딥러닝/클라우드

Chapter 16

RNN (Recurrent Neural Network)

오 세 종

DANKOOK UNIVERSITY

Contents

1. 자연어 처리
2. RNN
3. LSTM
4. Keras LSTM

참고: https://www.youtube.com/watch?v=Hn3GHHOXKCE

1. 자연어 처리

- 개요
  - 자연어(natural language): 우리가 일상 생활에서 사용하는 언어
  - 자연어 처리(natural language processing): 이러한 자연어의 의미를 분석하여 컴퓨터가 처리할 수 있도록 하는 작업
  - 자연어 처리의 예:
    - 음성 인식, 내용 요약, 번역, 사용자의 감성 분석, 텍스트 분류 작업(스팸 메일 분류, 뉴스 기사 카테고리 분류), 질의 응답 시스템, 챗봇
  - 최근 딥 러닝의 뛰어난 성능이 괄목할만한 성과를 얻으면서, 인공지능이 IT 분야의 중요 키워드로 떠오르고 있음.
  - 자연어 처리는 기계에게 인간의 언어를 이해시킨다는 점에서 인공지능에서 가장 의미있는 연구 분야이면서도 아직도 정복되어야 할 산이 많은 분야

1. 자연어 처리

- Corpus(코퍼스, 말뭉치)
  - 체계적으로 수집되고 구성된 텍스트 모음
  - 특정 언어, 방언, 주제, 스타일 또는 시대를 대표하는 텍스트들을 포함할 수 있다.
  - 자연어 처리 분야에서 말뭉치는 언어 모델을 훈련하고, 언어의 통계적 패턴을 분석하며, 다양한 컴퓨터 기반 언어 연구에 사용

- Tokenization(토큰화)
  - 주어진 코퍼스(corpus)에서 토큰(token)이라 불리는 단위로 나누는 작업

Time is an illusion. Lunchtime double so!

"Time", "is", "an", "illustion", "Lunchtime", "double", "so"

  - 한글의 경우 어떤 기준으로 토큰을 분리할지가 쉽지 않다. (띄어쓰기 기준?)

1. 자연어 처리

- Integer Encoding(정수 인코딩)
  - 컴퓨터는 텍스트보다는 숫자를 더 잘 처리
  - 자연어 처리에서는 텍스트를 숫자로 바꾸는 여러가지 기법들 존재
  - 각 단어를 고유한 정수에 맵핑(mapping)시키는 전처리 작업이 필요

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5}
```

  - 단어를 표현하는 방법에 따라서 자연어 처리의 성능이 크게 달라지기 때문에 단어를 수치화 하기 위한 많은 연구 진행

1. 자연어 처리

- Integer Encoding(정수 인코딩)
  - Sparse Representation (희소표현)

단어가 10,000개 있고 인덱스가 0부터 시작하면서 강아지란 단어의 인덱스는 4
강아지 = [0 0 0 0 1 0 0 0 0 0 0 0 ... 중략 ... 0] # 이 벡터의 차원은 10000

  - Dense Representation (밀집표현)

사용자가 밀집 표현의 차원을 128로 설정한다면
강아지 = [0.2 1.8 1.1 -2.1 1.1 2.8 ... 중략 ...] # 이 벡터의 차원은 128

  - word embedding
    - 밀집 표현과 유사
    - 임베딩 벡터 도출

	원-핫 벡터	임베딩 벡터
차원	고차원(단어 집합의 크기)	저차원
다른 표현	희소 벡터의 일종	밀집 벡터의 일종
표현 방법	수동	훈련 데이터로부터 학습함
값의 타입	1과 0	실수

```
========== 원래 문장  ============
<PAD> <PAD> <START> this film was just brilliant casting location scenery story
direction everyone's really suited the part they played and you could just imagine
being there robert <OOV> is an amazing actor and now the same being director <OOV>
father came from the same scottish island as myself so i loved the fact there was a
real connection with this film the witty remarks throughout the film were great it was
just brilliant so much that i bought the film as soon as it was released for <OOV> and
would recommend it to everyone to watch and the fly fishing was amazing really cried
at the end it was so sad and you know what they say if you cry at a film it must have
been good and this definitely was also <OOV> to the two little boy's that played the
<OOV> of norman and paul they were just brilliant children are often left out of the
<OOV> list i think because the stars that play them all grown up are such a big <OOV>
for the whole film but these children are amazing and should be praised for what they
have done don't you think the whole story was so lovely because it was true and was
someone's life after all that was shared with us all

============ 임베딩된 문장 확인 ============
(1, 400, 60)
[[[ 0.0088832  -0.0436639   0.07654434 ...  0.04261591 -0.00109518
    0.04554922]
  [ 0.0088832  -0.0436639   0.07654434 ...  0.04261591 -0.00109518
    0.04554922]
  [ 0.0088832  -0.0436639   0.07654434 ...  0.04261591 -0.00109518
    0.04554922]
  ...
  [ 0.05499315  0.07495009 -0.06593661 ...  0.02310263 -0.03915591
    0.03818268]
  [ 0.17379531 -0.04260442 -0.03142205 ...  0.01213738 -0.00449449
   -0.17220676]
  [-0.06178711  0.07810801  0.01260059 ...  0.11740308 -0.0989247
    0.01683315]]]
```

임베딩 벡터 길이: 60

Ref: https://sirzzang.github.io/ai/AI-NLP-WordEmbedding-Keras/

1. 자연어 처리

- Padding
  - 자연어 처리를 하다보면 각 문장(또는 문서)은 서로 길이가 다를 수 있음
  - 딥러닝 모델 입력을 위해서는 입력 문장의 길이가 같아야함
  - 패딩은 여러 문장의 길이를 임의로 동일하게 맞춰주는 작업

  - Zero padding 의 예

```
array([[ 1,  5,  0,  0,  0,  0,  0],
       [ 1,  8,  5,  0,  0,  0,  0],
       [ 1,  3,  5,  0,  0,  0,  0],
       [ 9,  2,  0,  0,  0,  0,  0],
       [ 2,  4,  3,  2,  0,  0,  0],
       [ 3,  2,  0,  0,  0,  0,  0],
       [ 1,  4,  6,  0,  0,  0,  0],
       [ 1,  4,  6,  0,  0,  0,  0],
       [ 1,  4,  2,  0,  0,  0,  0],
       [ 7,  7,  3,  2, 10,  1, 11],
       [ 1, 12,  3, 13,  0,  0,  0]])
```

1. 자연어 처리

- 인공지능 언어 모델
  - LLM (large language model)
    - 언어 이해, 생성, 번역, 요약 등 광범위한 NLP 작업에서 뛰어난 성능
    - GPT (GPT4: 1.8 trillion(조))
    - BARD
    - PaLM
    - LLaMa
  - Small LLM
    - 특정 도메인이나 작업에 특화되어 있을 때 좋은 성능
    - Phi3-mini (3.8B, microsoft)
    - ALPACA 7B (Small LLaMa, meta)
    - Gemma 7B (google)
    - Mistral 7B (Mistral AI)

1. 자연어 처리

- 인공지능 언어 모델
  - sLLM 의 장점
    - 개발비용 절감
    - 학습및 훈련시간 단축
    - 다양한 기계에서 사용가능 (모바일 기기 포함)
    - 할루시네이션 발생율 낮음
  - sLLM 의 단점
    - 범용성 부족
    - 데이터 수집의 어려움
    - 전문지식 필요

- Hugging face 에 다양한 open source sLLM 모델 소개
  - https://huggingface.co/models

1. 자연어 처리

- NLP를 위한 딥러닝 이론
  - RNN 이 기본
    - many-to-one 구조 : 텍스트 분류
    - many-to-many 구조 : 개체명 인식, 품사 태깅
    - one-to-many 구조 : 텍스트 생성, 챗봇
  - Seq2seq
    - 입력된 시퀀스로부터 다른 도메인의 시퀀스를 출력
    - Encoder-decoder 구조
    - 챗봇, 기계번역, 내용 요약, Speech to Text
  - Attention mechanism
    - 입력 문장이 길면 번역 품질이 떨어지는 현상
    - 이를 해결하기 위한 대안 등장
  - Transformer
    - 2017년 구글 발표
    - 기존의 seq2seq의 구조인 인코더-디코더를 따르면서도, 어텐션(Attention)만으로 구현한 모델 (RNN 사용하지 않음)
    - 오늘날 대부분의 LLM 모델은 Transformer 기반

[실습] 문장 분류

16_text_classification.py

```
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

sentences = ['nice great best amazing',
             'stop lies',
             'pitiful nerd',
             'excellent work',
             'supreme quality',
             'bad',
             'highly respectable']

y_train = [1, 0, 0, 1, 1, 0, 1]   # 긍정: 1, 부정: 0
```

- Learn about tokenization, word embedding

Ref: https://wikidocs.net/33793

```
## 토큰화
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)
vocab_size = len(tokenizer.word_index) + 1 # 패딩을 고려하여 +1
print('단어 집합 :',vocab_size)

X_encoded = tokenizer.texts_to_sequences(sentences)
print('정수 인코딩 결과 :',X_encoded)
```

```
>>> print('단어 집합 :',vocab_size)
단어 집합 : 16

>>> print('정수 인코딩 결과 :',X_encoded)
정수 인코딩 결과 : [[1, 2, 3, 4], [5, 6], [7, 8], [9, 10], [11, 12], [13], [14, 15]]
```

```
['nice great best amazing',
 'stop lies',
 'pitiful nerd',
 'excellent work',
 'supreme quality',
 'bad',
 'highly respectable']
```

```
## 가장 긴 문장 찾기
max_len = max(len(l) for l in X_encoded)
print('최대 길이 :',max_len)

## 패딩 삽입
X_train = pad_sequences(X_encoded, maxlen=max_len, padding='post')
y_train = np.array(y_train)
print('패딩 결과 :')
print(X_train)
```

```
>>> print('최대 길이 :',max_len)
최대 길이 : 4
>>> print('패딩 결과 :')
패딩 결과 :
>>> print(X_train)
[[ 1  2  3  4]
 [ 5  6  0  0]
 [ 7  8  0  0]
 [ 9 10  0  0]
 [11 12  0  0]
 [13  0  0  0]
 [14 15  0  0]]
```

```
# Model 구성
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, Flatten

embedding_dim = 4

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['acc'])
model.fit(X_train, y_train, epochs=100, verbose=2)
```

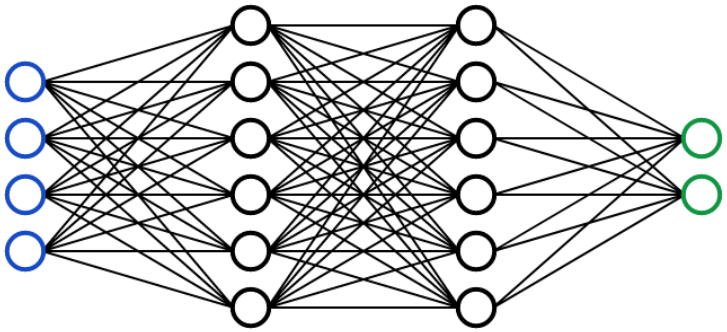
```
# prediction

result = model.predict(X_train)
print(result)
pred = [1 if i > 0.5 else 0 for i in result]
print(pred)
```

```
>>> print(result)
[[0.66449857]
 [0.4234189 ]
 [0.42209494]
 [0.59122306]
 [0.5957805 ]
 [0.47638056]
 [0.59563446]]
>>> pred = [1 if i > 0.5 else 0 for i in result]
>>> print(pred)
[1, 0, 0, 1, 1, 0, 1]
```

```
['nice great best amazing',
 'stop lies',
 'pitiful nerd',
 'excellent work',
 'supreme quality',
 'bad',
 'highly respectable']
```

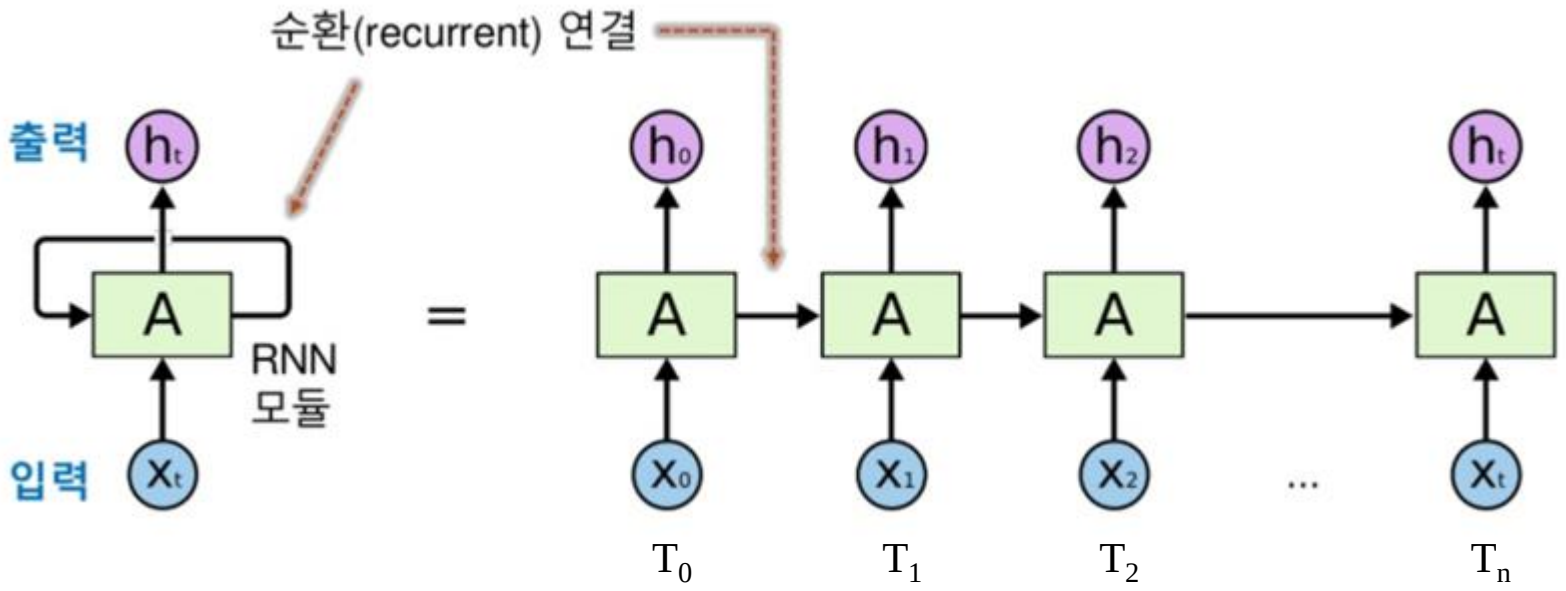
2. RNN

- RNN (Recurrent Neural Network)
  - 일반적인 신경망(FFNets; Feed-Forward Neural Networks)에서는 데이터를 입력하면 연산이 입력층에서 은닉층을 거쳐 출력층까지 차근차근 진행되고 이 과정에서 입력 데이터는 모든 노드를 딱 한번씩 지나가게 됨
  - FFNets는 시간 순서를 무시하고 현재 주어진 데이터만 가지고 판단



2. RNN

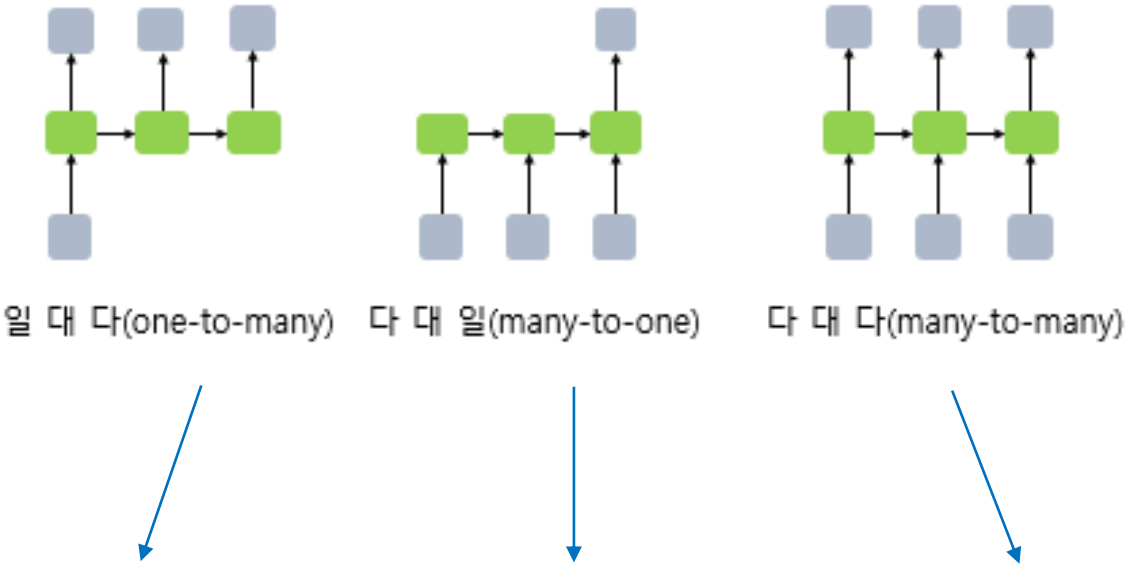
- RNN은 이전 은닉층의 결과가 다음번에 다시 같은 은닉층의 입력으로 들어가도록 연결
- 입력과 출력을 시퀀스 단위로 처리하는 시퀀스(Sequence) 모델
- Ex) 번역. (입력) 번역하고자 하는 문장 (단어의 시퀀스)
  (출력) 번역된 문장 (단어의 시퀀스)



https://ebbnflow.tistory.com/135

2. RNN

- RNN은 입력과 출력의 길이를 다르게 설계 할 수 있음



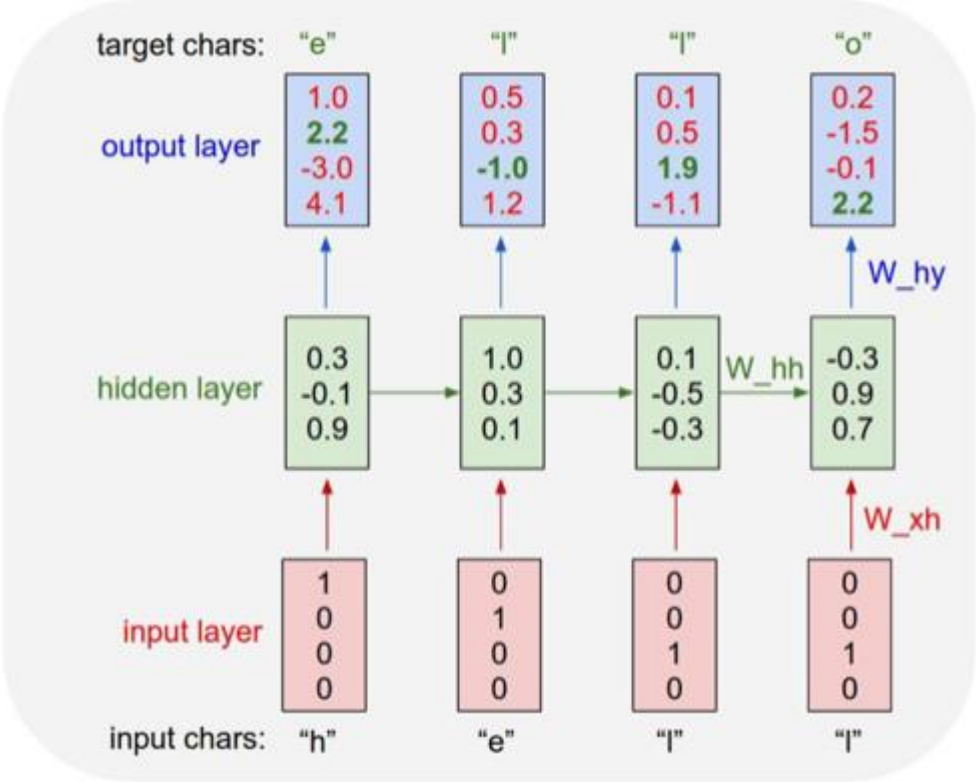
https://wikidocs.net/22886

2. RNN

- RNN의 활용 분야
  - Sequence Data를 다루는데 적합함
  - 글의 문장, 유전자, 손글씨, 음성 신호, 센서가 감지한 데이타, 주가 등 배열(sequence, 또는 시계열 데이터)의 형태를 갖는 데이터에서 패턴을 인식하는 인공 신경망

2. RNN

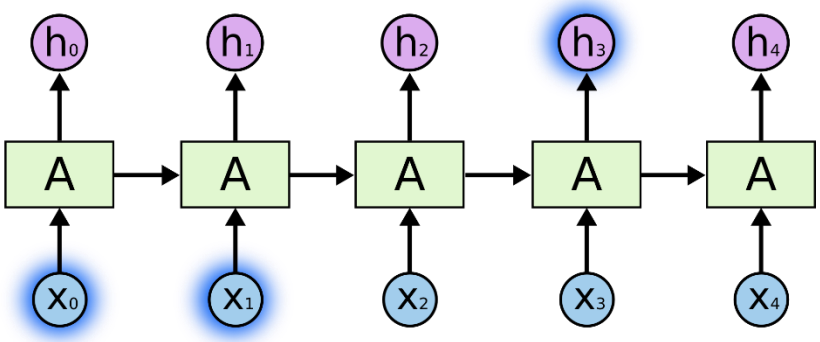
- 문자가 주어졌을 때 다음 문자를 예측하는 RNN모델
  - ‘hell'를 입력하면 'hello'를 출력



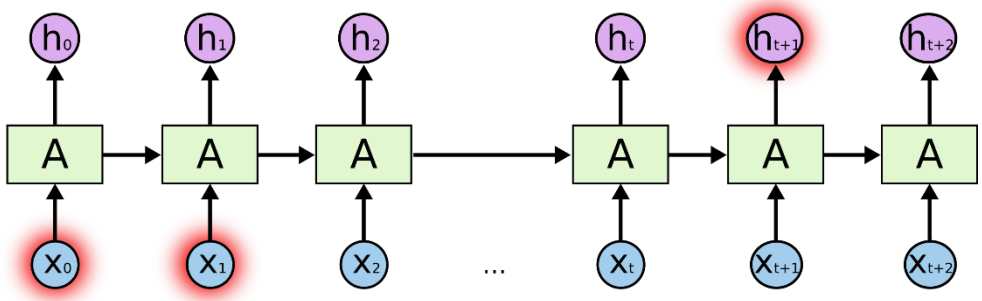
https://ebbnflow.tistory.com/135

3. LSTM

- RNN 의 단점



시간 격차가 크지 않은 경우는 잘 작동 (x_0, x_1 이 h_3에 잘 전달

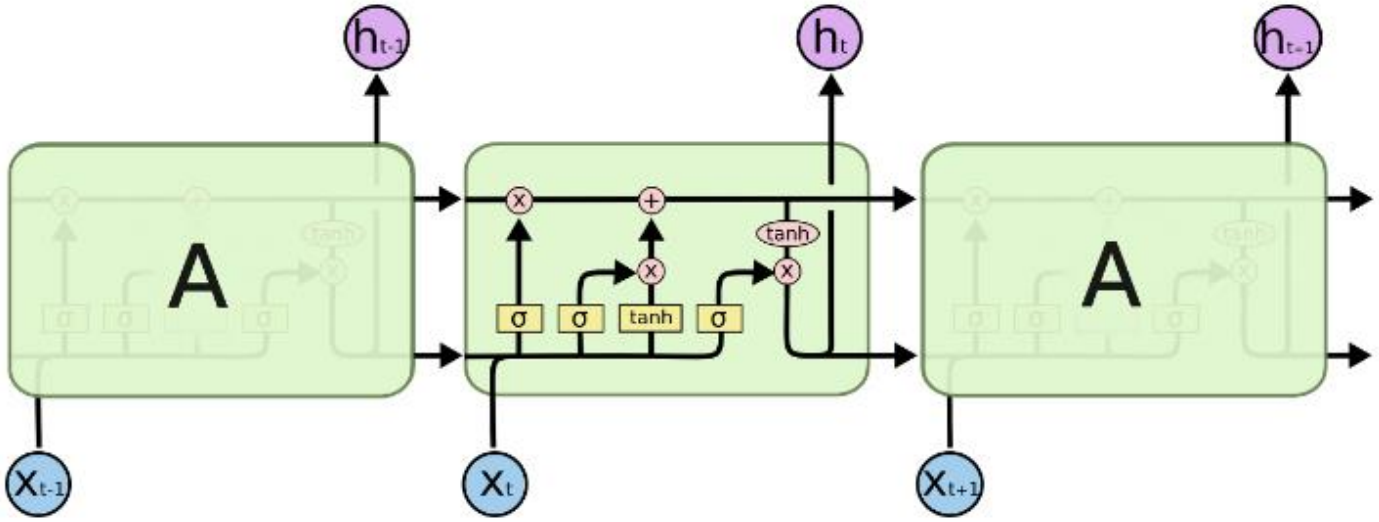


시간 격차가 큰 경우는 정보 전달이 잘 안됨 (x_0, x_1 이 h_{t+1}에 잘 전달되지 않음

https://dgkim5360.tistory.com/entry/understanding-long-short-term-memory-lstm-kr (설명 잘되어 있음)

3. LSTM

- LSTM (Long Short Term Memory network)
  - LSTM은 RNN의 특별한 한 종류로, 긴 의존 기간을 필요로 하는 학습을 수행할 능력을 갖음
  - 음성인식, 언어모델링, 번역, 이미지 캡셔닝 등 다양한 분야에 성공적으로 적용됨

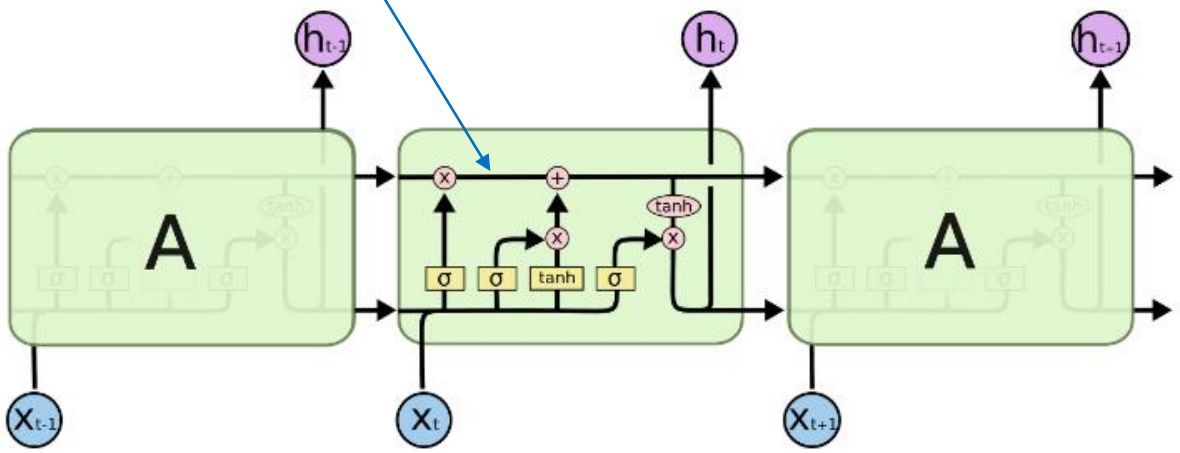


3. LSTM

- LSTM (Long Short Term Memory network)
  - LSTM도 RNN과 똑같이 체인과 같은 구조를 가지고 있지만 , 반복 모듈의 구조가 다름
  - 단순한 neural network layer 한 층 대신에, 4개의 layer가 특별한 방식으로 서로 정보를 주고 받도록 설계

이전의 정보중 필요한 것을 취사선택하고, 현재 입력과 결합하여 출력 정보를 생성

cell state (일종의 컨베이어벨트)



4. Keras LSTM

https://3months.tistory.com/168

- Dataset
  - cansim-0800020-eng-6674700030567901031.csv (link)
  - 북미지역의 월별 매출 데이터

```
1   "Table 080-0020 Retail trade, sales by the North American Industry Classification System (NAICS),
2   Survey or program details:
3   Retail Trade Survey (Monthly) - 2406
4   Monthly Retail Trade Survey (Department Store Organizations) - 2408
5   Geography,Canada,Canada
6   North American Industry Classification System (NAICS),Retail trade [44-45] ,Retail trade [44-45]
7   Adjustments,Unadjusted,Seasonally adjusted
8   Jan-1991,12588862,15026890
9   Feb-1991,12154321,15304585
10  Mar-1991,14337072,15413591
11  Apr-1991,15108570,15293409
12  May-1991,17225734,15676083
13  Jun-1991,16342833,15507931
14  Jul-1991,15996243,15556313
15  Aug-1991,16064910,15430645
```

```
317   Oct-2016,44770113,45061618
318   Nov-2016,46285062,45141762
319   Dec-2016,50016137,44943929
320   Jan-2017,37628452,45952103
321   Footnotes:
322   2,The total for retail trade excludes North American Industry Classification System
323   3,"This CANSIM table replaces archived table 80-0014, 80-0015 and 80-0017."
324   4,"Quality indicator: Code A=Excellent. Code B=Very good. Code C=Good. Code D=Accept
325   5,"Data for Northwest Territories includes Nunavut, from 1991-01 to 1998-12."
326   6,"In April 2013, data from 2004 onwards will be based on the 2012 North American Ir
327   Source:
328   "Statistics Canada. Table 080-0020 - Retail trade, sales by the North American Indus
329   "(accessed: April 19, 2017)"
```

4. Keras LSTM

16_LSTM.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('data/cansim-0800020.csv',
                 skiprows=6, skipfooter=9, engine='python')
df.head()
```

```
In [11]: df.head()
Out[11]:
  Adjustments  Unadjusted  Seasonally adjusted
0    Jan-1991    12588862              15026890
1    Feb-1991    12154321              15304585
2    Mar-1991    14337072              15413591
3    Apr-1991    15108570              15293409
4    May-1991    17225734              15676083
```

날짜 매출액 계절요인을 고려한 매출액

4. Keras LSTM

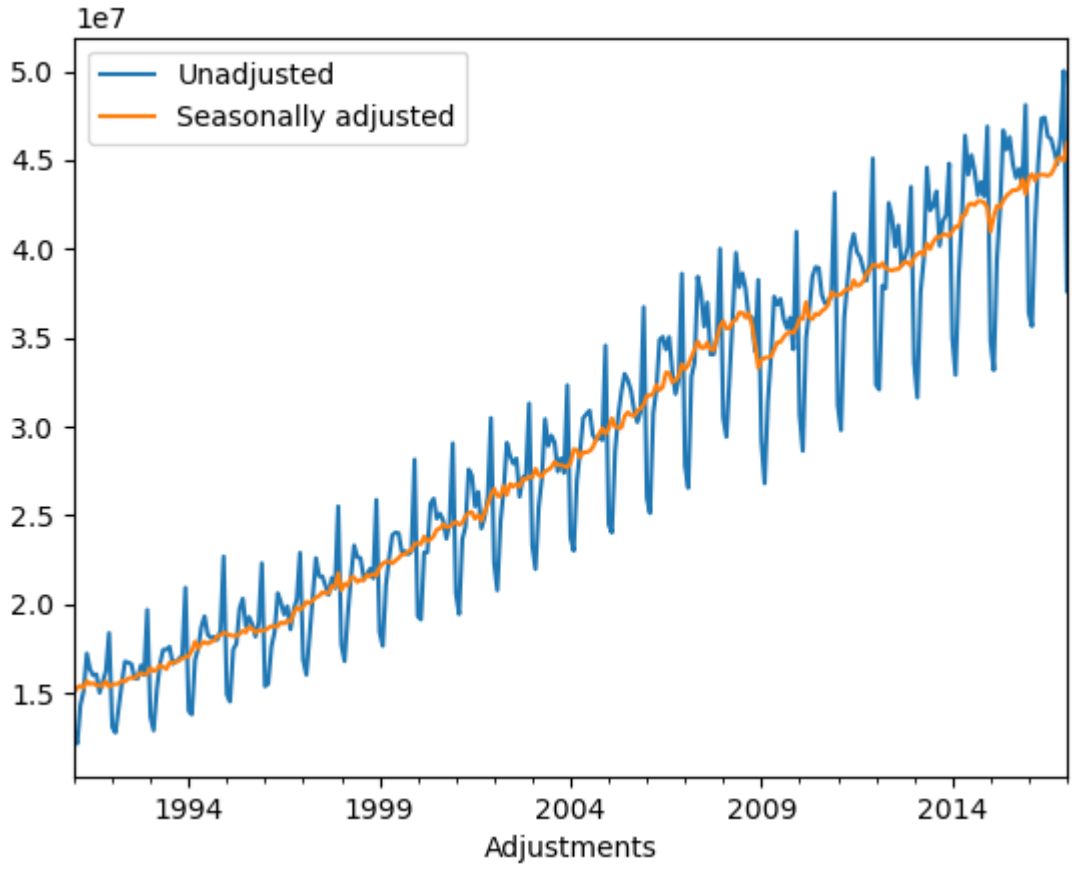
- 데이터 전처리

```
# 전처리
from pandas.tseries.offsets import MonthEnd

df['Adjustments'] = pd.to_datetime(df['Adjustments']) + \
                    MonthEnd()
df = df.set_index('Adjustments')    # 날짜를 인덱스로 지정
print(df.head())
df.plot()
plt.show()
```

★

```
In [22]: print(df.head())
             Unadjusted  Seasonally adjusted
Adjustments
1991-01-31     12588862              15026890
1991-02-28     12154321              15304585
1991-03-31     14337072              15413591
1991-04-30     15108570              15293409
1991-05-31     17225734              15676083
```

★ 문자형(월)을 숫자형으로 전환 + 뒤에 월말일을 붙인다.


1e7
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
Unadjusted
Seasonally adjusted
1994
1999
2004
2009
2014
Adjustments

4. Keras LSTM

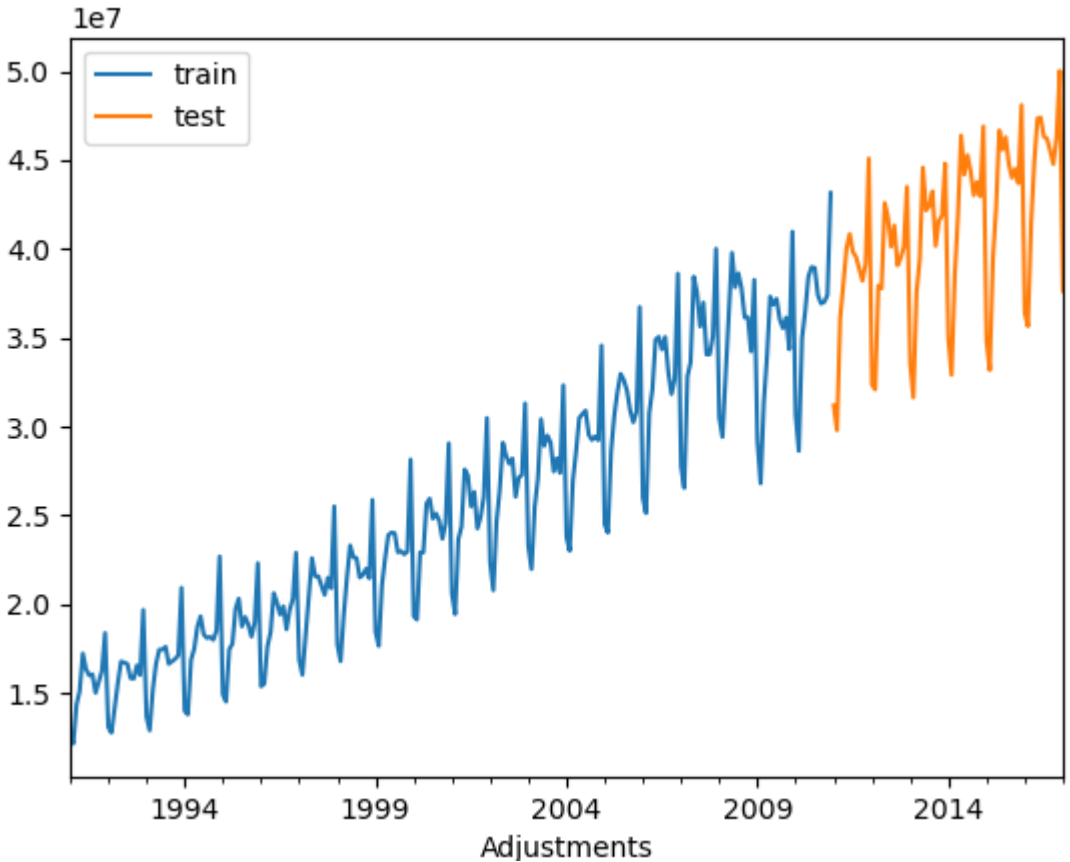
- Prepare train/test set

```
# 2011/1/1 까지의 데이터를 트레이닝셋.
# 그 이후 데이터를 테스트셋으로 한다.
# 예측할 feature는 Unadjusted

split_date = pd.Timestamp('01-01-2011')

train = df.loc[:split_date, ['Unadjusted']]
test = df.loc[split_date:, ['Unadjusted']]
```

  - Time series data는 train/test를 나누는 방법이 다르다

4. Keras LSTM

```
ax = train.plot()
test.plot(ax=ax)
plt.legend(['train', 'test'])
plt.show()
```



4. Keras LSTM

- 데이터 정규화

```
from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler()

train_sc = sc.fit_transform(train)
test_sc = sc.transform(test)

train_sc
```

```
In [29]: train_sc
Out[29]:
array([[0.01402033],
       [0.        ],
       [0.0704258 ],
       [0.09531795],
       [0.16362761],
       [0.13514108],
       [0.12395846],
       [0.12617398]
```

4. Keras LSTM

- Merge date and data

```
train_sc_df = pd.DataFrame(train_sc, columns=['Scaled'],
                           index=train.index)
test_sc_df = pd.DataFrame(test_sc, columns=['Scaled'],
                          index=test.index)
train_sc_df.head()
```

```
In [32]: train_sc_df.head()
Out[32]:
                Scaled
Adjustments
1991-01-31    0.014020
1991-02-28    0.000000
1991-03-31    0.070426
1991-04-30    0.095318
1991-05-31    0.163628
```

12개월치 매출 데이터로 다음달 매출액을 예측한다

X y

Ex) 1990.5 ~ 1991.4 매출액으로 1991.5 매출액 예측

4. Keras LSTM

- Make time series train dataset

```
for s in range(1, 13):
    train_sc_df['shift_{}'.format(s)] = \
                        train_sc_df['Scaled'].shift(s)
    test_sc_df['shift_{}'.format(s)] = \
                        test_sc_df['Scaled'].shift(s)

train_sc_df.head(13)
```

```
In [34]: train_sc_df.head(13)
Out[34]:
```

test_y

train_X

Adjustments	Scaled	shift_1	shift_2	...	shift_10	shift_11	shift_12
1991-01-31	0.014020	NaN	NaN	...	NaN	NaN	NaN
1991-02-28	0.000000	0.014020	NaN	...	NaN	NaN	NaN
1991-03-31	0.070426	0.000000	0.014020	...	NaN	NaN	NaN
1991-04-30	0.095318	0.070426	0.000000	...	NaN	NaN	NaN
1991-10-31	0.111395	0.092309	0.126174	...	NaN	NaN	NaN
1991-11-30	0.131738	0.111395	0.092309	...	0.014020	NaN	NaN
1991-12-31	0.200913	0.131738	0.111395	...	0.000000	0.01402	NaN
1992-01-31	0.030027	0.200913	0.131738	...	0.070426	0.00000	0.01402

예측

4. Keras LSTM

- NA 포함 행 제거

```
X_train = train_sc_df.dropna().drop('Scaled', axis=1)
y_train = train_sc_df.dropna()[['Scaled']]

X_test = test_sc_df.dropna().drop('Scaled', axis=1)
y_test = test_sc_df.dropna()[['Scaled']]

X_train.head()
```

```
In [36]: X_train.head()
Out[36]:
              shift_1   shift_2   shift_3  ...  shift_10  shift_11  shift_12
Adjustments                                ...
1992-01-31   0.200913  0.131738  0.111395  ...  0.070426  0.000000  0.014020
1992-02-29   0.030027  0.200913  0.131738  ...  0.095318  0.070426  0.000000
1992-03-31   0.019993  0.030027  0.200913  ...  0.163628  0.095318  0.070426
1992-04-30   0.065964  0.019993  0.030027  ...  0.135141  0.163628  0.095318
1992-05-31   0.109831  0.065964  0.019993  ...  0.123958  0.135141  0.163628
```

4. Keras LSTM

- dataframe to ndarray

```
X_train = X_train.values
X_test= X_test.values

y_train = y_train.values
y_test = y_test.values
print(X_train.shape)
print(X_train)
print(y_train.shape)
print(y_train)
```

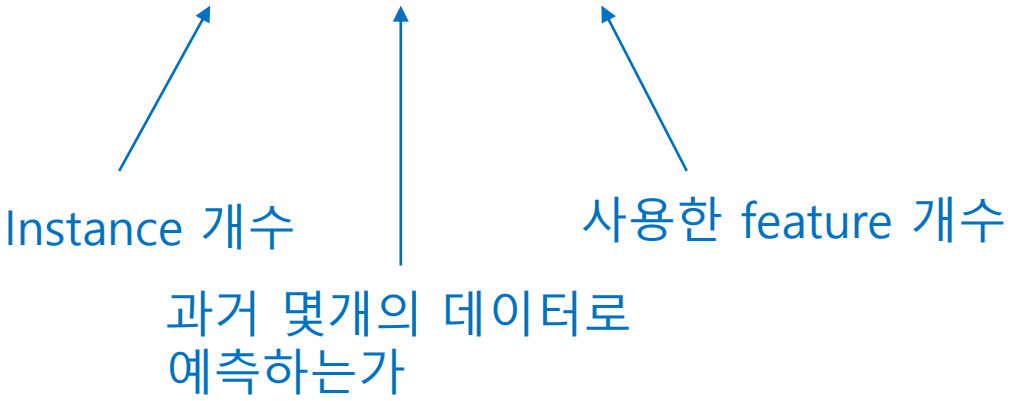
```
In [42]: print(X_train.shape)
(228, 12)

In [43]: print(X_train)
[[0.20091289 0.13173822 0.11139526 ... 0.0704258  0.         0.01402033]
 [0.03002688 0.20091289 0.13173822 ... 0.09531795 0.0704258  0.        ]
 [0.01999285 0.03002688 0.20091289 ... 0.16362761 0.09531795 0.0704258 ]
 ...
In [44]: print(y_train.shape)
(228, 1)

In [45]: print(y_train)
[[0.03002688]
 [0.01999285]
 [0.06596369]
```

4. Keras LSTM

- 최종 dataset 구성하기

```
X_train_t = X_train.reshape(X_train.shape[0], 12, 1)
X_test_t = X_test.reshape(X_test.shape[0], 12, 1)
```

  - LSTM 입력 데이터셋은 3차원 배열이어야 함
  - 3차원: (size, timestep, feature)



4. Keras LSTM

- 최종 dataset 구성하기

```
print("Final DATASET")
print(X_train_t.shape)
print(X_train_t)
print(y_train)
```

```
In [49]: print(X_train_t.shape)
(228, 12, 1)
```

```
In [50]: print(X_train_t)
[[[0.20091289]
  [0.13173822]
  [0.11139526]
  ...
  [0.0704258 ]
  [0.        ]
  [0.01402033]]

 [[0.03002688]
  [0.20091289]
```

```
In [51]: print(y_train)
[[0.03002688]
 [0.01999285]
 [0.06596369]
 [0.10983126]
 [0.14912986]
 [0.14718865]
 [0.14464787]
 [0.11898427]
```

4. Keras LSTM

- LSTM 모델 구축

```
from keras.layers import LSTM
from keras.models import Sequential
from keras.layers import Dense
import keras.backend as K
from keras.callbacks import EarlyStopping

K.clear_session()
model = Sequential()                          # Sequential Model
model.add(LSTM(20, input_shape=(12, 1))) # (timestep, feature)
model.add(Dense(1))                           # output 개수 = 1
model.compile(loss='mean_squared_error', optimizer='adam')

model.summary()
```

clear_session() 함수는 현재 세션에 존재하는 모든 텐서 객체들을 삭제
이 함수는 메모리 누수(memory leak)를 방지하고, 모델을 여러 번 훈련할 때 이전에 사용되었던 가중치(weight) 등을 초기화하는 데 사용

4. Keras LSTM

```
In [67]: model.summary()
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 20)                1760

 dense (Dense)               (None, 1)                 21

=================================================================
Total params: 1,781
Trainable params: 1,781
Non-trainable params: 0
_________________________________________________________________
```

4. Keras LSTM

```
model.fit(X_train_t, y_train, epochs=100,
          batch_size=30, verbose=1)
```

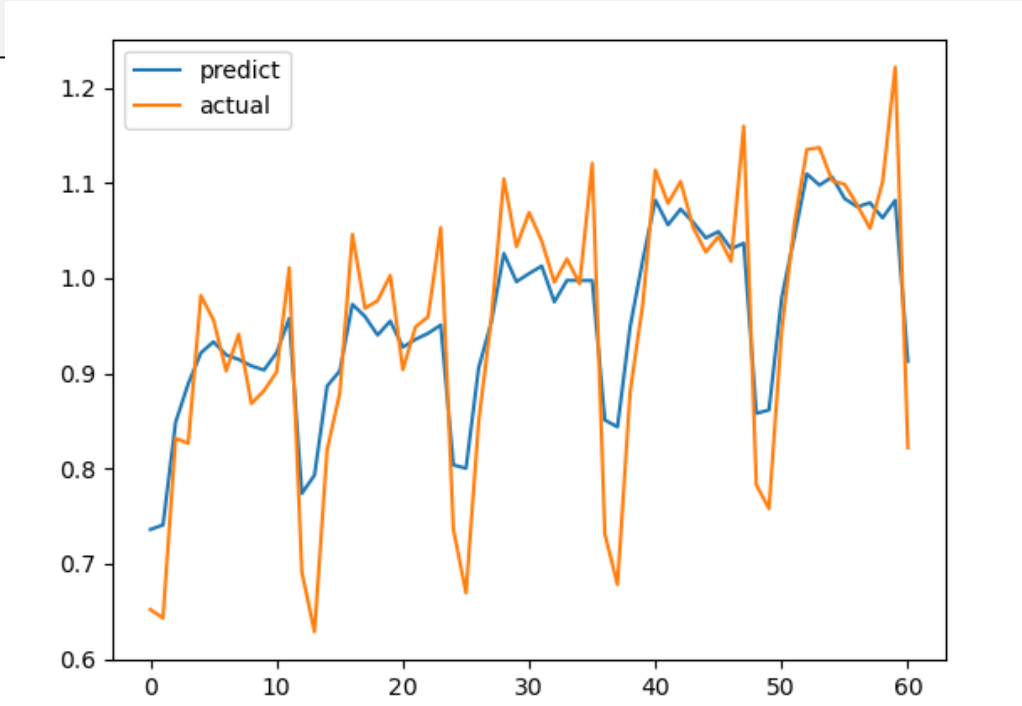
```
In [68]: model.fit(X_train_t, y_train, epochs=100,
    ...:           batch_size=30, verbose=1)
Epoch 1/100
8/8 [==============================] - 4s 21ms/step - loss: 0.2595
Epoch 2/100
8/8 [==============================] - 0s 3ms/step - loss: 0.1419
Epoch 3/100
8/8 [==============================] - 0s 4ms/step - loss: 0.0589
Epoch 4/100
8/8 [==============================] - 0s 4ms/step - loss: 0.0231
```

```
8/8 [==============================] - 0s 3ms/step - loss: 0.0036
Epoch 99/100
8/8 [==============================] - 0s 3ms/step - loss: 0.0035
Epoch 100/100
8/8 [==============================] - 0s 3ms/step - loss: 0.0034
Out[68]: <keras.callbacks.History at 0x23d65db2250>
```

4. Keras LSTM

- 모델 테스트

```
y_pred = model.predict(X_test_t)
print(y_pred)

plt.figure()
plt.plot(y_pred)
plt.plot(y_test)
plt.legend(['predict', 'actual'])
plt.show()
```



4. Keras LSTM

```
from sklearn.metrics import mean_squared_error
print('Mean squared error: {0:.2f}'.\
      format(mean_squared_error(y_test, y_pred)))
```

```
In [6]: from sklearn.metrics import mean_squared_error
   ...: print('Mean squared error: {0:.2f}'.\
   ...:       format(mean_squared_error(y_test, y_pred)))
Mean squared error: 0.01
```

[Practice 1]

- 이번 단원에서 제공된 LSTM 코드를 이용하여 다음의 결과를 제출하시오
- 1) epoch=100 인경우와 epoch=200 인 경우 MSE 값을 비교하시오
- 2) epoch=100 일 때 time step을 6으로 했을 때와 12로 했을때 MSE 값을 비교하시오.

[Practice 2]

- DJIA 30 Stock Time Series dataset link
  - Historical stock data for DIJA 30 companies (2006-01-01 to 2018-01-01)

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Volume	Name
2	2006-01-03	39.69	41.22	38.79	40.91	24232729	AABA
3	2006-01-04	41.22	41.9	40.77	40.97	20553479	AABA
4	2006-01-05	40.93	41.73	40.85	41.53	12829610	AABA
5	2006-01-06	42.88	43.57	42.8	43.21	29422828	AABA
6	2006-01-09	43.1	43.66	42.82	43.42	16268338	AABA
7	2006-01-10	42.96	43.34	42.34	42.98	16288580	AABA
8	2006-01-11	42.19	42.31	41.72	41.87	26192772	AABA
9	2006-01-12	41.92	41.99	40.76	40.89	18921686	AABA
10	2006-01-13	41	41.08	39.62	39.9	30966185	AABA

주가

거래량

주식명 (삭제)

[Practice 2]

- 이전 30일의 종가(close) 데이터로 다음날 종가(close)를 예측하는 LSTM 모델을 생성하고 테스트 하시오
  - Trainset : 2006-01-01 ~ 2017-06-30
  - Testset : 2017-07-01 이후

[Practice 3] 도전과제

- **Practice 2**에서 종가(close)를 예측할 때 이전일의 open, high, low, close 변수를 모두 사용하시오