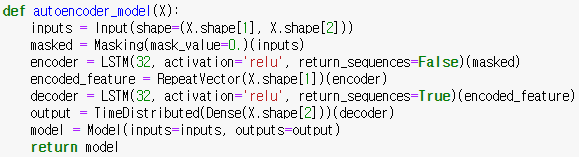
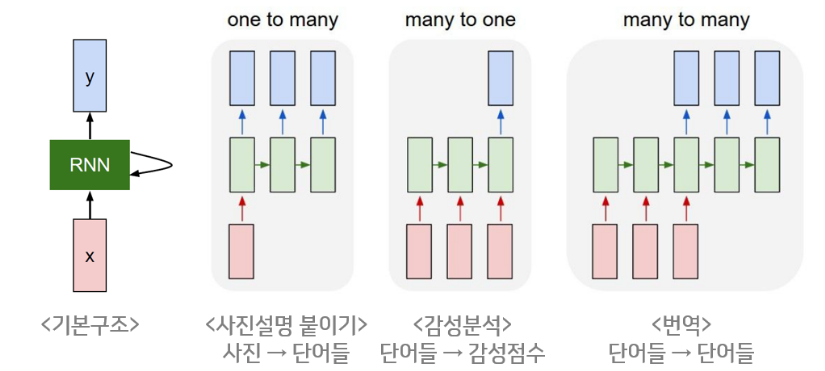
* **수행 내용**
  + LSTM을 공부한 내용을 바탕으로 모델 구성
    - Encoder layer와 Decoder layer 구성
    - variable sequence length에 대한 padding
* **LSTM Autoencoder 모델 구성**
  + **모델 구성**



* + - **Masking (masked)**
      * 패딩된 데이터 중, 의미 있는 데이터를 구분하기 위한 layer
    - **Encoder (encoder)**
      * unit 개수: 32개
      * 활성화 함수: ReLU
    - **RepeatVector (encoded\_feature)**
      * Encoder단에서 1\*32 크기의 Encoded feature vector만 내보냈기 때문에,

Decoder단에 다시 전달하기 위해 input data의 timestep만큼 복사

* + - **Decoder (decoder)**
      * unit 개수: 32개
      * 활성화 함수: ReLU
  + **Many-to-Many 유형으로 설계**



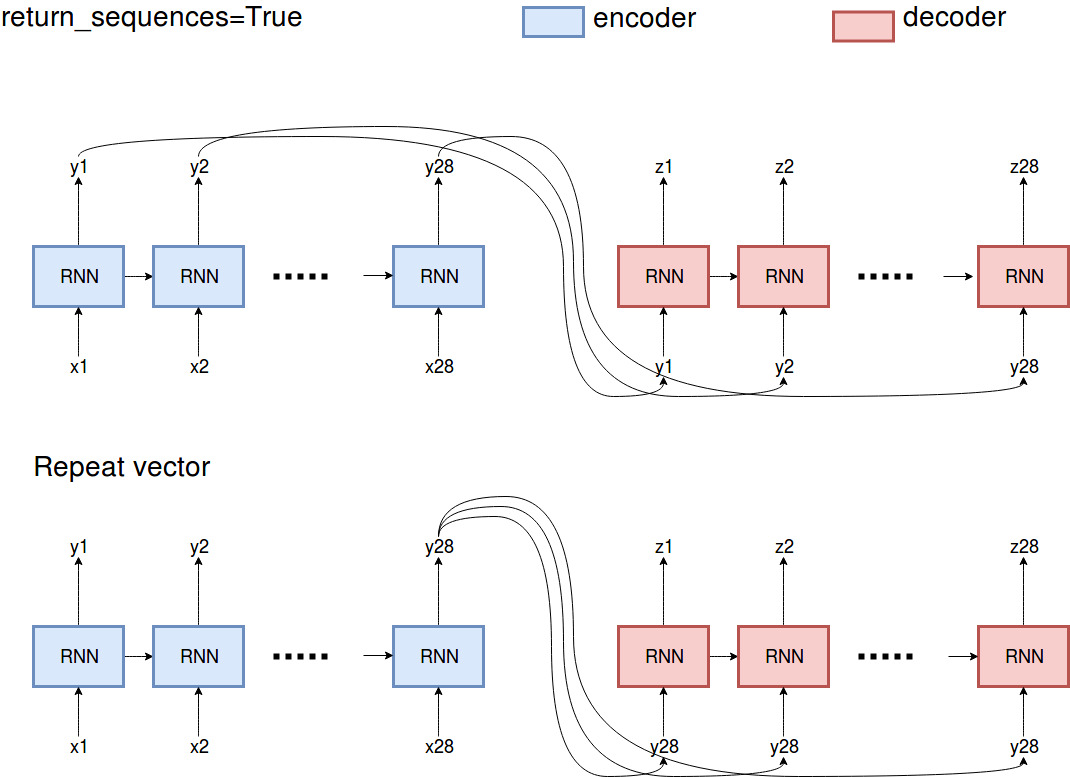
* + - * seq2seq 모델이 영어를 불어로 번역하는 것처럼,

사용자 log의 위치 정보를 segment 정보로 번역

* + - * + input, output 모두 sequential data
      * 이번주는 LSTM Autoencoder가 원본을 잘 복원해낼 수 있는지 테스트를 시도

*사진 출처:* [*Sequence Model (RNN, LSTM) (tistory.com)*](https://dodonam.tistory.com/152?category=885467)

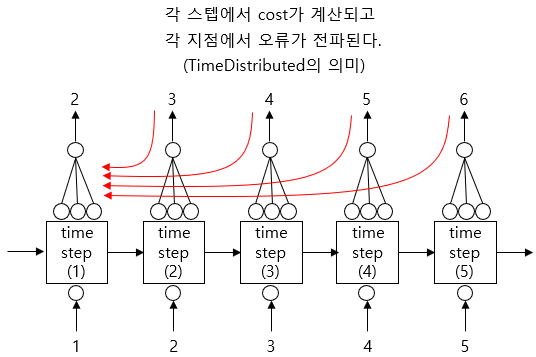
* + - **return\_sequences 인자**



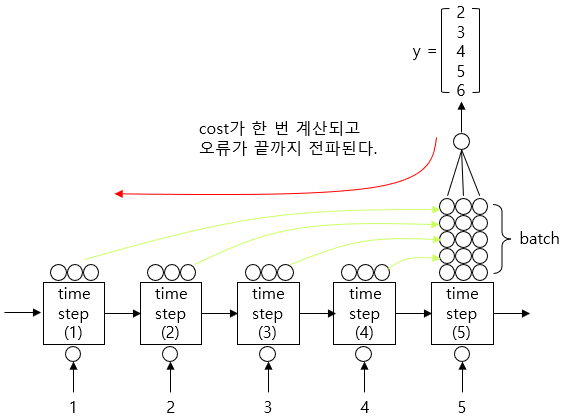
* + - * return\_sequences=True일 때
        + 각 time step에서의 출력을 모두 다음 layer의 각 time step에서 이용
      * return\_sequences=False일 때
        + 마지막 time step의 출력만 이용

*사진 출처:* [*tensorflow - How to connect LSTM layers in Keras, RepeatVector or return\_sequence=True? - Stack Overflow*](https://stackoverflow.com/questions/51749404/how-to-connect-lstm-layers-in-keras-repeatvector-or-return-sequence-true)

* + - **RepeatVector**
      * 모델의 L1에서 return\_sequences=False로 하나의 Encoded Feature만 내보냄
* 다음 layer의 입력으로 사용할 수 있도록 input을 복사하는 layer
  + - **TimeDistributed**



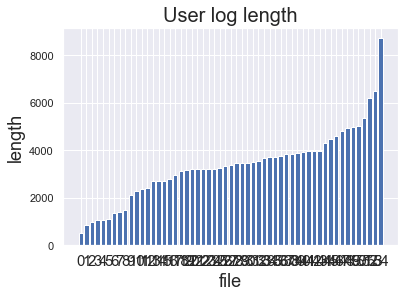
* + - * 각 time step 마다 cost를 계산하고, 하위 step으로 오류를 전파하여 weight를 업데이트
    - **TimeDistributed를 사용하지 않을 경우**



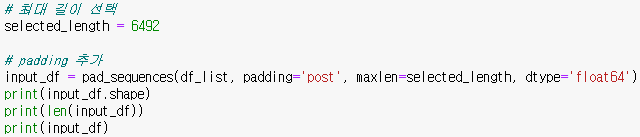
* + - * 마지막 step에서만 cost를 계산하고, 하위 step으로 오류를 전파

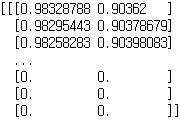
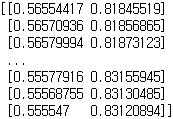
*사진 출처:* [*﻿2. Keras LSTM 유형 정리 (2/5) – 단층-단방향 & many-to-many 유형 : 네이버 블로그 (naver.com)*](https://m.blog.naver.com/chunjein/221589624838)

* + **Variable sequence length에 대한 padding**
    - **Padding**
      * 기준으로 잡은 vector 길이보다 작으면, 0으로 채우는 기법
      * 유저 0mnEB226qqgHE79KLEfxRj6fiEK2의 모든 log data의 행 수

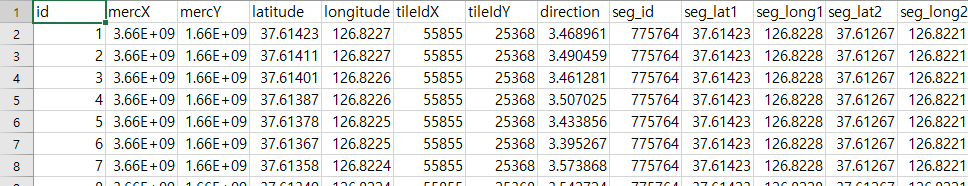
****

* + - * keras의 pad\_sequences 함수를 이용해서 데이터의 뒷부분을 0으로 채워서 sequence를 같도록 만듦



(패딩 된 data) (패딩이 필요없는 data)

* + - **(참고) 원본 데이터**

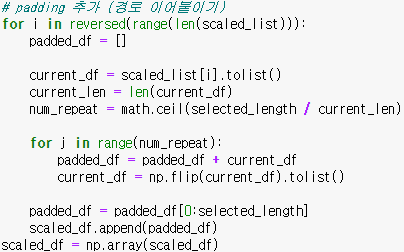


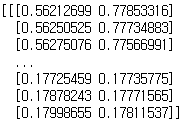
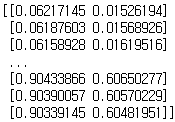
* + - **Masking**
      * padding된 데이터에서, 의미 있는 데이터를 구분



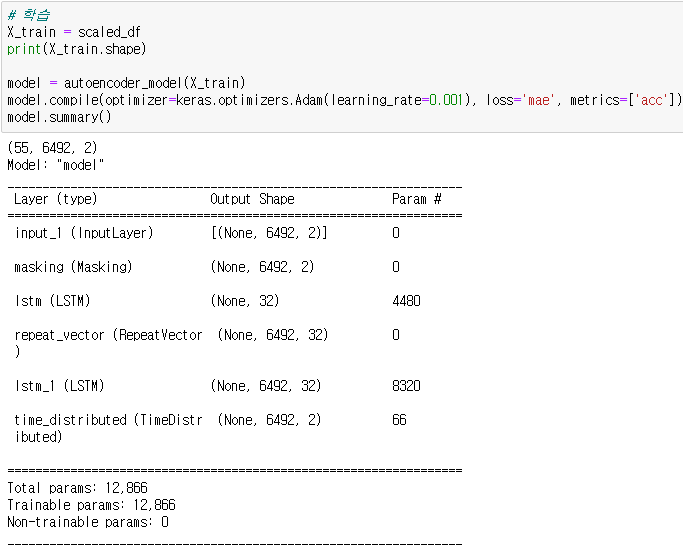
* + - * 값이 0인 부분은 False, 그 외에는 True로 표시하여 학습에 사용
    - **0으로 채우는 padding 외에 시도해본 padding (+ input 역순으로 변환)**
      * 지정한 length에 도달할 때까지, 해당 log의 경로를 왕복하는 것처럼 이어 붙이기
        + length = 10이라고 했을 때의 예시



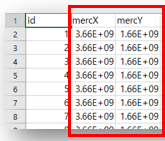
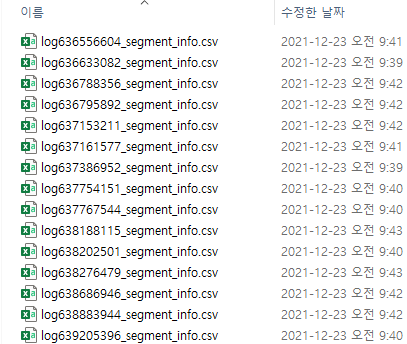


(패딩 된 data 1) (패딩 된 data 2)

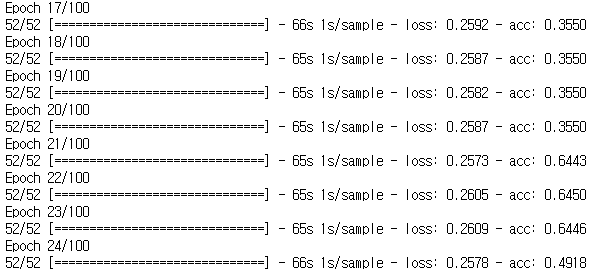
* + **실제 학습 1 (user 1의 모든 log 이용 + padding, masking 수행)**



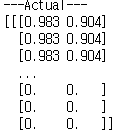
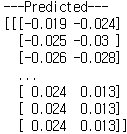
* + - 55개의 sample
    - window\_size = 6492
    - 2개의 feature (mercX, mercY)



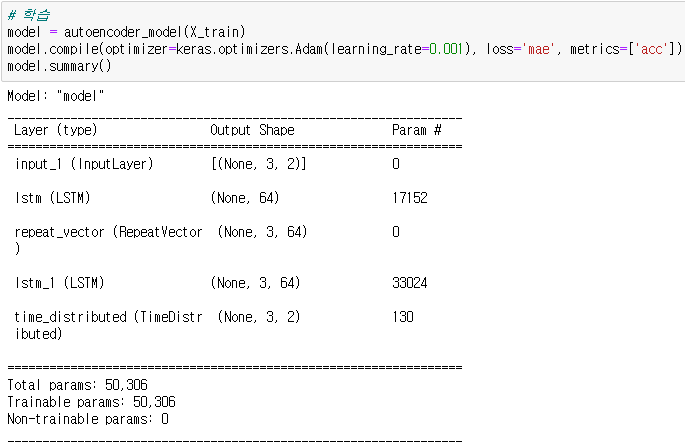
* + **학습 결과**
    - 처음부터 끝까지 accuracy가 0.355부터 0.644정도까지 왔다갔다하며 제대로 학습이 되지 않는 모습을 보임

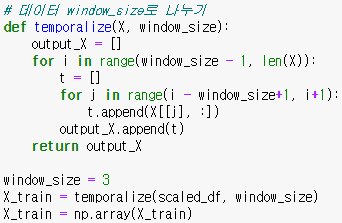
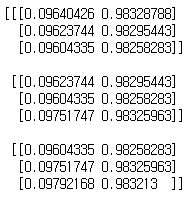


* + - 모델에 의해 예측된 값

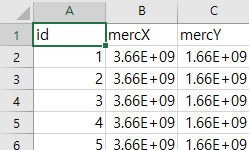
(실제 값) (예측된 값)

* + **실제 학습 2 (user 1의 log 한개만 이용 + padding, masking 수행 X)**

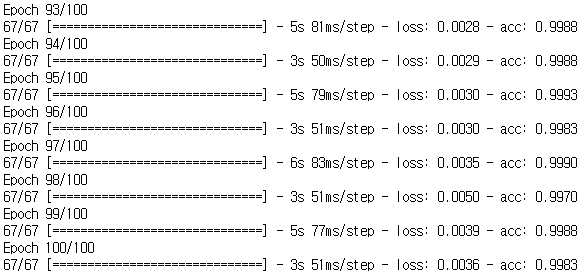


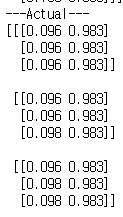
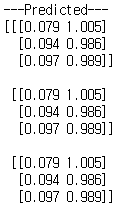
* + - 1405개의 sample
    - window\_size = 3
    - 2개의 feature (mercX, mercY)



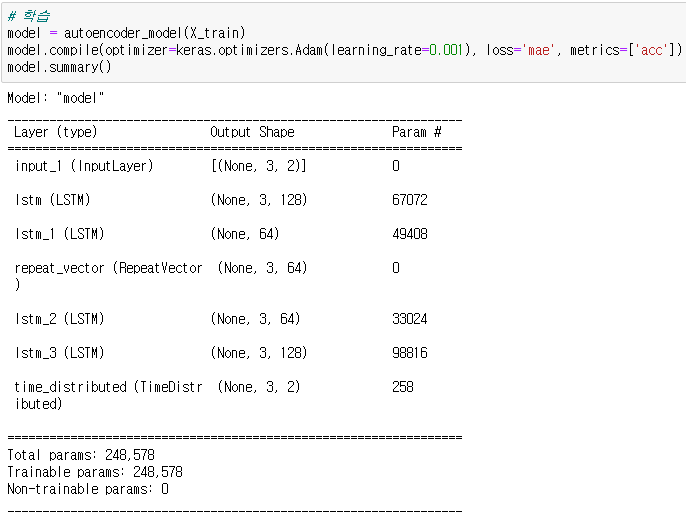
* + 학습 결과
    - 높은 accuracy를 보이며 잘 학습이 된 것으로 보였지만, 예측된 값은 만족스럽지 않음



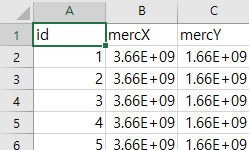
* + - 모델에 의해 예측된 값

(실제 값) (예측된 값)

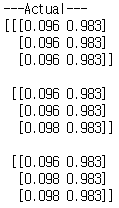
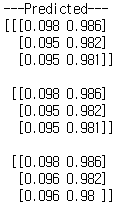
* + **실제 학습 3 (user 1의 log 한개만 이용 + padding, masking 수행 X)**

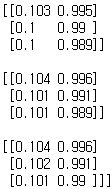
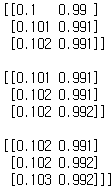


* + - 1405개의 sample
    - window\_size = 3
    - 2개의 feature (mercX, mercY)

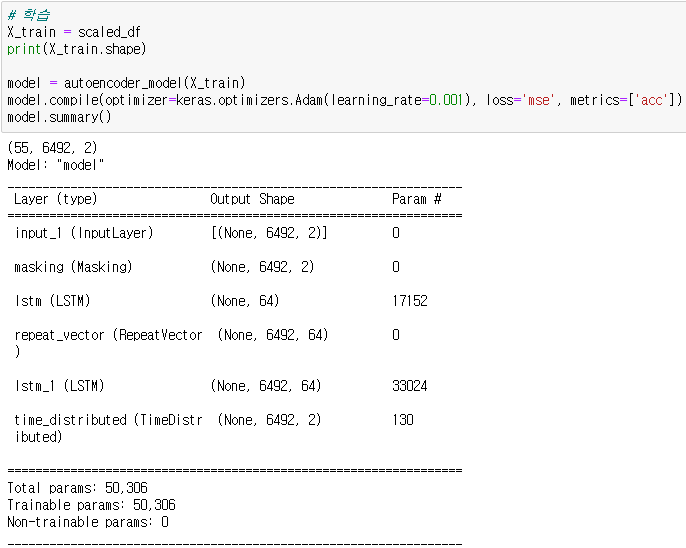


* + 학습 결과
    - 학습 2보다 더 비슷한 결과를 보임
    - 모델에 의해 예측된 값

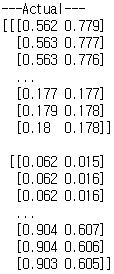
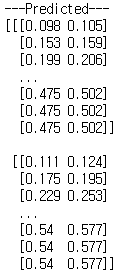
 (실제 값)  (예측된 값)

(실제 값) (예측된 값)

* + **실제 학습 4 (user 1의 모든 log 이용 + custom padding 수행)**



* + - 모델에 의해 예측된 값

 (실제 값)  (예측한 값)