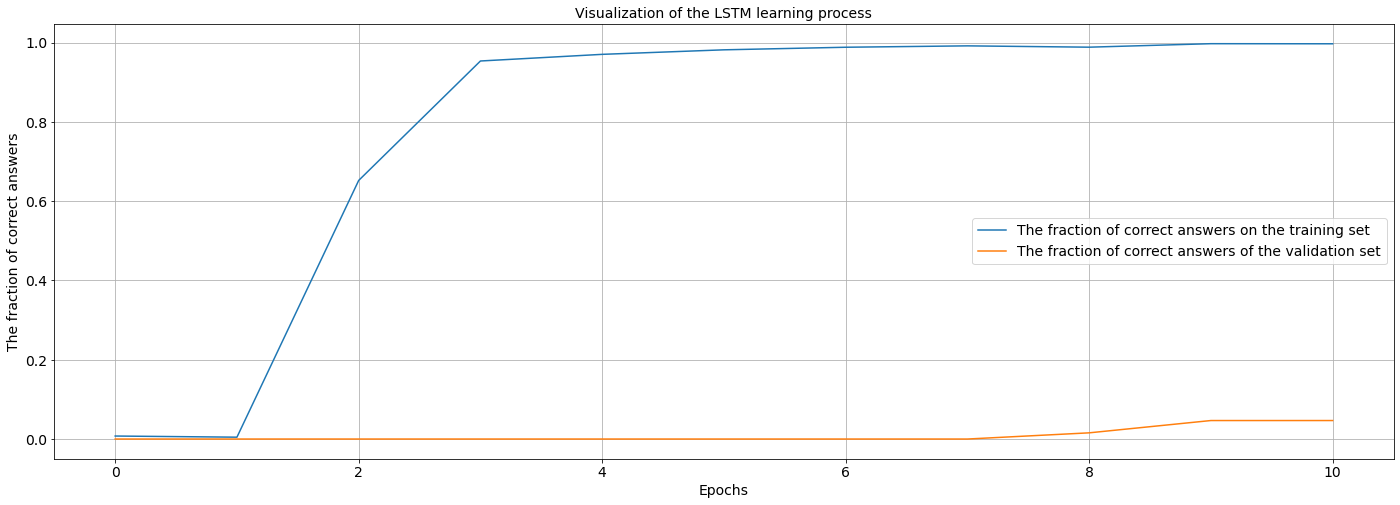
테스트 데이터 정확도 : 0.8713782429695129

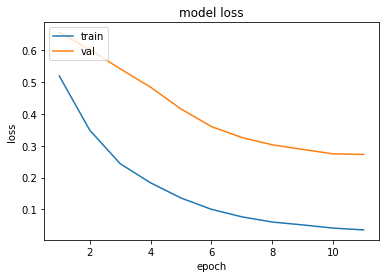
threshold에 따라 f1\_score의 차이가 존재하는데 0.6에서 가장 높은 수치를 보여줌.





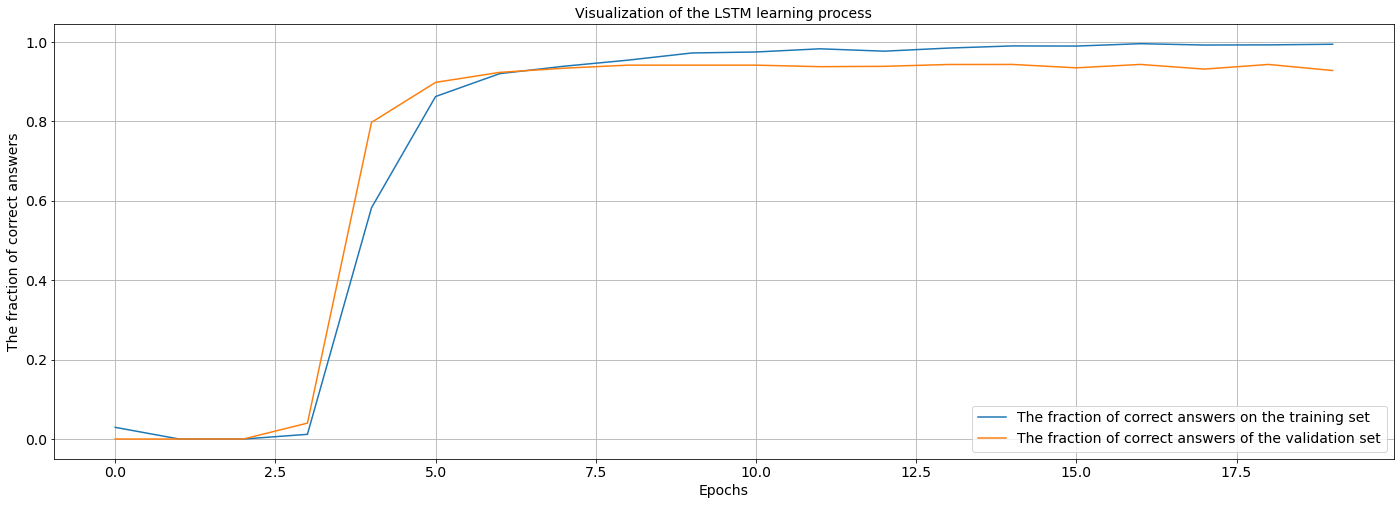
배치 사이즈를 또다시 2배해서 256으로 해봤다.

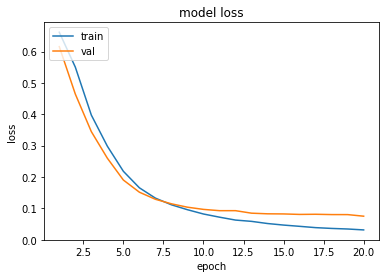




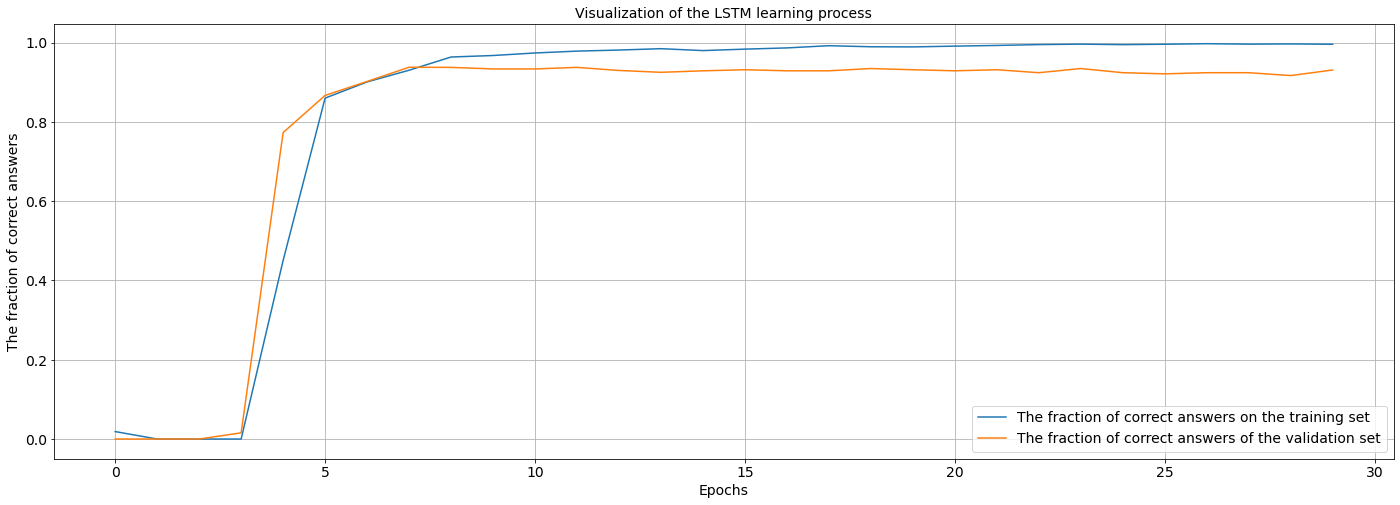
배치정규화 층을 쌓으니 이상해졌다…

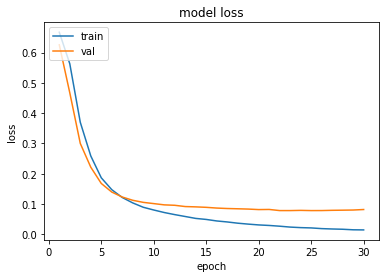
그래서 배치정규화 층은 제거하고 학습을 더할 수 있을 것 같은데 earlystop이 돼서 끝까지 학습을 돌려봤다.



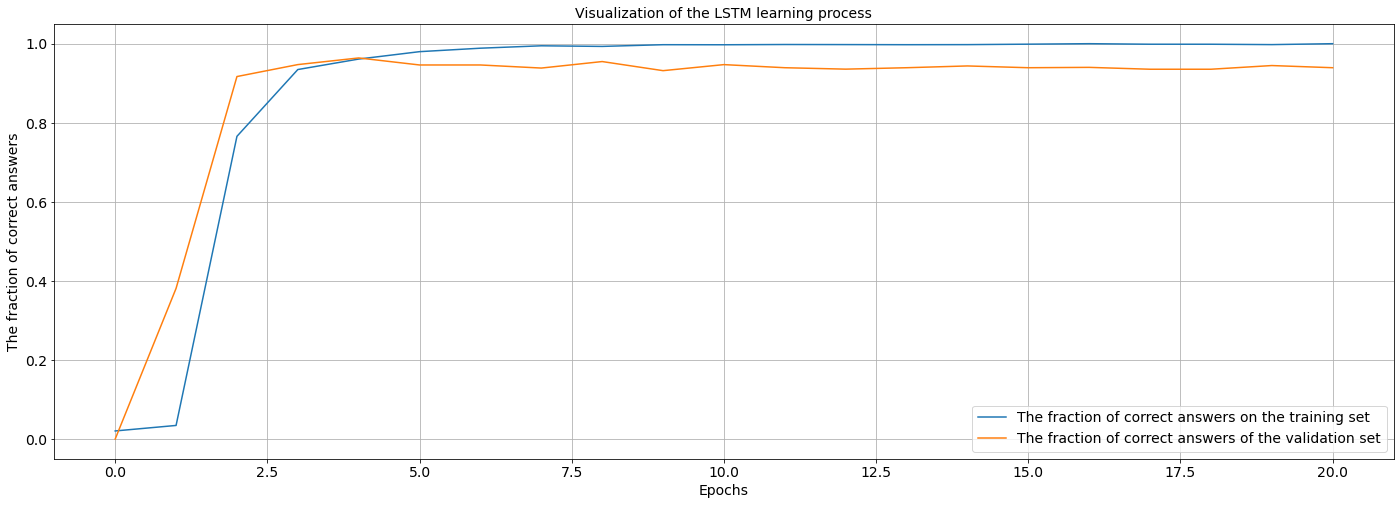


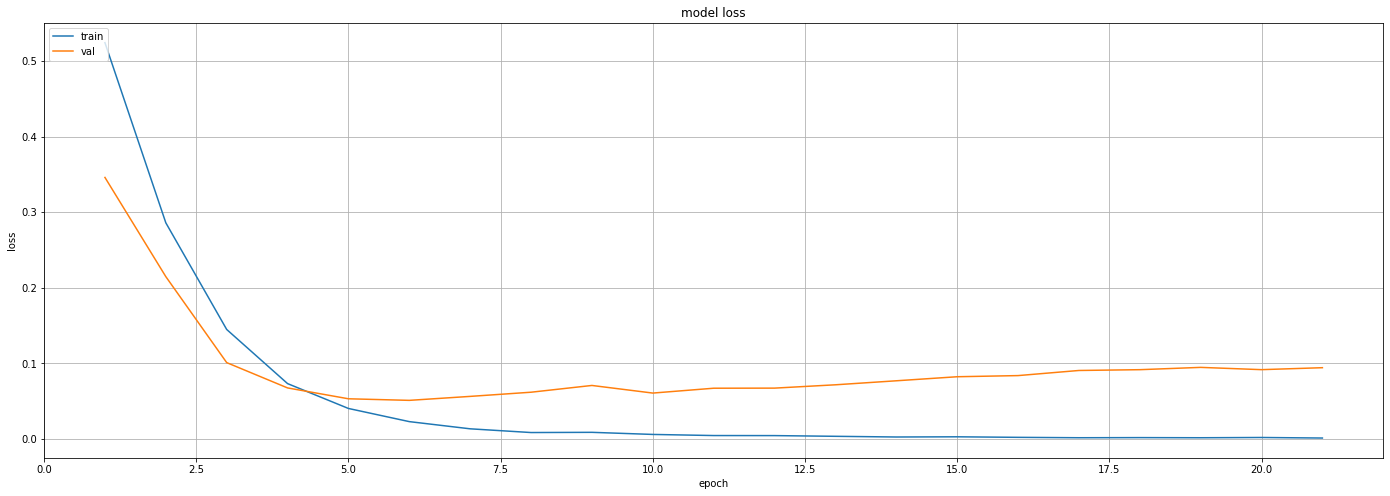
그래도 좀 더 가능할 것 같아 epoch를 늘려서 다시 돌려봤다.





epoch를 30으로 늘렸는데 20일 때가 지속되는 것으로 보임.





lstm의 unit 사이즈를 변경해보았다.

model = Sequential()

model.add(Embedding(vocab\_size, 128, input\_length=MAX\_TEXT\_LEN))

"""model.add(LSTM(3, dropout= 0.2, recurrent\_dropout=0.2, return\_sequences=True))

model.add(LSTM(5, return\_sequences=True))"""

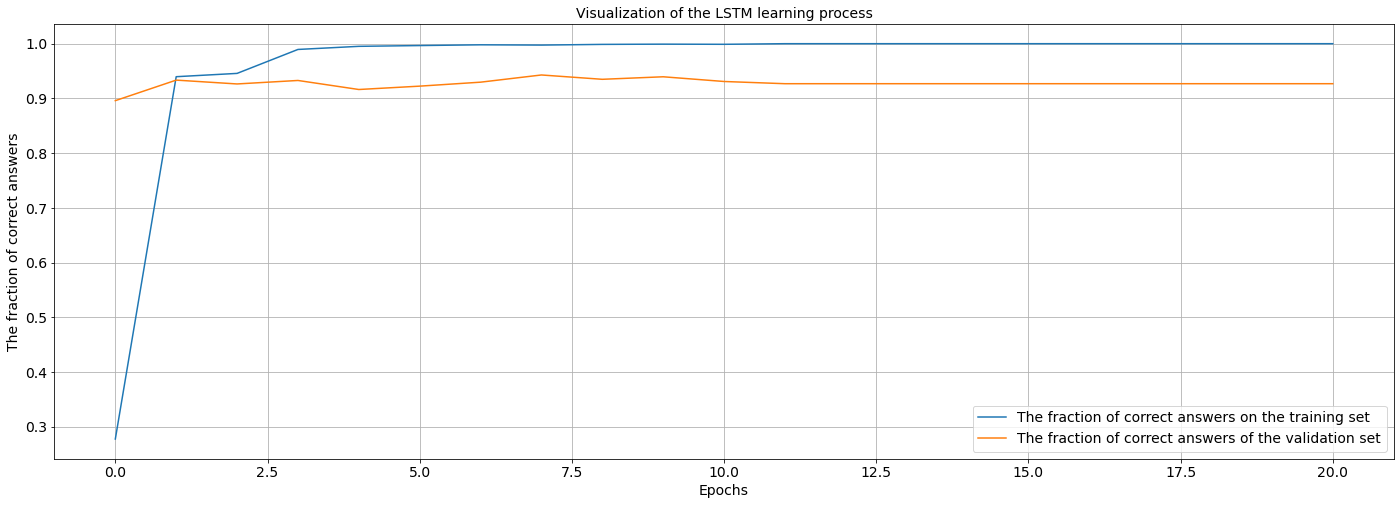
"""model.add(LSTM(7, return\_sequences=True))"""

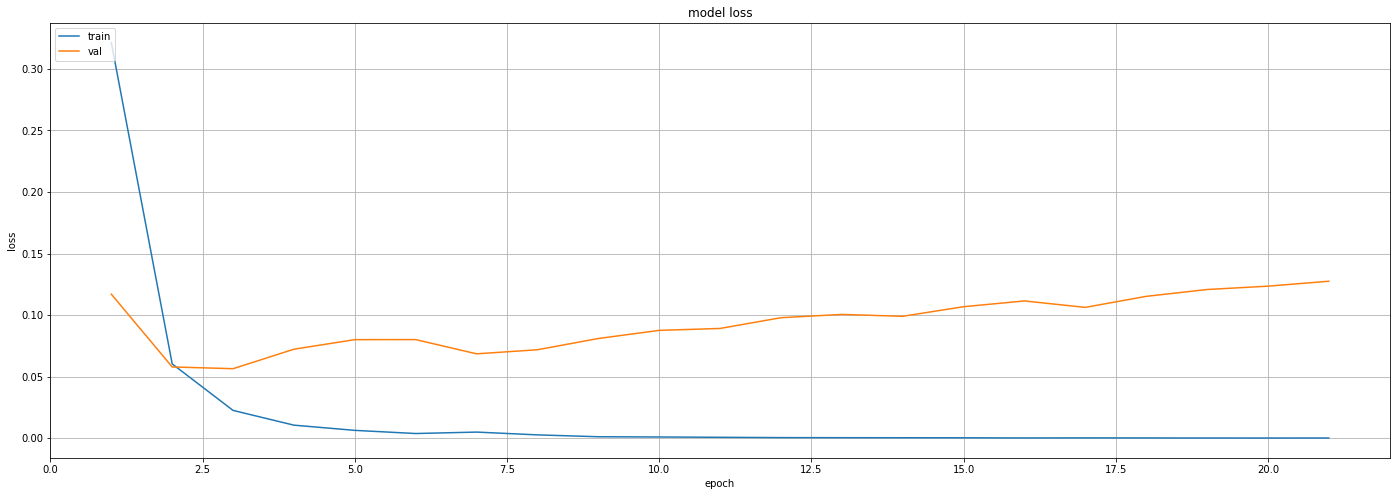
"""model.add(BatchNormalization())"""

model.add(LSTM(32, dropout=0.3, return\_sequences = True))

model.add(LSTM(64, dropout=0.3))

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





model = Sequential()

model.add(Embedding(vocab\_size, 128, input\_length=MAX\_TEXT\_LEN))

"""model.add(LSTM(3, dropout= 0.2, recurrent\_dropout=0.2, return\_sequences=True))

model.add(LSTM(5, return\_sequences=True))"""

"""model.add(LSTM(7, return\_sequences=True))"""

"""model.add(BatchNormalization())"""

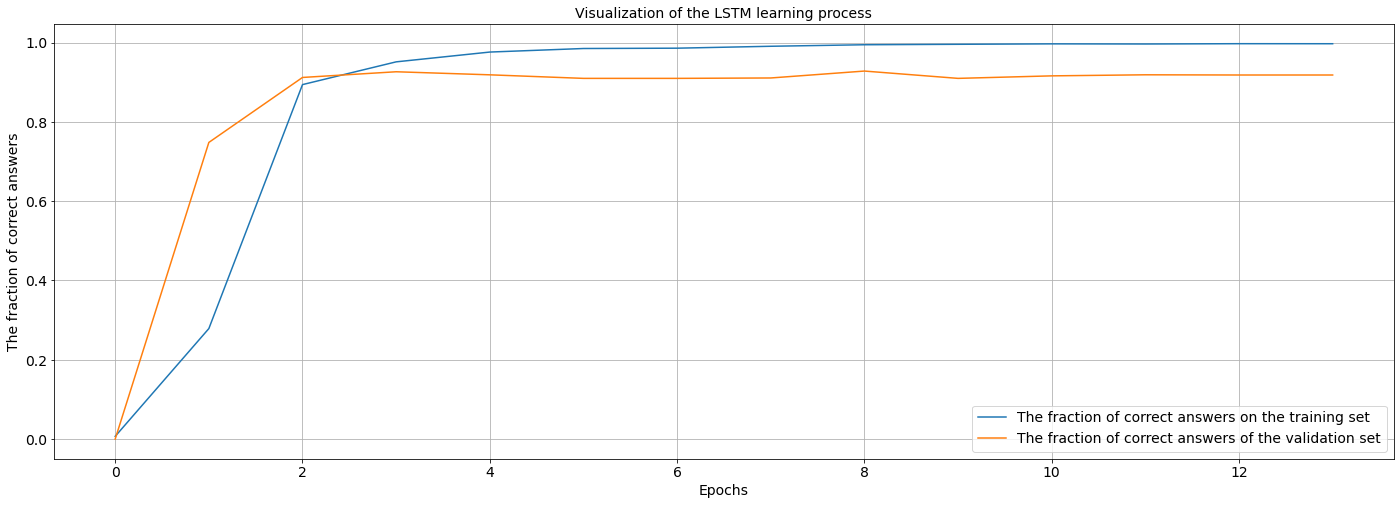
model.add(LSTM(32, dropout=0.3, return\_sequences = True))

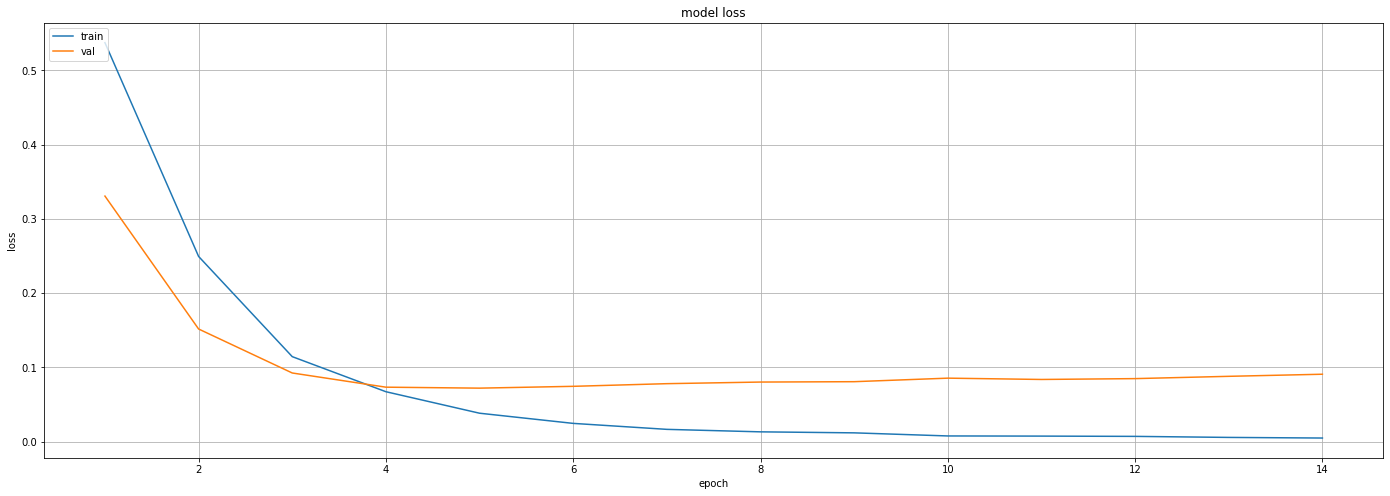
model.add(LSTM(64, dropout=0.3))

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

배치를 128로 줄여서 진행했을 때, 256일 때와 큰 차이가 없었다. 어느 지점에서 차이가 두드러지는지 확인해봐징~

배치size가 낮을수록 그래프가 그려지는게 불안전한 형태로 그려짐





model = Sequential()

model.add(Embedding(vocab\_size, 128, input\_length=MAX\_TEXT\_LEN))

"""model.add(LSTM(3, dropout= 0.2, recurrent\_dropout=0.2, return\_sequences=True))"""

model.add(LSTM(5, return\_sequences=True))

"""model.add(BatchNormalization())"""

model.add(LSTM(16, dropout=0.3, return\_sequences = True))

model.add(LSTM(32, dropout=0.3))

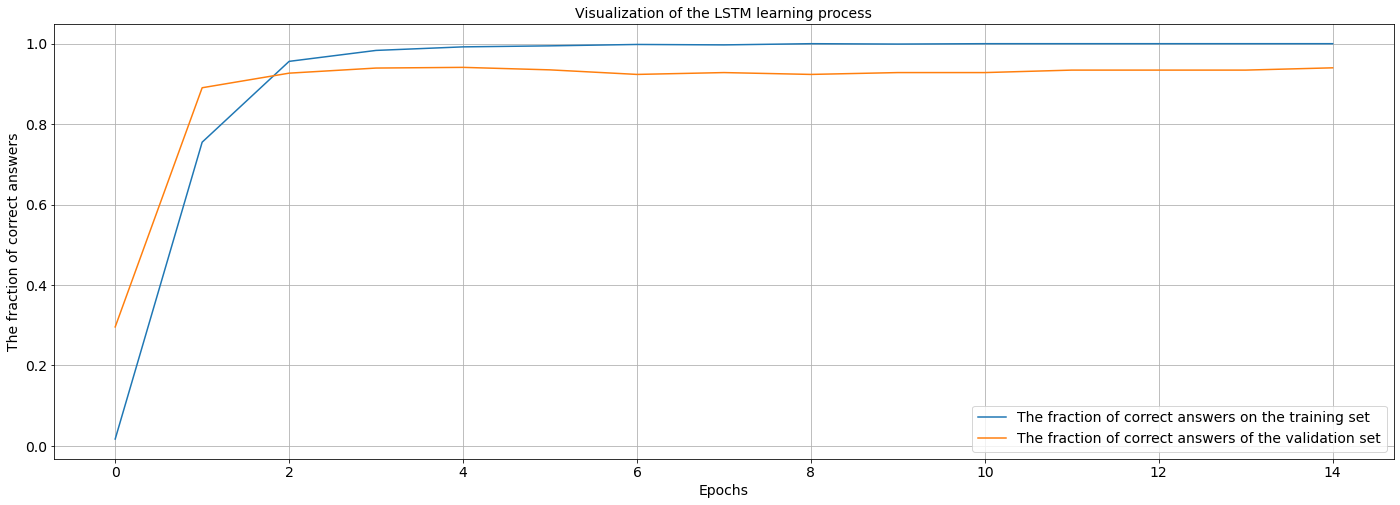
model.add(Dense(1, activation='sigmoid'))

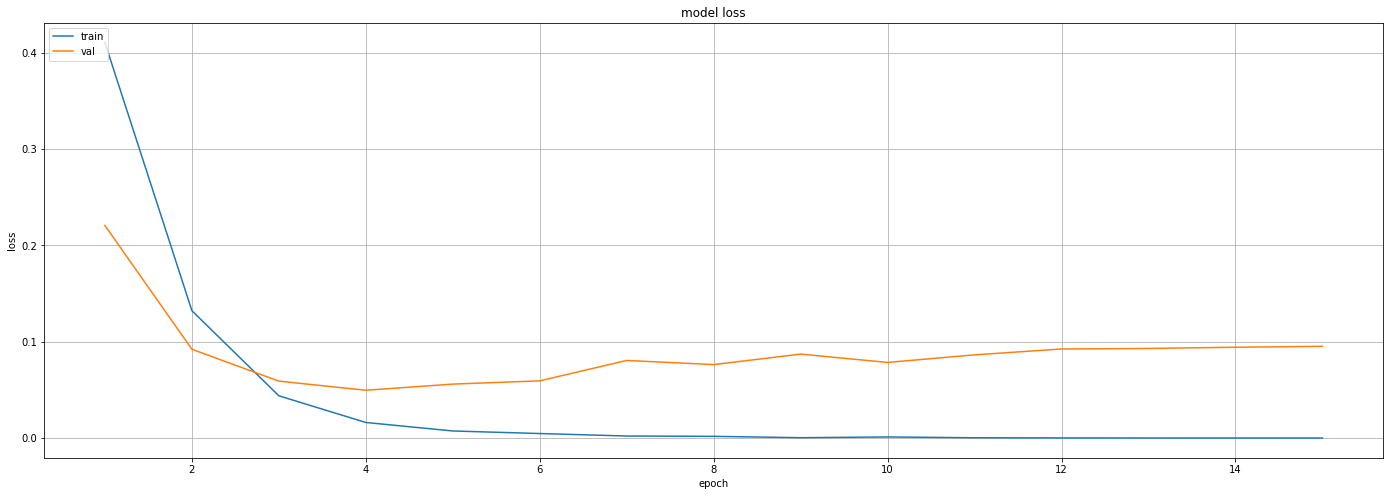
층을 추가했는데 큰 차이가 없었다.

왜왜왜왜왜왜왜왜왜왜왜왜왜왜왜왜

LSTM 한 층으로는 안되는 것인가…

오오오 나는 멍충이었다. 층 쌓을 때 사용하는 return\_sequence=true를 계속 사용하니 되지 않았던 것이다. 그걸 고치고 성능을 돌렸을 때,





f1\_score X\_test 평가 : 0.9306122448979591

evaluate : 0.9140452146530151

model = Sequential()

model.add(Embedding(vocab\_size, 128, input\_length=MAX\_TEXT\_LEN))

"""model.add(LSTM(3, dropout= 0.2, recurrent\_dropout=0.2, return\_sequences=True))"""

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

"""model.add(BatchNormalization())"""

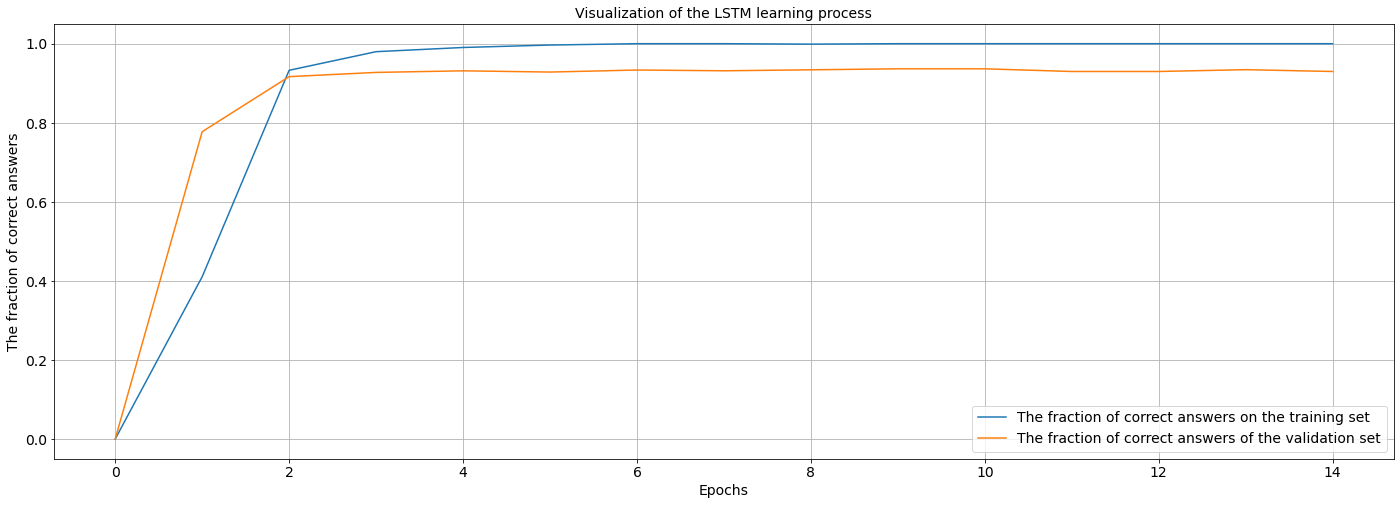
model.add(LSTM(128, dropout=0.3, return\_sequences = True))

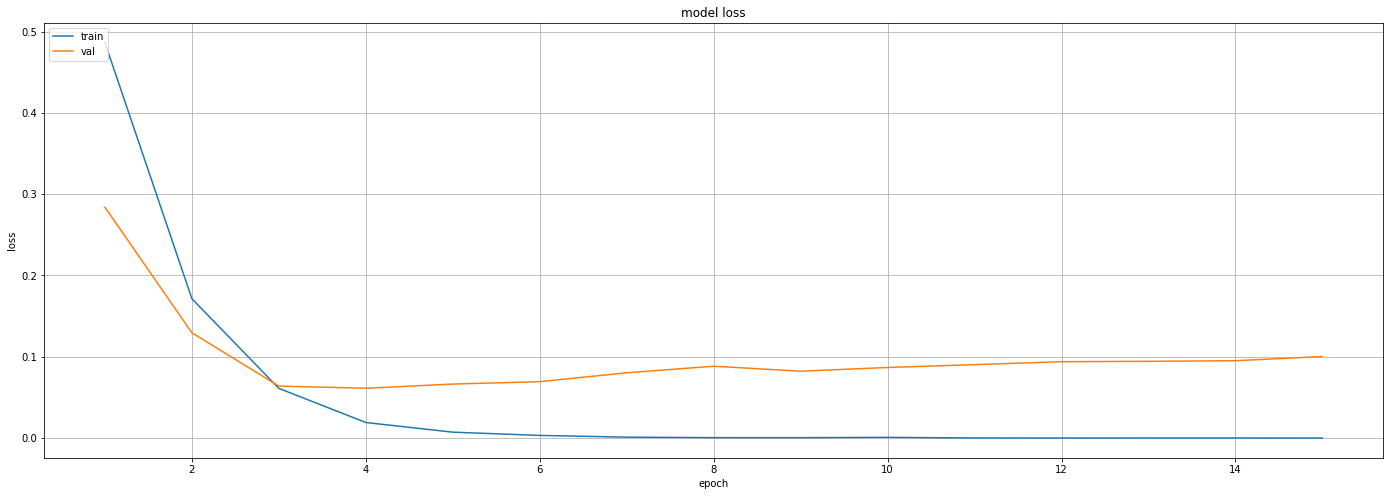
model.add(LSTM(64, dropout=0.3))

model.add(Dense(64,activation='relu'))

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

렐루 함수 층을 추가해봤다.





model = Sequential()

model.add(Embedding(vocab\_size, 128, input\_length=MAX\_TEXT\_LEN))

#model.add(LSTM(128, dropout=0.3, return\_sequences = True)) # 근데 만약에 한번더 모델의 출력을 다시 입력으로 넣게 된다면 무슨 이득이 있지?

model.add(GRU(64, dropout=0.3, recurrent\_dropout=0.3))

model.add(Dense(64,activation='relu'))

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

GRU로 바꿨는데 lstm의 간단화 버전이라서 그런지 그래프의 큰 변화가 발생하지 않았다.

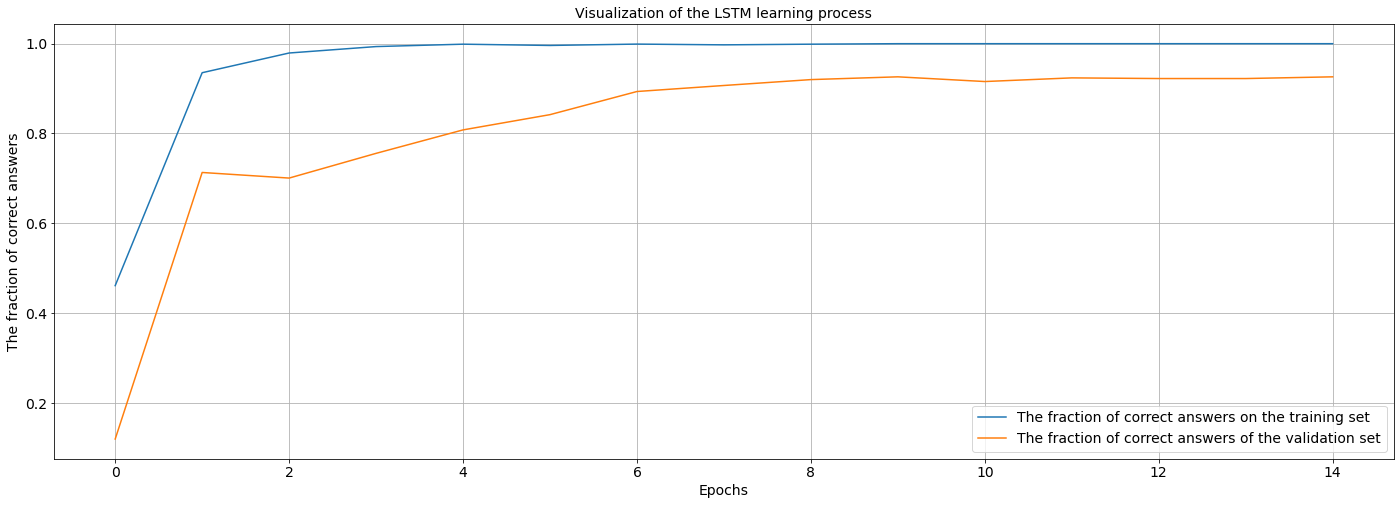
LSTM의 구조적 특징

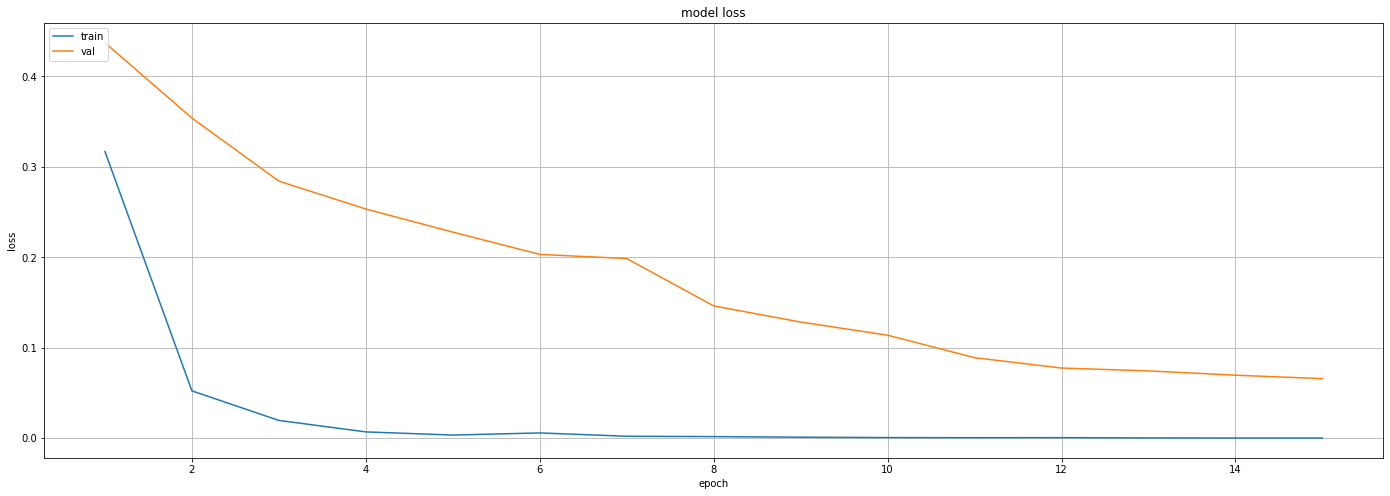
-> 장기기억 가능

-> 데이터 특징 상

층이 많지 않다보니 렐루함수를 적용할 필요는 없지만 혹시나 하는 마음에 적용

배치 정규화 시에 성능이 안좋았던 이유는 dropout과 효과가 비슷한데 데이터 손실이 많이 일어난게 아닌가 의심.



테스트 데이터 정확도 : 0.9014779925346375

배치 정규화 추가 후 속도가 많이 느려졌고 성능도 낮아졌다

model = Sequential() # 자~ 모델을 만들거야~

model.add(Embedding(vocab\_size, 128, input\_length=MAX\_TEXT\_LEN)) # 먼저 임베딩을 해서 단어를 넣어야겠지??

# 이제 본격적으로 모델에 넣어서 출력을 해야지

#model.add(LSTM(128, dropout=0.3, return\_sequences = True)) # 근데 만약에 한번더 모델의 출력을 다시 입력으로 넣게 된다면 무슨 이득이 있지?

model.add(LSTM(128, dropout=0.3, recurrent\_dropout=0.3)) #성능 향상을 위한 스태킹

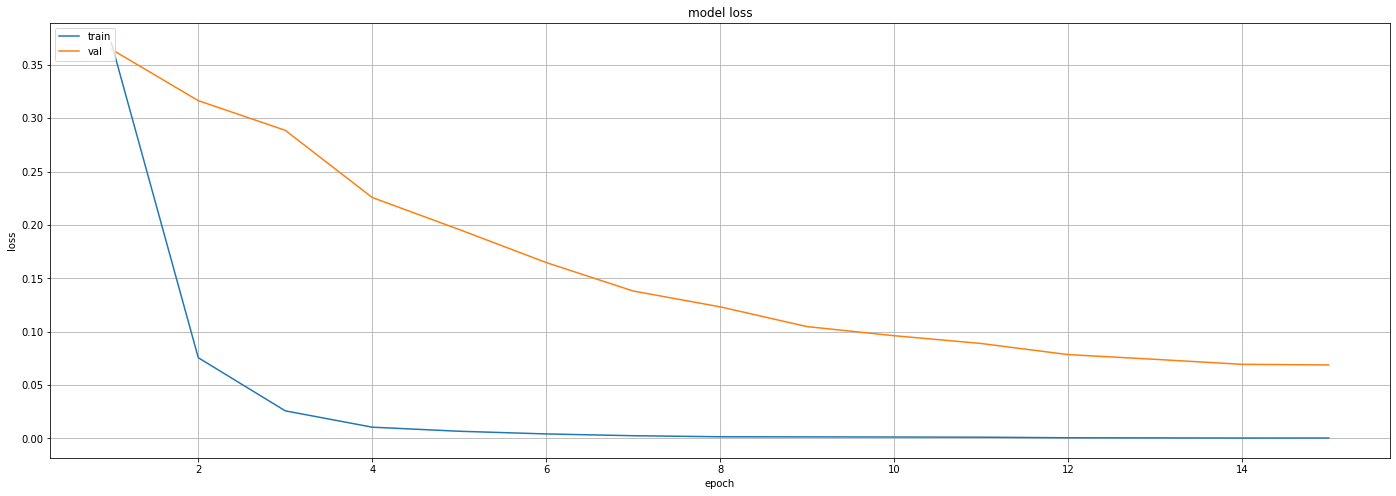
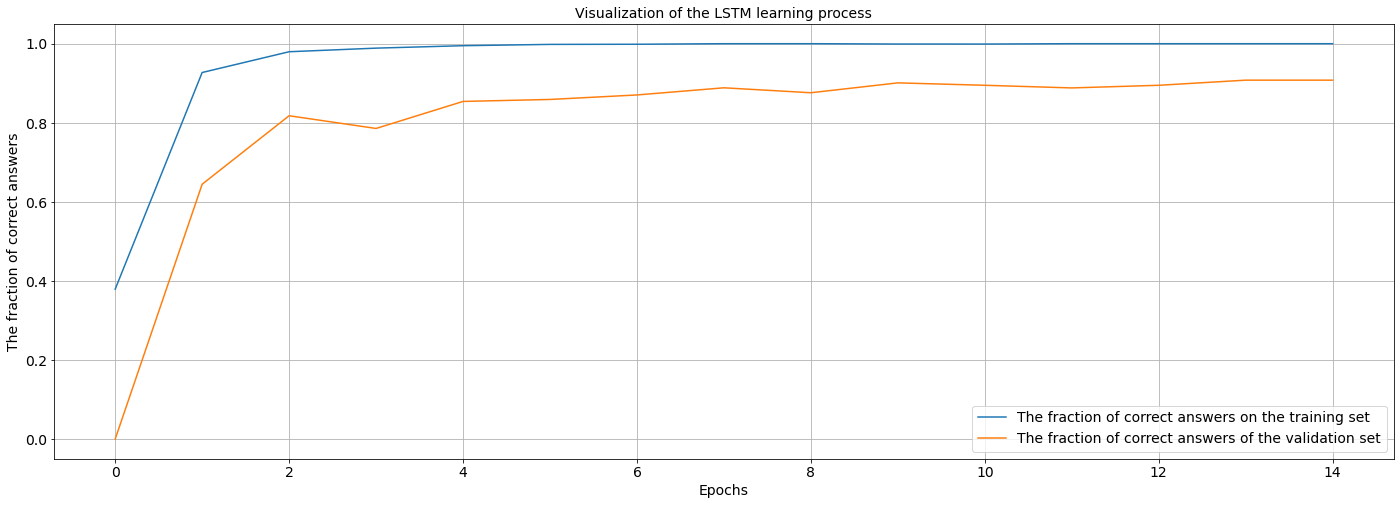
model.add(BatchNormalization())

model.add(Dense(64,activation='relu')) # 혹시나하는 기울기 손실 방지

model.add(Dense(1, activation='sigmoid')) # 활성화 함수를 통해 이진 아웃풋을 출력해~!~!~!~!

물론 유닛 수도 늘렸다.

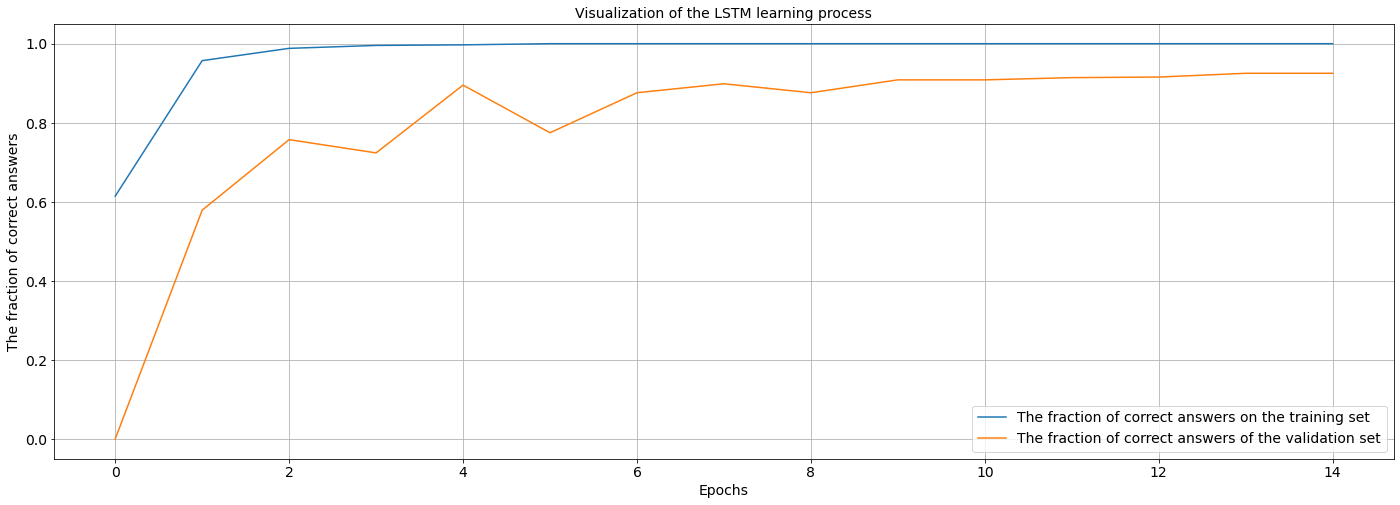
유닛 수를 64로 다시 낮추면

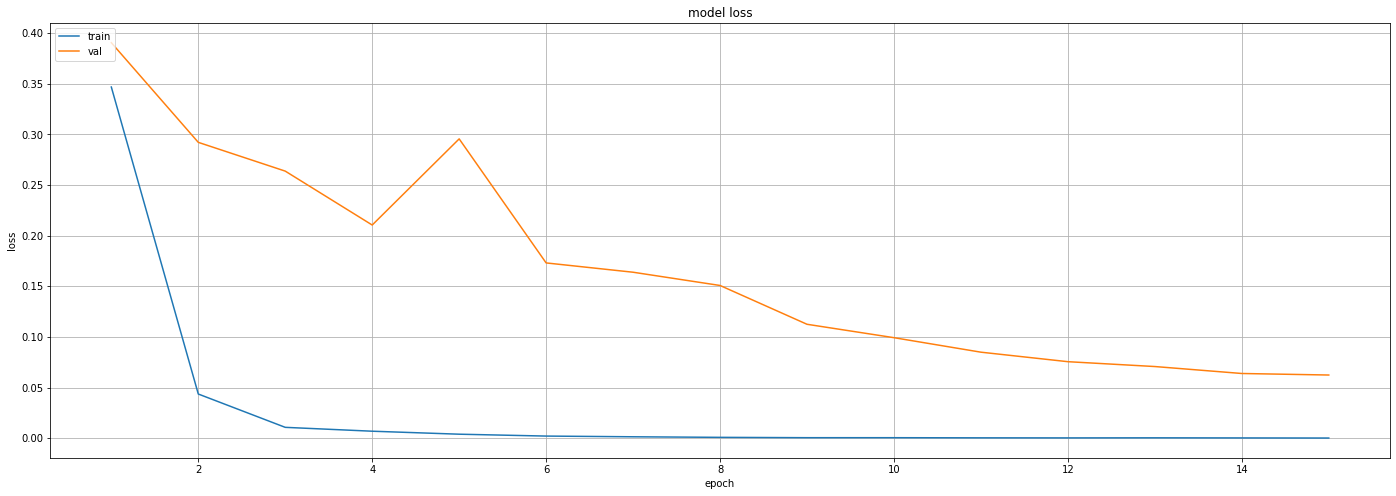


0.9152542372881356

큰 개선이 없다

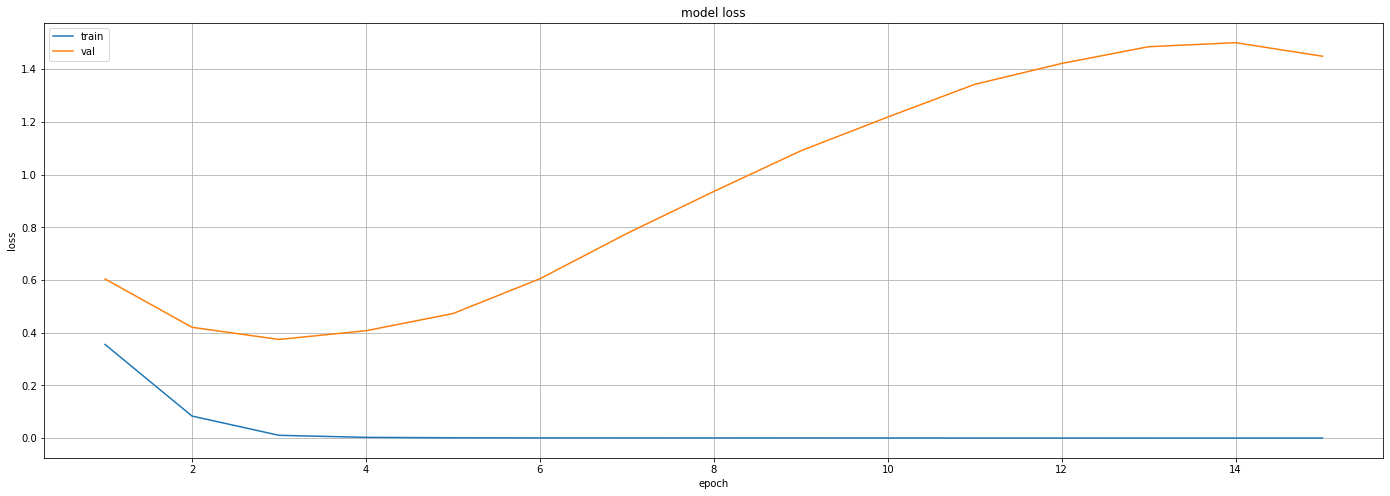
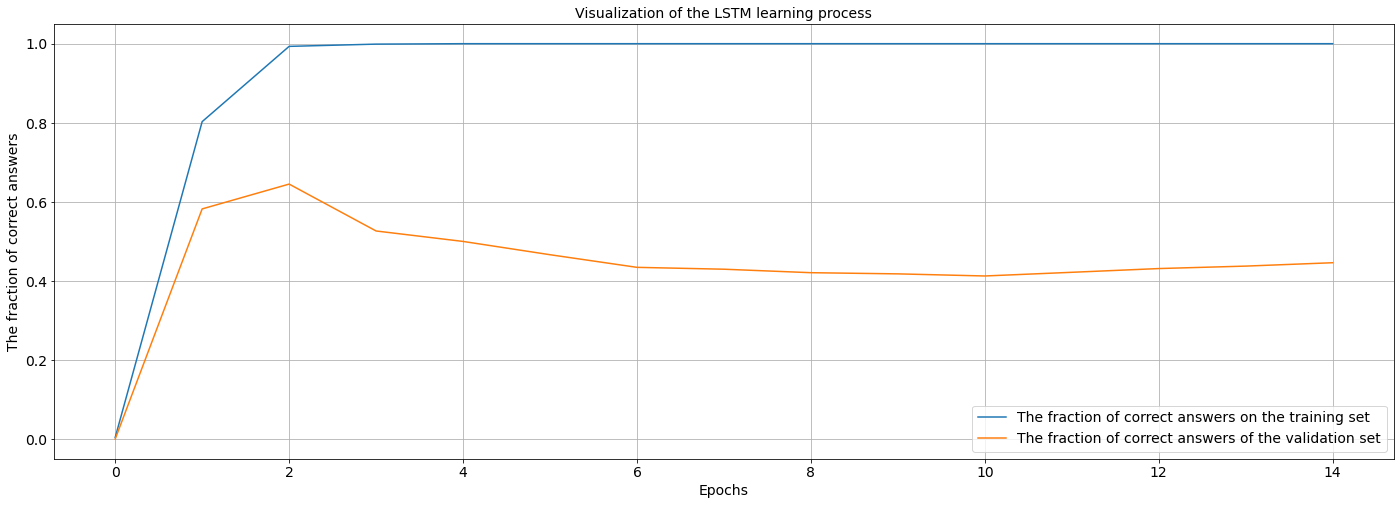
유닛 수는 큰 영향을 미치지 않는다.





배치 정규화와 드롭아웃이 문제가 될 줄 알고 드롭아웃을 없앴지만 더 안좋아짐.

0.8974358974358975



model = Sequential() # 자~ 모델을 만들거야~

model.add(Embedding(vocab\_size, 128, input\_length=MAX\_TEXT\_LEN)) # 먼저 임베딩을 해서 단어를 넣어야겠지??

# 이제 본격적으로 모델에 넣어서 출력을 해야지

#model.add(LSTM(128, dropout=0.3, return\_sequences = True)) # 근데 만약에 한번더 모델의 출력을 다시 입력으로 넣게 된다면 무슨 이득이 있지?

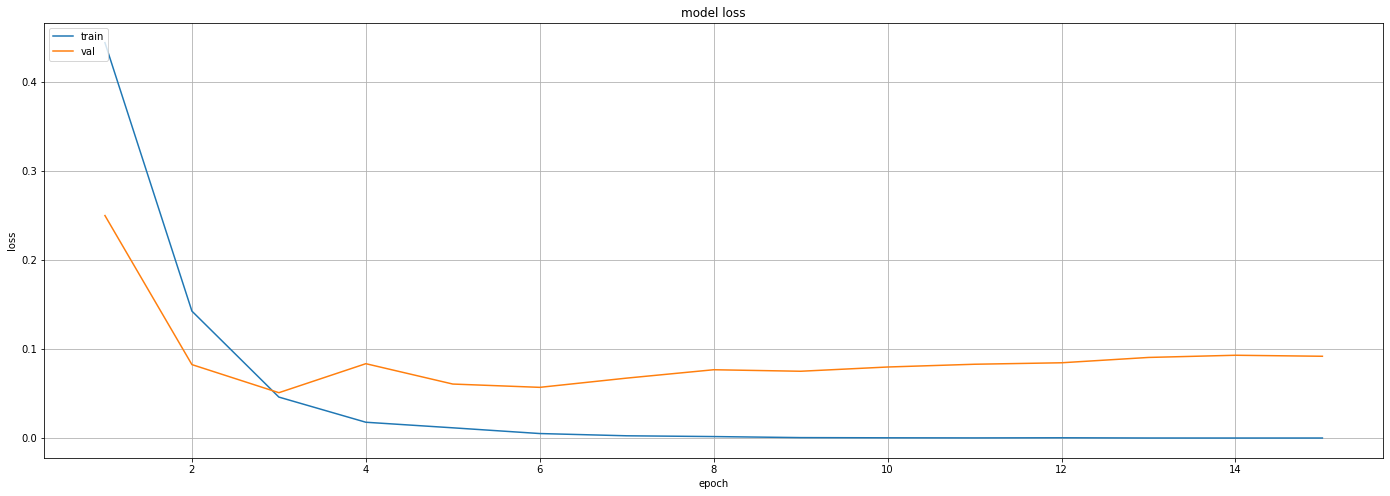
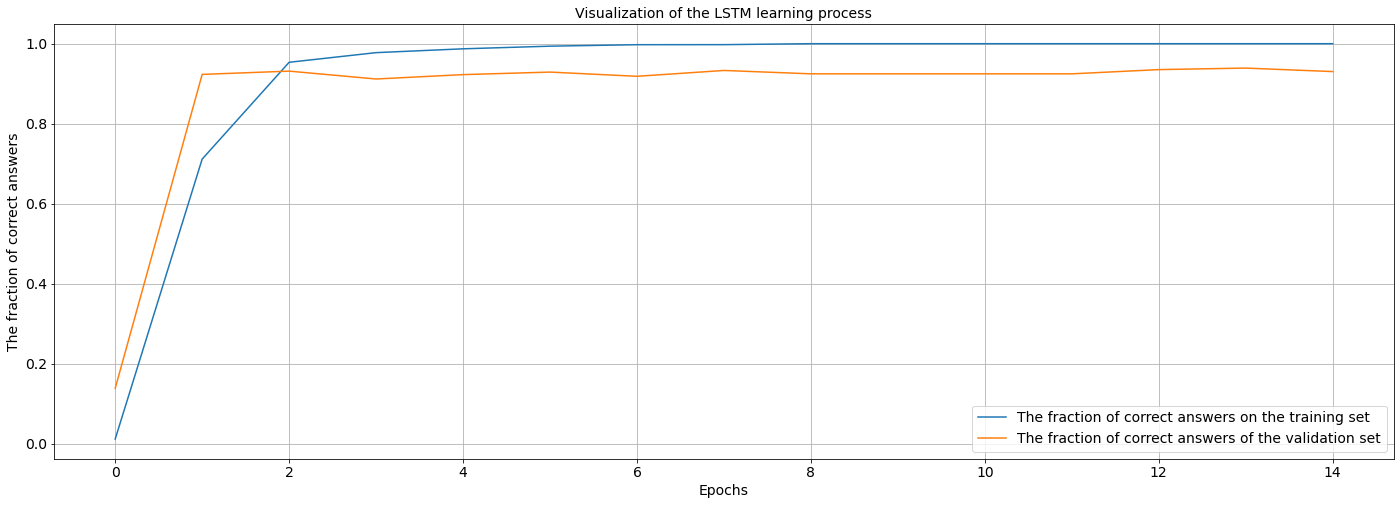
model.add(BatchNormalization())

model.add(LSTM(64)) #성능 향상을 위한 스태킹

model.add(Dense(64,activation='relu')) # 혹시나하는 기울기 손실 방지

model.add(Dense(1, activation='sigmoid')) # 활성화 함수를 통해 이진 아웃풋을 출력해~!~!~!~!

0.48249027237354086

0.9344262295081966

model = Sequential() # 자~ 모델을 만들거야~

model.add(Embedding(vocab\_size, 128, input\_length=MAX\_TEXT\_LEN)) # 먼저 임베딩을 해서 단어를 넣어야겠지??

# 이제 본격적으로 모델에 넣어서 출력을 해야지

#model.add(LSTM(128, dropout=0.3, return\_sequences = True)) # 근데 만약에 한번더 모델의 출력을 다시 입력으로 넣게 된다면 무슨 이득이 있지?

#model.add(BatchNormalization())

model.add(LSTM(64, dropout=0.3, recurrent\_dropout=0.3)) #성능 향상을 위한 스태킹

model.add(Dense(64,activation='relu')) # 혹시나하는 기울기 손실 방지

model.add(Dense(1, activation='sigmoid')) # 활성화 함수를 통해 이진 아웃풋을 출력해~!~!~!~!

다음에서 성공함.

그대로 GRU로만 바꿔보자