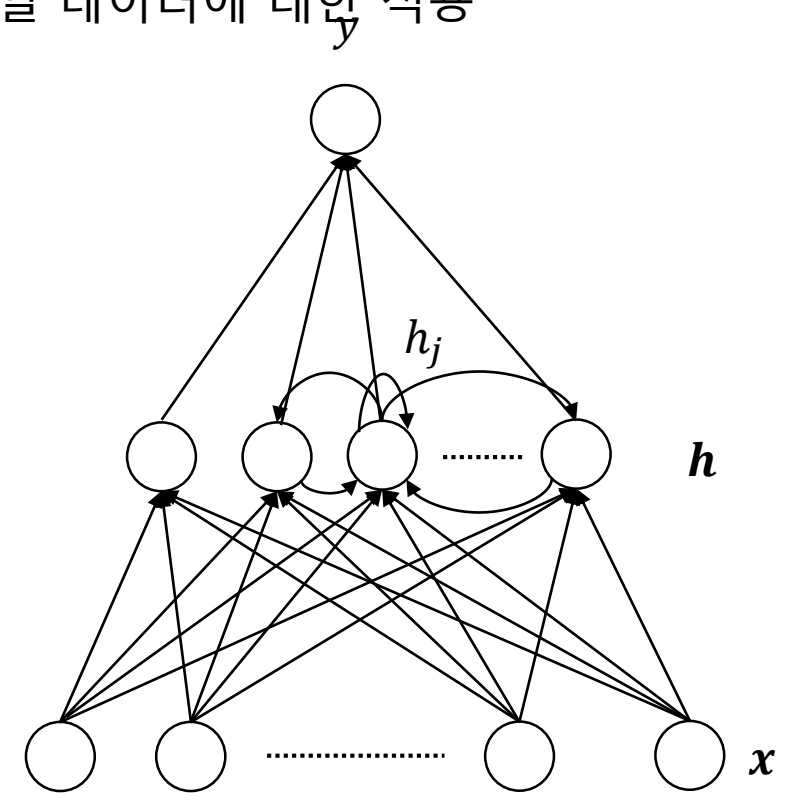
AI 실습

Contents

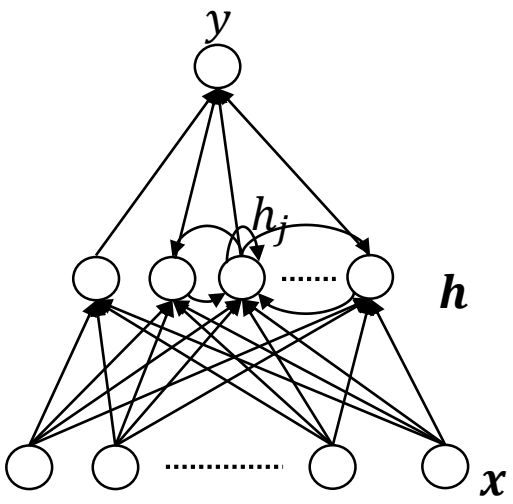
6.1. 회귀신경회로망(Recurrent Neural Networks)

시계열 데이터(time series data): 시간에 따라 변하는 데이터 (주가, 유가, 심전도, 음성..)
보편적 근사화 이론의 시계열 데이터에 대한 적용



은닉 노드의 회귀 연결(Recurrent connection for hidden nodes)

$$\hat{h_j}(t) = \sum_i w_{ji} x_i + \sum_k T_{jk} h_k(t-1) \qquad (6.1.1)$$



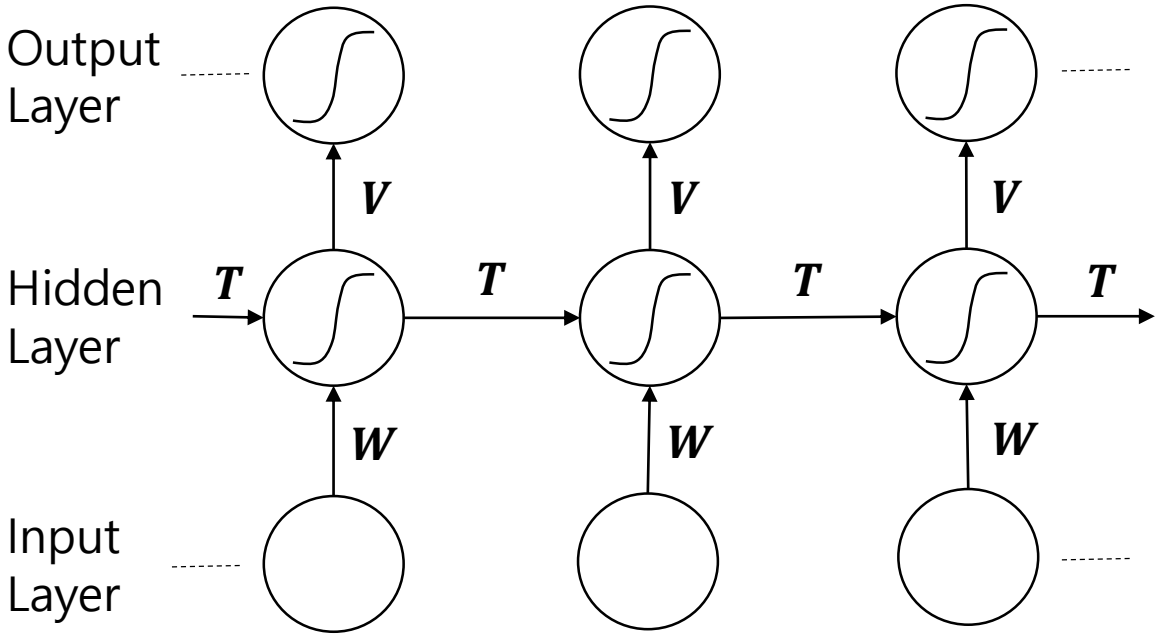


그림 6.4. 시간에 대하여 펼쳐진 은닉노드 회귀연결 신경회로망. 각 원은 특정 시간의 해당 층 노드들을 뜻함. W는 입력층에서 은닉층으로의 연결 가중치, V는 은닉층에서 출력층으로의 연결 가중치, T는 은닉층 내부의 회귀연결 가중치를 각각 뜻함.

$$\delta_j^{(hidden)}(t) = f'(h_j^{(t)})\left(\sum_{k=1}^{M} v_{kj}\delta_k + \sum_{k=1}^{H} T_{jk}\delta_k^{(hidden)}(t+1)\right) \qquad (6.1.4)$$

6.2. Long Short-Term Memory(LSTM)

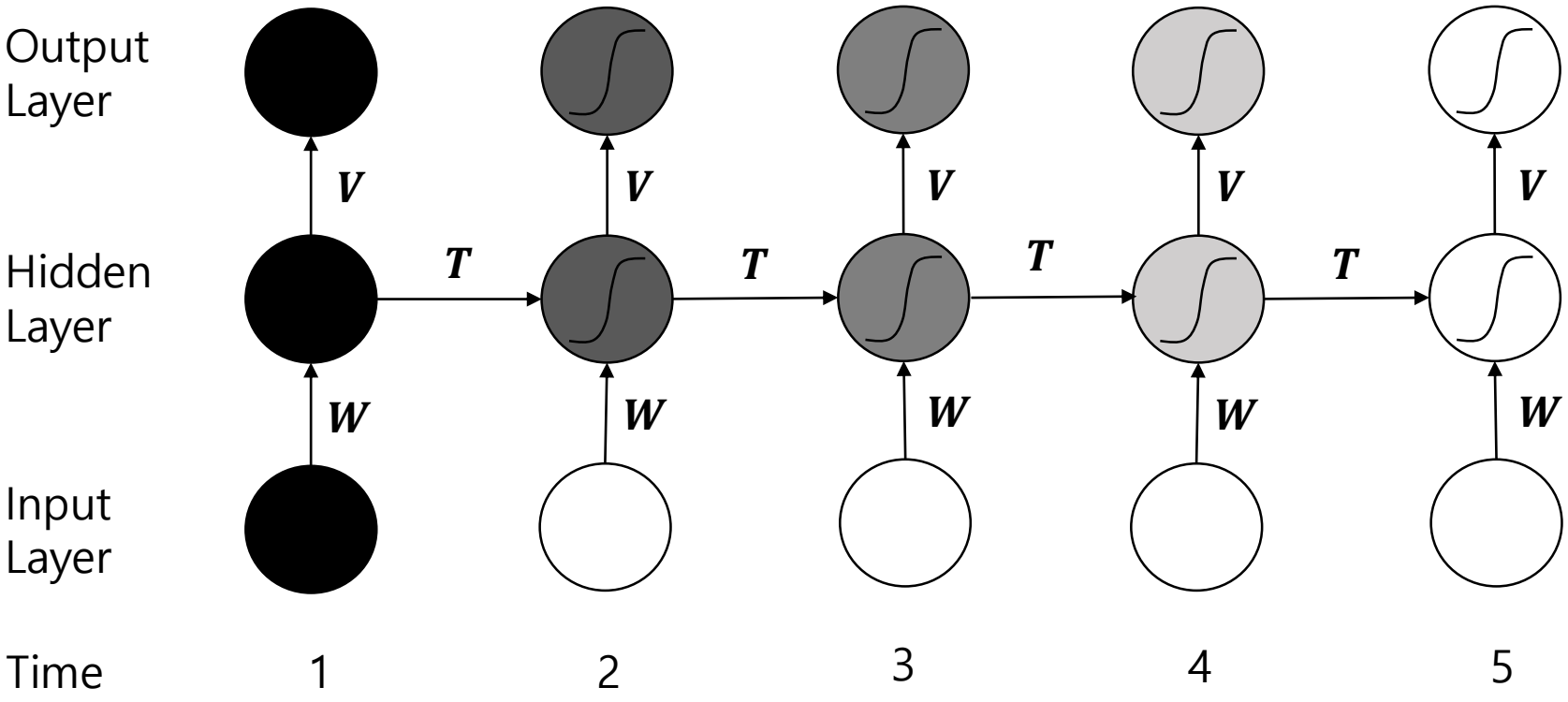


그림 6.5. 회귀신경회로망에서 시간에 따른 영향의 감소. 색깔의 강도는 민감도의 강도를 나타냄.

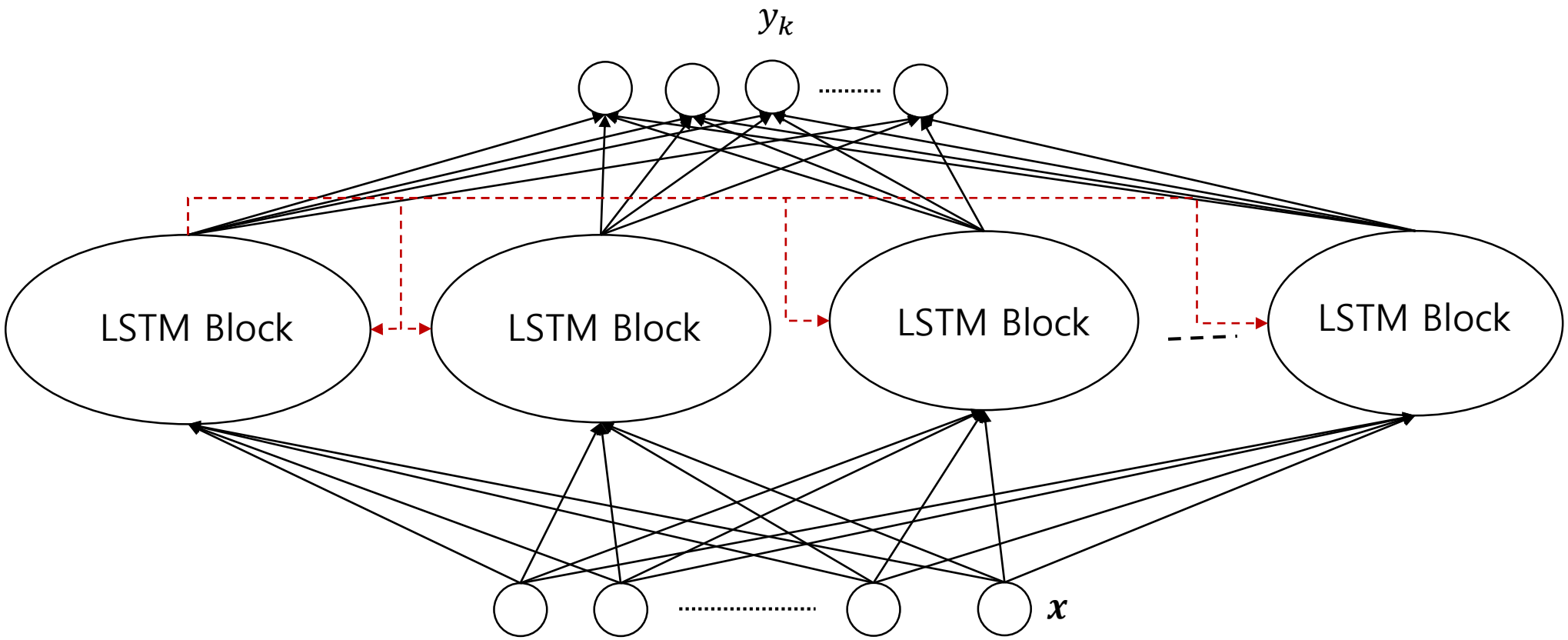
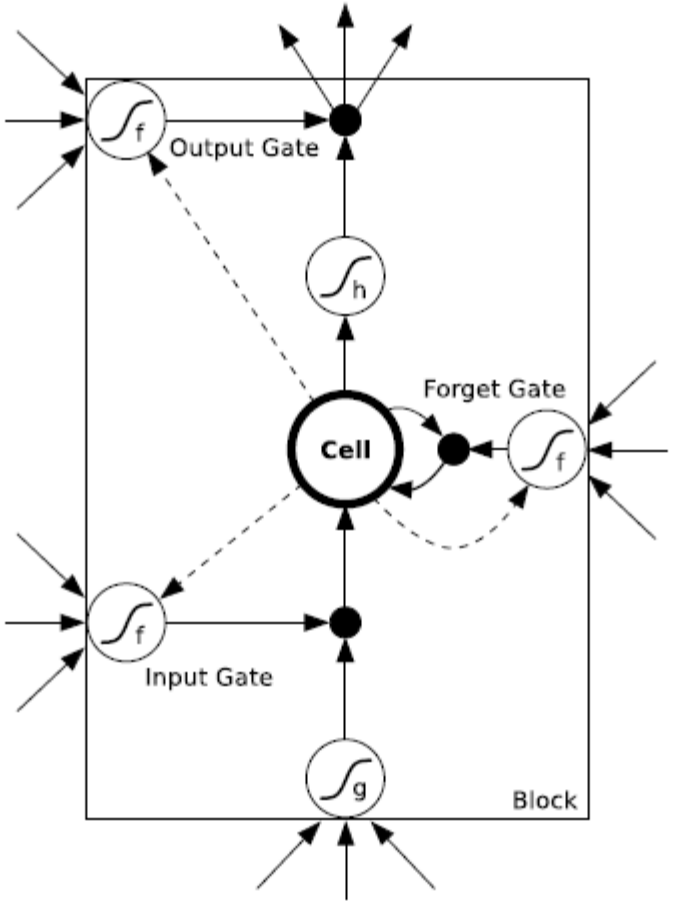


그림 6.6. LSTM 블록을 지닌 회귀 신경회로망. LSTM 블록 사이의 연결은 한 방향 만을 표현하였지만, 모든 LSTM 블록들은 서로 연결되어 있음.

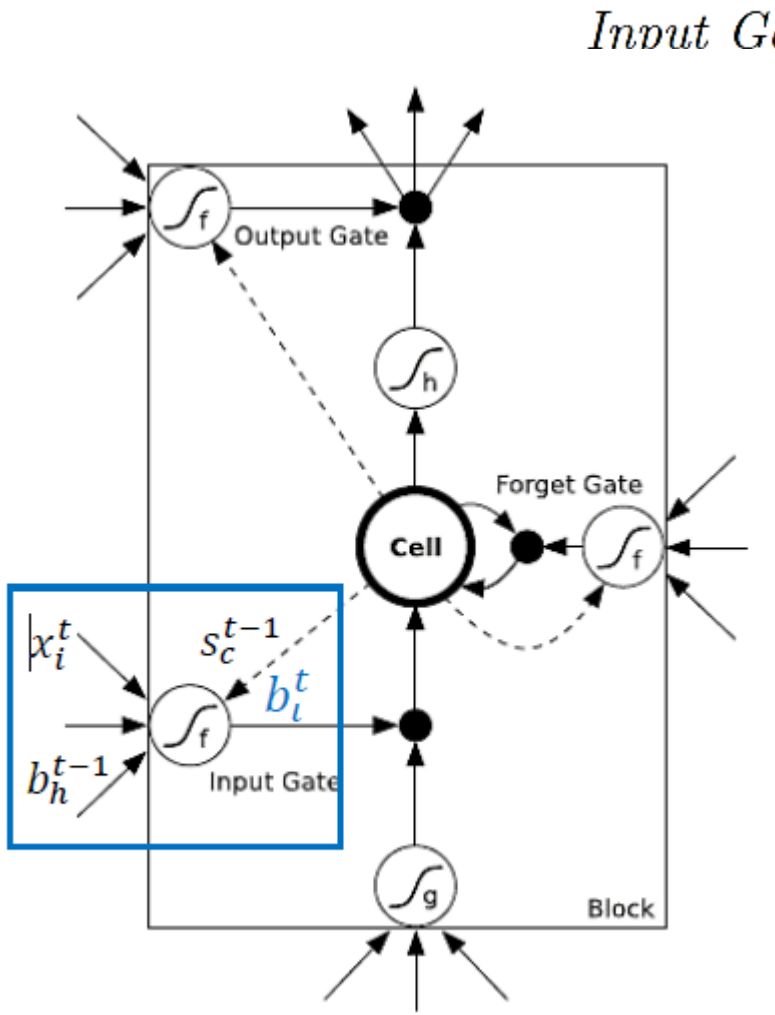
Long Short-term Memory (LSTM, Hochreiterand Schmidhuber, 1997)

Consists of a set of recurrently connected subnets, known as **memory blocks**
Each block contains one or more self-connected memory **cells** and three multiplicative units –the **input, output, and forget gates.**
The three gates control the activation of the cell via multiplications



f: activation function of the gates
g: cell input activation function
h: cell output activation function

6.3. Forward Pass

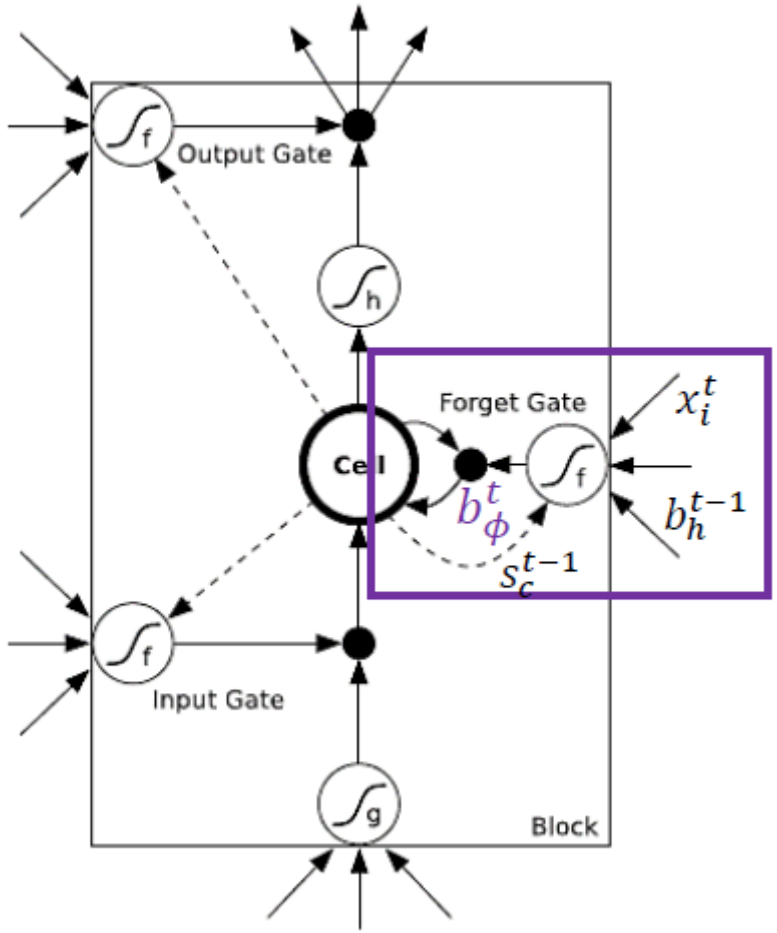


$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1} \qquad (6.3.1)$$

$$b_\iota^t = f(a_\iota^t)$$

I : # inputs
K : # outputs
H : # cells in the hidden layer
C : # cells per a memory
G : total # inputs to the hidden layer ($= 4\mathrm{H}$)

Forget Gates

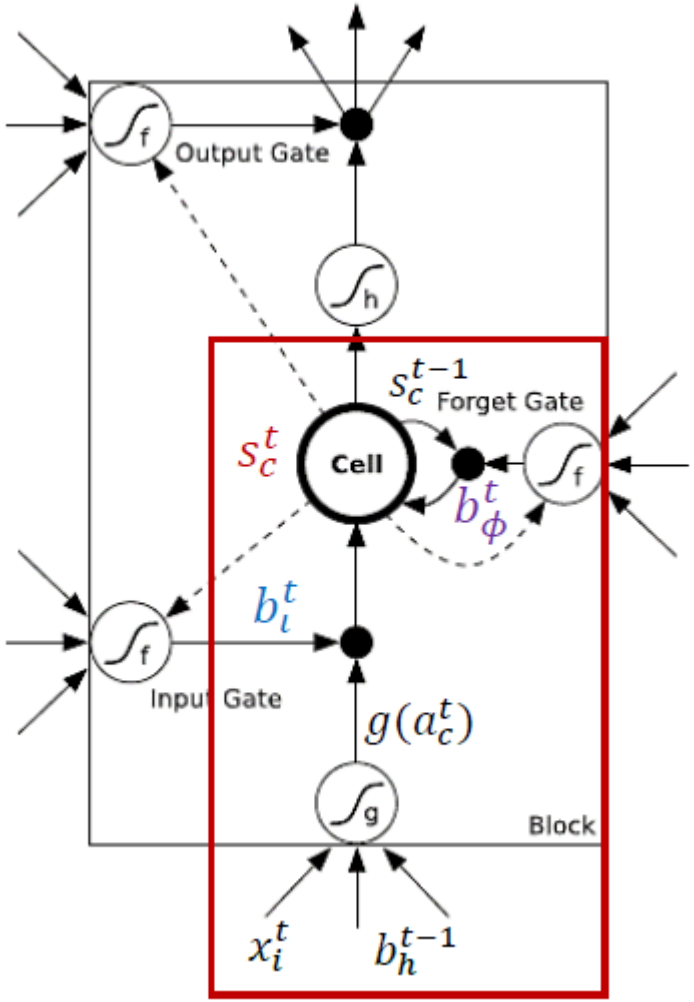


$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1} \qquad (6.3.3)$$

$$b_\phi^t = f(a_\phi^t)$$

I : # inputs
K : # outputs
H : # cells in the hidden layer
C : # cells per a memory
G : total # inputs to the hidden layer ($= 4H$)

Cells

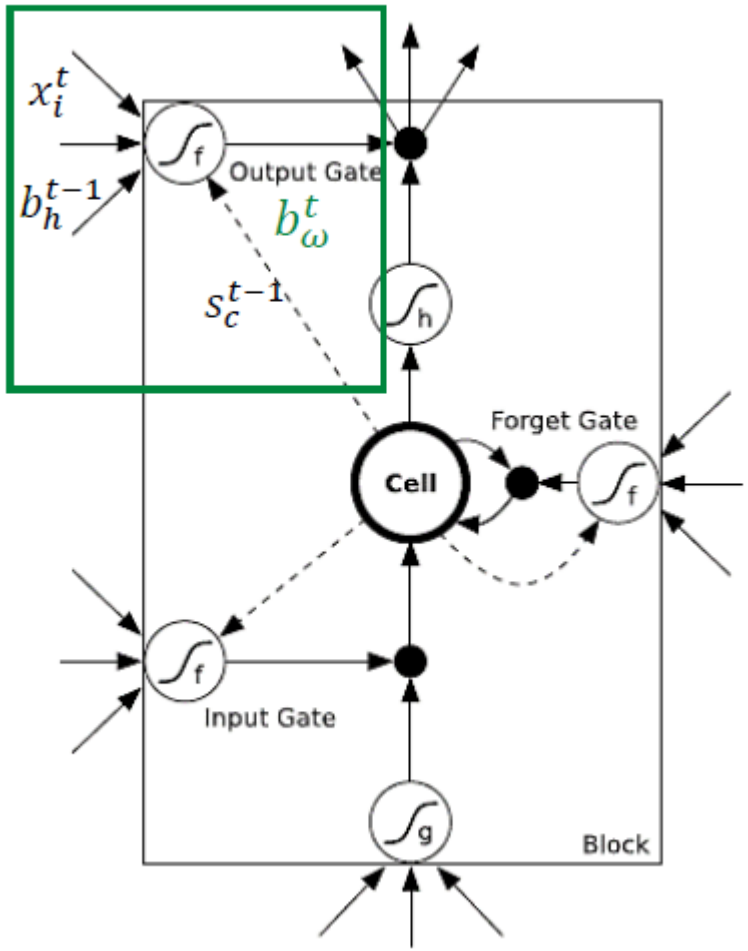


$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1} \qquad (6.3.5)$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t) \qquad (6.3.6)$$

I : # inputs
K : # outputs
H : # cells in the hidden layer
C : # cells per a memory
G : total # inputs to the hidden layer ($= 4\mathrm{H}$)

Output Gates

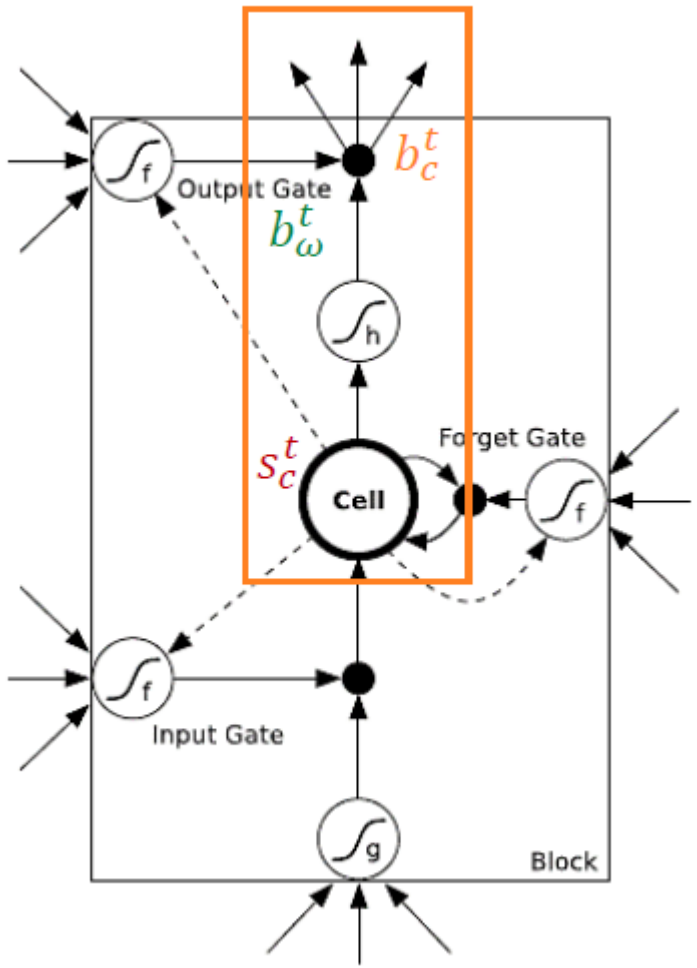


$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^t \qquad (6.3.7)$$

$$b_\omega^t = f(a_\omega^t)$$

I : # inputs
K : # outputs
H : # cells in the hidden layer
C : # cells per a memory
G : total # inputs to the hidden layer ($= 4\mathrm{H}$)

Cell Outputs



$$b_c^t = b_\omega^t h(s_c^t)$$

I : # inputs
K : # outputs
H : # cells in the hidden layer
C : # cells per a memory
G : total # inputs to the hidden layer $(= 4\mathrm{H})$

6.4. Backward Pass

Loss function: L

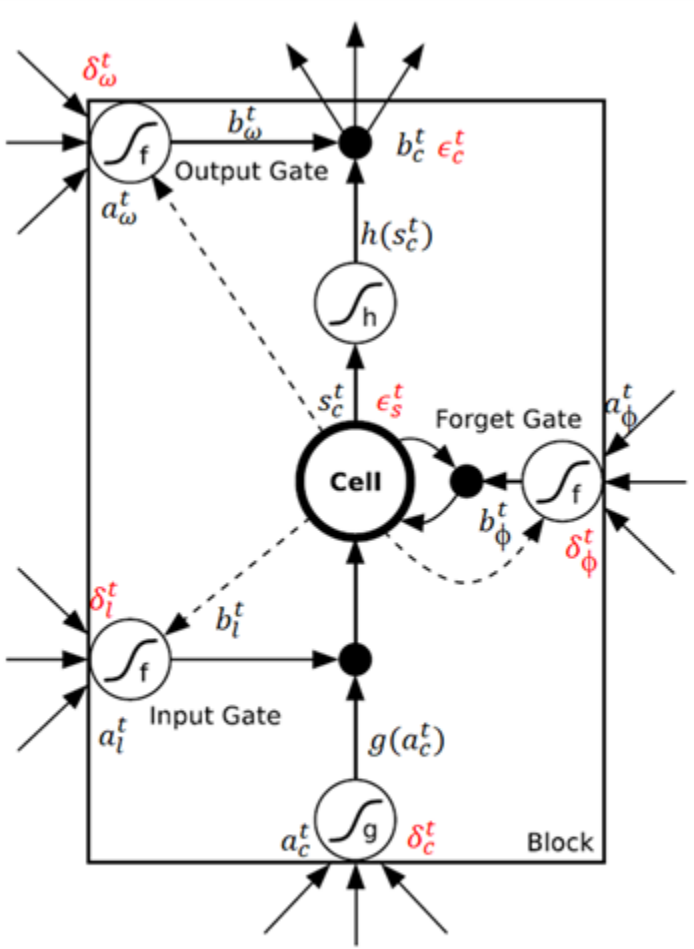


그림 6.12: 전방향 및 역방향 변수

Cell Outputs

$$\epsilon_c^t \equiv \frac{\partial L}{\partial b_c^t} = \sum_{k=1}^{K} w_{ck}\delta_k^t + \sum_{g=1}^{G} w_{cg}\delta_g^{t+1} \tag{6.4.1}$$

$$a_l^t = \sum_{i=1}^{I} w_{il}x_i^t + \sum_{h=1}^{H} w_{hl}b_h^{t-1} + \sum_{c=1}^{C} w_{cl}s_c^{t-1} \tag{6.3.1}$$

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi}x_i^t + \sum_{h=1}^{H} w_{h\phi}b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi}s_c^{t-1} \tag{6.3.3}$$

$$a_c^t = \sum_{i=1}^{I} w_{ic}x_i^t + \sum_{h=1}^{H} w_{hc}b_h^{t-1} \tag{6.3.5}$$

$$a_w^t = \sum_{i=1}^{I} w_{iw}x_i^t + \sum_{h=1}^{H} w_{hw}b_h^{t-1} + \sum_{c=1}^{C} w_{cw}s_c^t \tag{6.3.7}$$

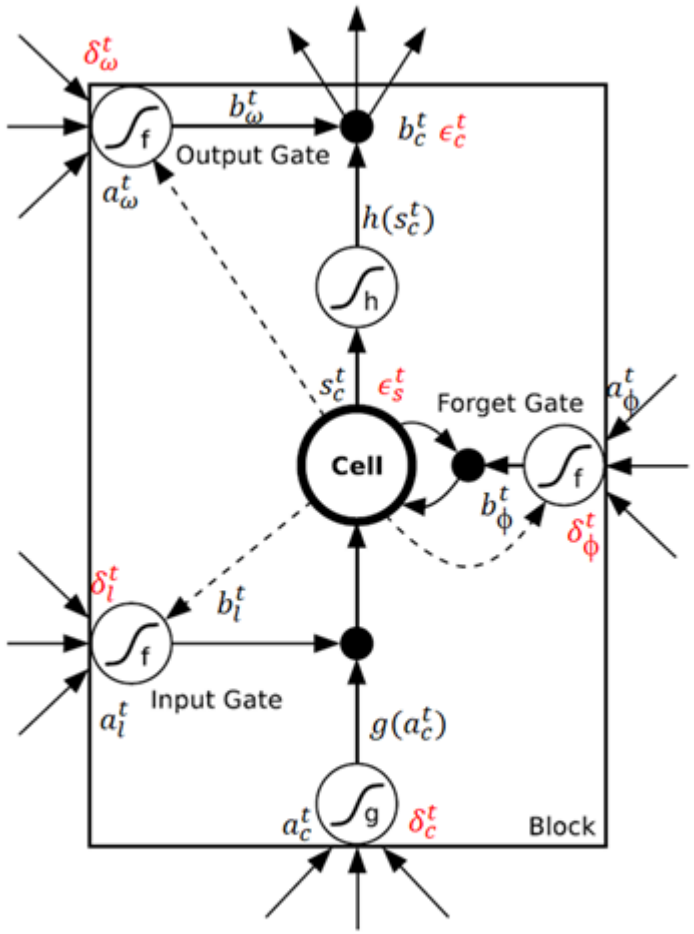


그림 6.12. 전방향 및 역방향 변수

Output Gates

$$\delta_w^t \equiv \frac{\partial L}{\partial a_w^t} = f'(a_w^t) \sum_{c=1}^{C} h(s_c^t)\epsilon_c^t \qquad (6.4.2)$$

States

$$\epsilon_s^t \equiv \frac{\partial L}{\partial s_c^t} = b_w^t h'(s_c^t)\epsilon_c^t + b_\phi^{t+1}\epsilon_s^{t+1} + w_{cl}\delta_l^{t+1} + w_{c\phi}\delta_\phi^{t+1} + w_{cw}\delta_w^t \qquad (6.4.3)$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_l^t g(a_c^t) \qquad (6.3.6)$$

$$a_l^t = \sum_{i=1}^{I} w_{il}x_i^t + \sum_{h=1}^{H} w_{hl}b_h^{t-1} + \sum_{c=1}^{C} w_{cl}s_c^{t-1} \qquad (6.3.1)$$

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi}x_i^t + \sum_{h=1}^{H} w_{h\phi}b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi}s_c^{t-1} \qquad (6.3.3)$$

$$a_w^t = \sum_{i=1}^{I} w_{iw}x_i^t + \sum_{h=1}^{H} w_{hw}b_h^{t-1} + \sum_{c=1}^{C} w_{cw}s_c^t \qquad (6.3.7)$$

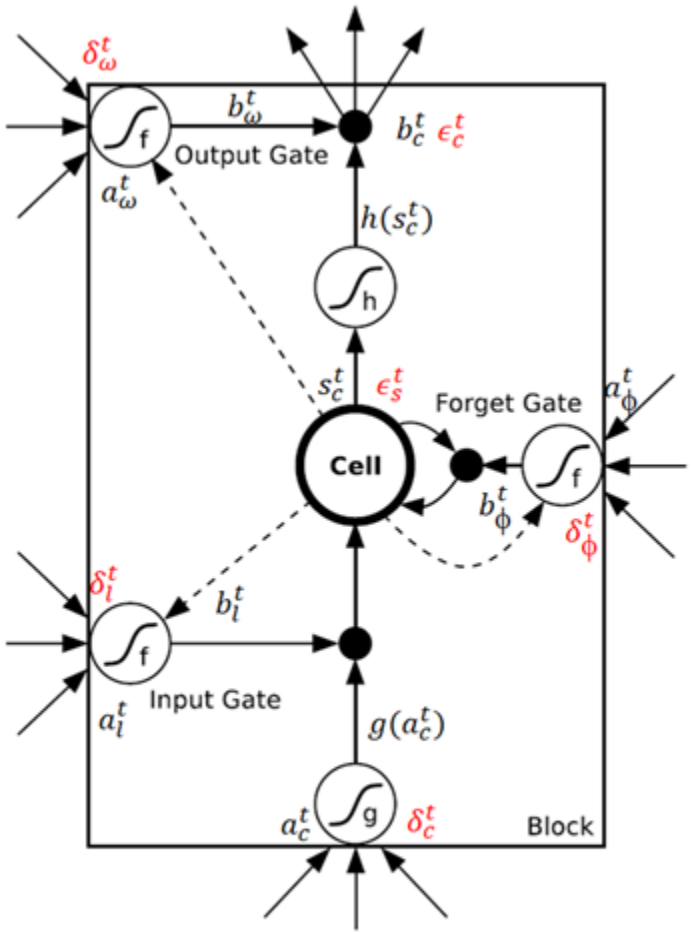


그림 6.12: 전방향 및 역방향 변수

Cells

$$\delta_c^t \equiv \frac{\partial L}{\partial a_c^t} = b_l^t g'(a_c^t)\epsilon_s^t \tag{6.4.4}$$

Forget Gates

$$\delta_\phi^t \equiv \frac{\partial L}{\partial a_\phi^t} = f'(a_\phi^t)\sum_{c=1}^{C} s_c^{t-1}\epsilon_s^t \tag{6.4.5}$$

Input Gates

$$\delta_l^t \equiv \frac{\partial L}{\partial a_l^t} = f'(a_l^t)\sum_{c=1}^{C} g(a_c^t)\epsilon_s^t \tag{6.4.6}$$

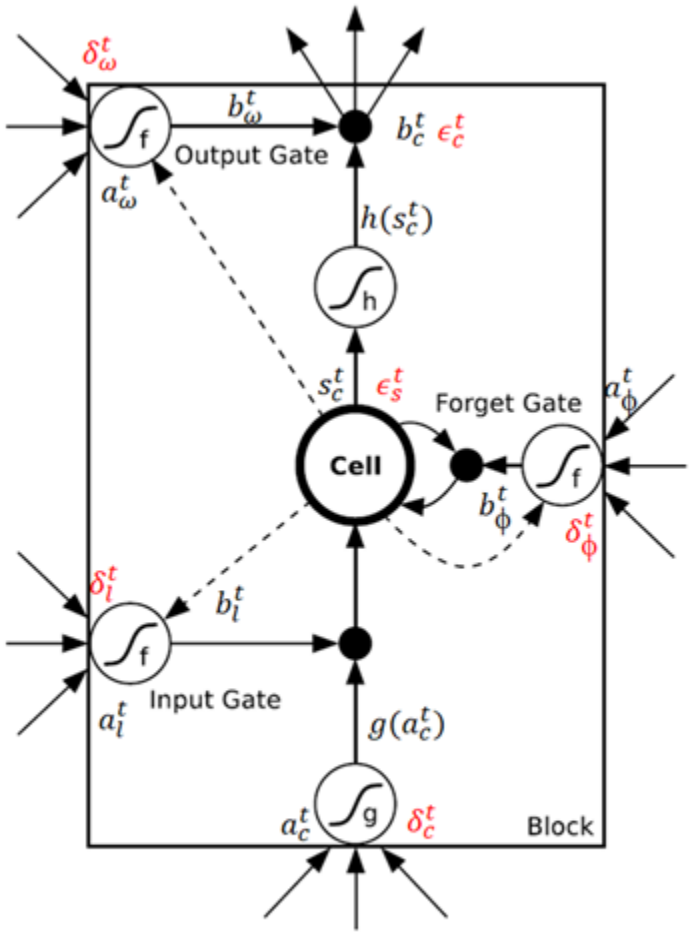


그림 6.12: 전방향 및 역방향 변수

$$\epsilon_c^t \stackrel{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial b_c^t} \qquad \epsilon_s^t \stackrel{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial s_c^t}$$

Cell Outputs

$$\epsilon_c^t = \sum_{k=1}^{K} w_{ck}\delta_k^t + \sum_{g=1}^{G} w_{cg}\delta_g^{t+1}$$

Output Gates

$$\delta_\omega^t = f'(a_\omega^t)\sum_{c=1}^{C} h(s_c^t)\epsilon_c^t$$

States

$$\epsilon_s^t = b_\omega^t h'(s_c^t)\epsilon_c^t + b_\phi^{t+1}\epsilon_s^{t+1} + w_{c\iota}\delta_\iota^{t+1} + w_{c\phi}\delta_\phi^{t+1} + w_{c\omega}\delta_\omega^t$$

Cells

$$\delta_c^t = b_\iota^t g'(a_c^t)\epsilon_s^t$$

Forget Gates

$$\delta_\phi^t = f'(a_\phi^t)\sum_{c=1}^{C} s_c^{t-1}\epsilon_s^t$$

Input Gates

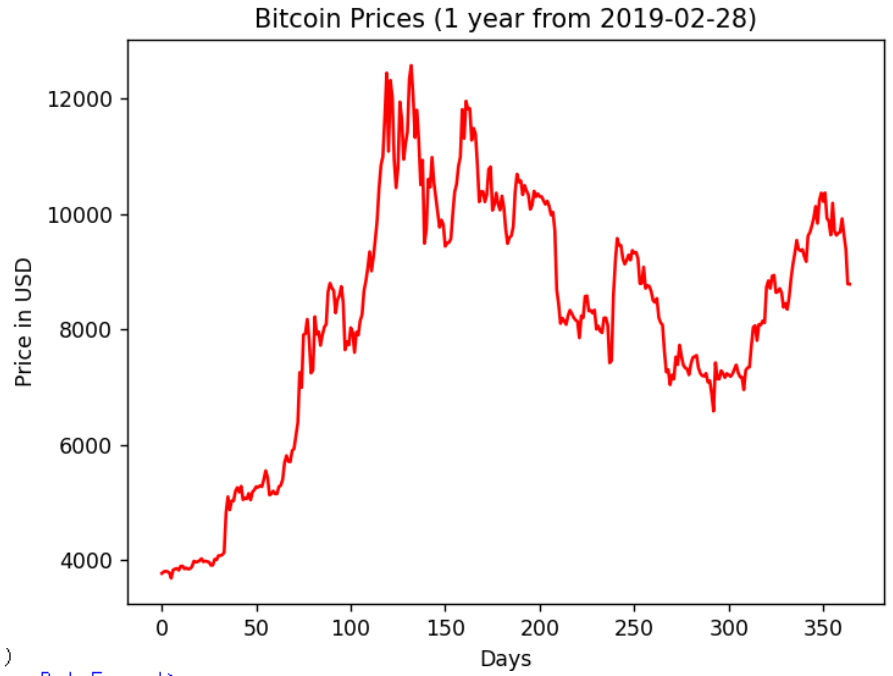
$$\delta_\iota^t = f'(a_\iota^t)\sum_{c=1}^{C} g(a_c^t)\epsilon_s^t$$

실습

8-1.py

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

# 코인데스크 사이트에서 다운로드한 1년치 비트코인 가격 데이터 읽기
f=open('BTC_USD_2019-02-28_2020-02-27-CoinDesk.csv','r')
coindesk_data=pd.read_csv(f,header=0)
seq=coindesk_data[['Closing Price (USD)']].to_numpy() # 종가만 취함
print('데이터 길이:',len(seq),'₩n앞쪽 5개 값:',seq[0:5])

# 그래프로 데이터 확인
plt.plot(seq,color='red')
plt.title('Bitcoin Prices (1 year from 2019-02-28)')
plt.xlabel('Days');plt.ylabel('Price in USD')
plt.show()

# 시계열 데이터를 윈도우 단위로 자르는 함수
def seq2dataset(seq,window,horizon):
    X=[]; Y=[]
    for i in range(len(seq)-(window+horizon)+1):
        x=seq[i:(i+window)]
        y=(seq[i+window+horizon-1])
        X.append(x); Y.append(y)
    return np.array(X), np.array(Y)

w=7 # 윈도우 크기
h=1 # 수평선 계수

X,Y = seq2dataset(seq,w,h)
print(X.shape,Y.shape)
print(X[0],Y[0]); print(X[-1],Y[-1])
```

```
데이터 길이: 365
앞쪽 5개 값: [[3772.93633533]
 [3799.67854295]
 [3811.61197937]
 [3804.41917011]
 [3782.66410112]]
(358, 7, 1) (358, 1)
[[3772.93633533]
 [3799.67854295]
 [3811.61197937]
 [3804.41917011]
 [3782.66410112]
 [3689.86289319]
 [3832.08088473]] [3848.95636968]
[[9631.48494596]
 [9670.85865437]
 [9689.08674285]
 [9919.55144784]
 [9640.46950506]
 [9392.86962872]
 [8787.97836316]] [8784.99535244]
```




```
>>> coindesk_data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 365 entries, 0 to 364
Data columns (total 6 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Currency              365 non-null    object
 1   Date                  365 non-null    object
 2   Closing Price (USD)   365 non-null    float64
 3   24h Open (USD)        365 non-null    float64
 4   24h High (USD)        365 non-null    float64
 5   24h Low (USD)         365 non-null    float64
dtypes: float64(4), object(2)
memory usage: 17.2+ KB
```

실습

8-2.py

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

# 코인데스크 사이트에서 1년치 비트코인 가격 데이터 읽기
f=open("BTC_USD_2019-02-28_2020-02-27-CoinDesk.csv","r")
coindesk_data=pd.read_csv(f,header=0)
seq=coindesk_data[['Closing Price (USD)']].to_numpy() # 종가만 취함

# 시계열 데이터를 윈도우 단위로 자르는 함수
def seq2dataset(seq,window,horizon):
    X=[]; Y=[]
    for i in range(len(seq)-(window+horizon)+1):
        x=seq[i:(i+window)]
        y=(seq[i+window+horizon-1])
        X.append(x); Y.append(y)
    return np.array(X), np.array(Y)

w=7 # 윈도우 크기
h=1 # 수평선 계수

X,Y=seq2dataset(seq,w,h)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout

# 훈련 집합과 테스트 집합으로 분할
split=int(len(X)*0.7)
x_train=X[0:split]; y_train=Y[0:split]
x_test=X[split:]; y_test=Y[split:]

# LSTM 모델 설계와 학습
model=Sequential()
model.add(LSTM(units=128,activation='relu',input_shape=x_train[0].shape))
model.add(Dense(1))
model.compile(loss='mae',optimizer='adam',metrics=['mae'])
hist=model.fit(x_train,y_train,epochs=200,batch_size=1,validation_data=(x_test,y_test),verbose=2)
```

실습

8-2.py

```
# LSTM 모델 평가
ev=model.evaluate(x_test,y_test,verbose=0)
print("손실 함수:",ev[0],"MAE:",ev[1])

# LSTM 모델로 예측 수행
pred=model.predict(x_test)
print("평균절댓값백분율오차(MAPE):",sum(abs(y_test-pred)/y_test)/len(x_test))

# 학습 곡선
plt.plot(hist.history['mae'])
plt.plot(hist.history['val_mae'])
plt.title('Model mae')
plt.ylabel('mae')
plt.xlabel('Epoch')
plt.ylim([120,800])
plt.legend(['Train','Validation'], loc='best')
plt.grid()
plt.show()

# 예측 결과 시각화
x_range=range(len(y_test))
plt.plot(x_range,y_test[x_range], color='red')
plt.plot(x_range,pred[x_range], color='blue')
plt.legend(['True prices','Predicted prices'], loc='best')
plt.grid()
plt.show()

# 일부 구간을 확대하여 시각화
x_range=range(50,64)
plt.plot(x_range,y_test[x_range], color='red')
plt.plot(x_range,pred[x_range], color='blue')
plt.legend(['True prices','Predicted prices'], loc='best')
plt.grid()
plt.show()
```

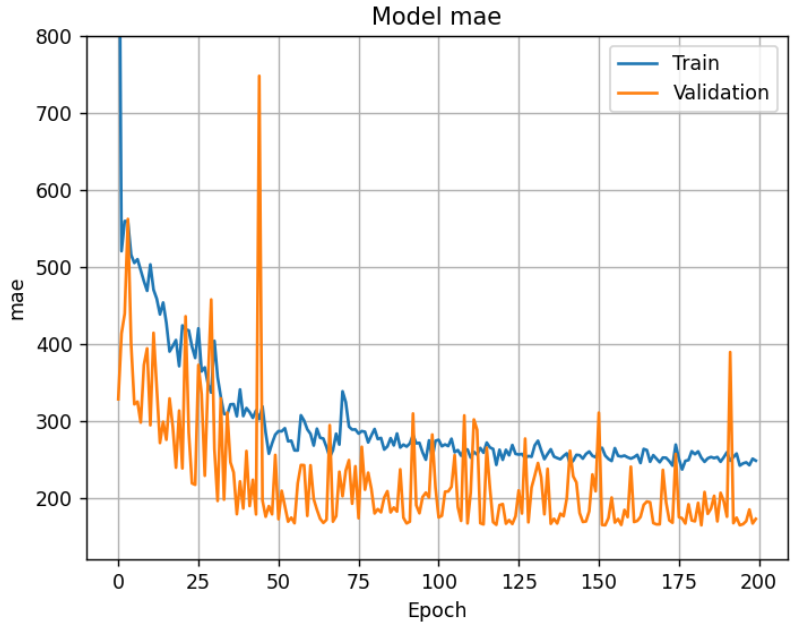
실습

```
손실 함수: 172.44252014160156 MAE: 172.44252014160156
♪1/4 [======>.....................] - ETA: 0s◆◆◆◆◆◆◆
============] - 0s 5ms/step
평균절댓값백분율오차(MAPE): [0.02086973]
```

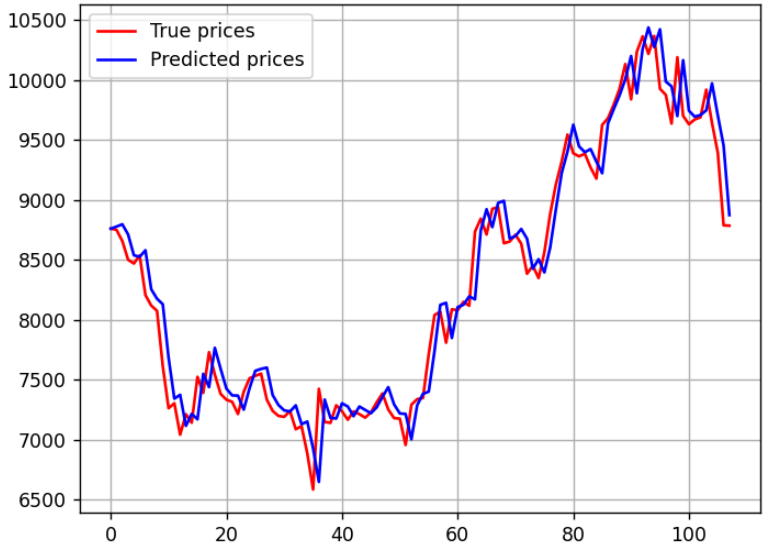
$$MAE = \frac{\sum |y - \hat{y}|}{n}$$

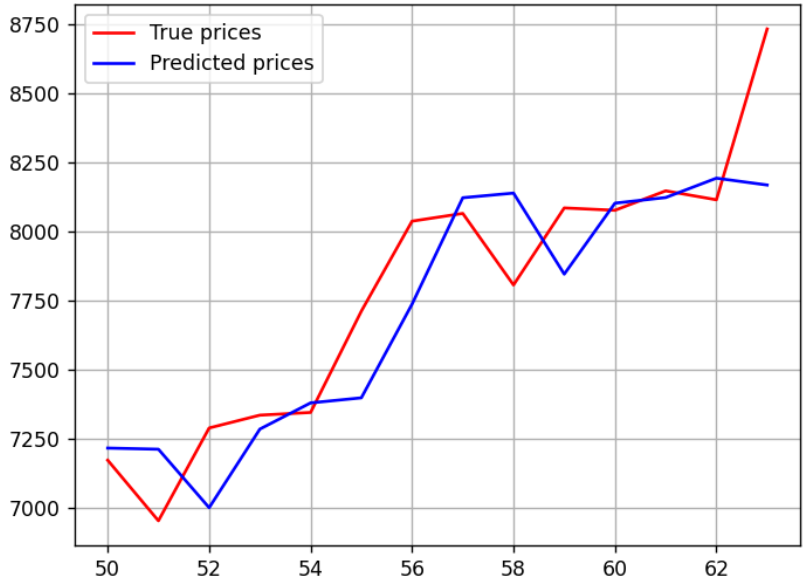
$$MAPE = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{y - \hat{y}}{y} \right|$$

?..18-2.py는 100 없는 계산이네...



8-2.py





실습

8-3.py

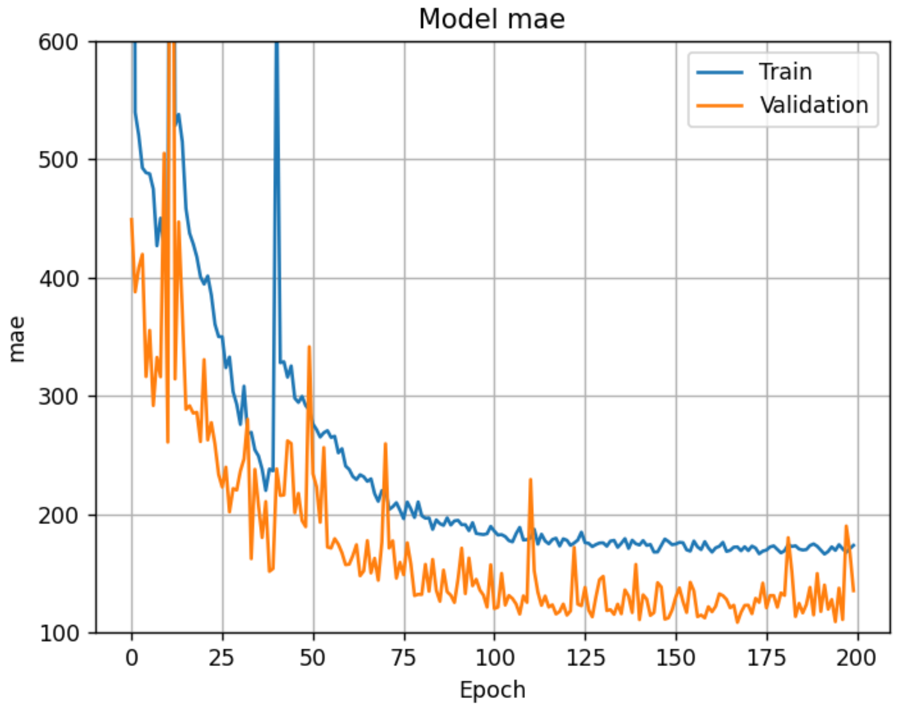
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 코인데스크 사이트에서 1년치 비트코인 가격 데이터 읽기
f=open("BTC_USD_2019-02-28_2020-02-27-CoinDesk.csv","r")
coindesk_data=pd.read_csv(f,header=0)
seq=coindesk_data[['Closing Price (USD)','24h Open (USD)','24h High (USD)','24h Low (USD)']].to_numpy()

# 시계열 데이터를 윈도우 단위로 자르는 함수
def seq2dataset(seq,window,horizon):
    X=[]; Y=[]
    for i in range(len(seq)-(window+horizon)+1):
        x=seq[i:(i+window)]
        y=(seq[i+window+horizon-1])
        X.append(x); Y.append(y)
    return np.array(X), np.array(Y)

w=7 # 윈도우 크기
h=1 # 수평선 계수

X,Y = seq2dataset(seq,w,h)
print(X.shape,Y.shape)
print(X[0],Y[0])

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout

# 훈련 집합과 테스트 집합으로 분할
split=int(len(X)*0.7)
x_train=X[0:split]; y_train=Y[0:split]
x_test=X[split:]; y_test=Y[split:]
```

```
============= RESTART: C:₩MyDocuments₩강의자료₩AI실습₩파이썬으로만드는인공지능₩8-3.py ============
(358, 7, 4) (358, 4)
[[3772.93633533 3796.63728431 3824.16587937 3666.52401643]
 [3799.67854295 3773.44146075 3879.23118467 3753.80002246]
 [3811.61197937 3799.36702601 3840.04482307 3788.91849833]
 [3804.41917011 3806.69151279 3819.19435612 3759.40921647]
 [3782.66410112 3807.84575592 3818.69548135 3766.24204823]
 [3689.86289319 3783.35506344 3804.35361623 3663.47774336]
 [3832.08088473 3701.04987103 3866.71870424 3688.69715385]] [3848.95636968 3832.59242908 3881.96576977 3802.51605364]
```

실습

8-3.py

```
# LSTM 모델의 설계와 학습
model = Sequential()
model.add(LSTM(units=128,activation='relu',input_shape=x_train[0].shape))
model.add(Dense(4))
model.compile(loss='mae',optimizer='adam',metrics=['mae'])
hist=model.fit(x_train,y_train,epochs=200,batch_size=1,validation_data=(x_test,y_test),verbose=2)

# LSTM 모델 평가
ev=model.evaluate(x_test,y_test,verbose=0)
print("손실 함수:",ev[0],"MAE:",ev[1])

# LSTM 모델로 예측 수행
pred=model.predict(x_test) # LSTM
print("LSTM 평균절댓값백분율오차(MAPE):",sum(abs(y_test-pred)/y_test)/len(x_test))

# 학습 곡선
plt.plot(hist.history['mae'])
plt.plot(hist.history['val_mae'])
plt.title('Model mae')
plt.ylabel('mae')
plt.xlabel('Epoch')
plt.ylim([100,600])
plt.legend(['Train','Validation'], loc='best')
plt.grid()
plt.show()
```




```
Epoch 198/200
250/250 - 1s - loss: 167.6992 - mae: 167.6992 - val_loss: 189.9554 - val_mae: 189.9554 - 1s/epoch - 5ms/step
Epoch 199/200
250/250 - 1s - loss: 170.8203 - mae: 170.8203 - val_loss: 167.6121 - val_mae: 167.6121 - 1s/epoch - 5ms/step
Epoch 200/200
250/250 - 1s - loss: 173.5988 - mae: 173.5988 - val_loss: 135.2576 - val_mae: 135.2576 - 1s/epoch - 5ms/step
손실 함수: 135.2576446533203 MAE: 135.2576446533203
♪1/4 [======>.......................] - ETA: 0s◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘◘♪4/4 [==============================] - 0s 5ms/step
LSTM 평균절댓값백분율오차(MAPE): [0.02279652 0.00987363 0.01640743 0.01679709]
```

8.5 자연어 처리

- 언어를 사용하는 인간
  - 다른 동물보다 지능이 월등하다는 증거
  - 시를 읽고 반응하는 인간

> 연탄재 함부로 발로 차지 마라
>
> 너는
>
> 누구에게 한 번이라도 뜨거운 사람이었느냐

- 자연어 처리natural language processing(NLP)
  - 인간이 구사하는 언어를 자동으로 처리하는 인공지능 분야
  - 다양한 응용
    - 언어 번역
    - 영화평 댓글 분석하여 흥행 추정
    - 고객 응대 챗봇
    - 소설이나 시를 쓰는 창작 인공지능 등

8.5.1 텍스트 데이터에 대한 이해

- 영화평 데이터셋인 IMDB의 예제 문장
  - 텍스트의 특성이 잘 나타남

> Once again Mr. Costner has dragged out a movie for far longer than necessary. Aside from the terrific sea rescue sequences, of which there are very few I just did not care about any of the characters. Most of us have ghosts in the closet, and Costner's character are realized early on, and then forgotten until much later, by which time I did not care. ……

- 텍스트 데이터의 특성
  - 시계열 데이터로서 시간 정보가 있고 샘플 마다 길이가 다르다는 기본 성질
  - 그 외 독특한 특성
    - 심한 잡음
    - 형태소 분석 필요
    - 구문론과 의미론
    - 다양한 언어 특성
    - 신경망에 입력하려면 기호를 수치로 변환해야 함

8.5.1 텍스트 데이터에 대한 이해

■ 원핫 코드 표현으로 변환하는 절차

▪ 예) 말뭉치$_{corpus}$ 사례

[말뭉치]

```
Freshman loves python.
We teach python to freshman.
How popular is Python?
```

- 단어 수집(괄호 속은 빈도수)
  - Python(3), freshman(2), loves(1), we(1), teach(1), to(1), how(1), popular(1), is(1)
- 파이썬의 자료구조인 딕셔너리를 이용한 표현(빈도수에 따른 순위 부여)
  - {'python':1, 'freshman':2, 'loves':1, 'we':1, 'teach':1, 'to':1, 'how':1, 'popular':1, 'is':1}
- 텍스트를 숫자 코드로 변환

```
Freshman loves python.          →     [2 3 1]
We teach python to freshman.    →     [4 5 1 6 2]
How popular is Python?          →     [7 8 9 1]
```

8.5.1 텍스트 데이터에 대한 이해

원 핫 코드 표현

```
[231]   →  [[010000000] [001000000] [100000000]]
[45162] →  [000100000] [000010000] [100000000] [000001000] [010000000]]
[7891]  →  [000000100] [000000010] [000000001] [100000000]]
```

원 핫 코드의 문제점과 해결책

- 사전 크기가 크면 원 핫 코드는 희소 벡터가 되어 메모리 낭비
- 단어 사이의 연관 관계를 반영하지 못함
  - 예, king과 queen, bridegroom과 bride의 관계 표현 불가능
- 8.5.3항에서는 현대적인 표현 기법인 단어 임베딩을 다룸

8.5.2 텍스트 데이터 사례: 영화평 데이터셋 IMDB

- 텐서플로가 제공하는 텍스트 데이터
  - 영화를 평가한 댓글을 모아둔 IMDB 데이터셋
    - 50000개의 댓글을 긍정 평가와 부정 평가로 레이블링
    - 감정 분류sentiment classification 문제에 주로 사용
  - 다양한 토픽의 뉴스를 모아둔 Reuters 데이터셋
    - 로이터 통신의 뉴스 11228개를 46개 토픽으로 레이블링
    - 토픽 분류 문제에 주로 사용

- 이 절에서는 IMDB로 프로그래밍 실습

8.5.2 텍스트 데이터 사례: 영화평 데이터셋 IMDB

- IMDB로 실습
  - http://mng.bz/0tIo에 접속하여 원본 데이터 다운로드
  - 소스 프로그램이 있는 폴더에 data라는 폴더 만들어 거기에 저장

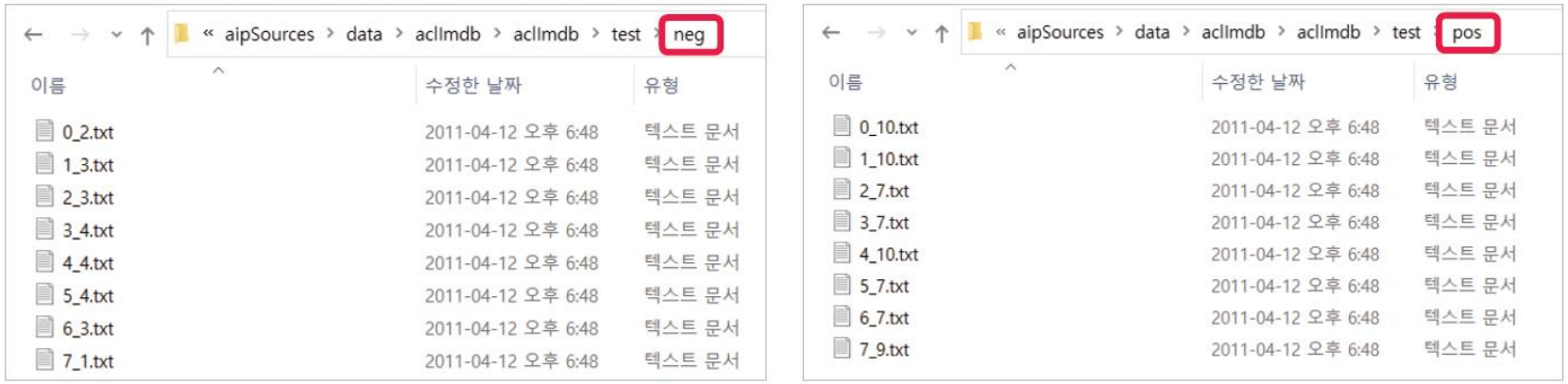


그림 8-16 IMDB 데이터셋의 폴더 구조 – neg와 pos 폴더

8.5.2 텍스트 데이터 사례: 영화평 데이터셋 IMDB

- IMDB 원본 데이터 살펴보는 [프로그램 8-6]

프로그램 8-6 IMDB 원본 텍스트 데이터 살펴보기

```
01  import os
02
03  # 원본 IMDB 데이터 읽기
04  directory='./data/aclImdb/aclImdb/test'
05  x=[];y=[]
06  for c in ['neg','pos']:
07      curr=os.path.join(directory,c)
08      for fname in os.listdir(curr):
09          if fname[-4:]=='.txt':
10              f=open(os.path.join(curr,fname),encoding='utf8')
11              x.append(f.read())
12              y.append(c)
13
14  print("첫 번째 댓글:",x[0]); print("첫 번째 평가:",y[0])
15  print("마지막 댓글:",x[-1]); print("마지막 평가:",y[-1])
```

'neg'와 'pos' 폴더 각각에서 데이터를 읽어옴

8.5.2 텍스트 데이터 사례: 영화평 데이터셋 IMDB

```
첫 번째 댓글: Once again Mr. Costner has dragged out a movie for far longer than
necessary. Aside from the terrific sea rescue sequences, of which there are very
few I just did not care about any of the characters. Most of us have ghosts in the
closet, and Costner's character are realized early on, and then forgotten until
much later, by which time I did not care. …
첫 번째 평가: neg

마지막 댓글: When I saw this movie for the first time I was both surprised and a
little shocked by the blatant vibrance of the story. It is a very artistic drama
with incredible special effects, spectacular acting, not to mention a very
excellent job in the makeup department. …
마지막 평가: pos
```

8.5.2 텍스트 데이터 사례: 영화평 데이터셋 IMDB

■ 텐서플로가 제공하는 IMDB

- 텐서플로는 쓰기 좋게 가공한 IMDB 데이터셋을 제공

프로그램 8-7(a) 텐서플로가 제공하는 IMDB 데이터 살펴보기

```
01  from tensorflow.keras.datasets import imdb
02  from tensorflow.keras.models import Sequential
03  from tensorflow.keras.layers import Dense,Flatten,Embedding
04  from tensorflow.keras import preprocessing
05
06  dic_siz=10000        # 사전의 크기(사전에 있는 단어 개수)
07  sample_siz=512       # 샘플의 크기
08
09  # tensorflow가 제공하는 간소한 버전의 IMDB 읽기
10  (x_train,y_train),(x_test,y_test)=imdb.load_data(num_words=dic_siz)
11  print(x_train.shape,x_test.shape)
12  print(x_train[0])
13
14  # 단어를 숫자, 숫자를 단어로 변환하는데 쓰는 표(표는 딕셔너리로 구현)
15  word2id=imdb.get_word_index()
16  id2word={word:id for id,word in word2id.items()}
17
18  for i in range(1,21):
19      print(id2word[i],end='/')
```

상위 빈도 20개 단어 출력

```
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Flatten,Embedding
from tensorflow.keras import preprocessing

dic_siz=10000 # 사전의 크기(사전에 있는 단어 개수)
sample_siz=512 # 샘플의 크기

# tensorflow가 제공하는 간소한 버전의 IMDB 읽기
#(x_train,y_train),(x_test,y_test)=imdb.load_data(num_words=dic_siz)
import numpy as np

x_train = np.load('imdb10000/x_train.npy', allow_pickle=True)
x_test = np.load('imdb10000/x_test.npy', allow_pickle=True)
y_train = np.load('imdb10000/y_train.npy', allow_pickle=True)
y_test = np.load('imdb10000/y_test.npy', allow_pickle=True)


print(x_train.shape,x_test.shape)
print(x_train[0])

# 단어를 숫자, 숫자를 단어로 변환하는데 쓰는 표(표는 딕셔너리로 구현
#word2id=imdb.get_word_index()
#id2word={word:id for id,word in word2id.items()}
word2id=np.load('imdb10000/word2id.npy', allow_pickle=True)
id2word={word:id for id,word in word2id.item().items()}

for i in range(1,21):
    print(id2word[i],end='/')
```

8.5.2 텍스트 데이터 사례: 영화평 데이터셋 IMDB

훈련 집합과 테스트 집합의 크기

```
(25000,) (25000,)
```

훈련 집합의 첫 번째 샘플

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5,
25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336,
385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19,
14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22,
17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16,
480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33,
6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2,
7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4,
381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476,
26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334,
88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]
```

```
the/and/a/of/to/is/br/in/it/i/this/that/was/as/for/with/movie/but/film/on/
```

상위 빈도 20개 단어

8.5.3 단어 임베딩

- 텍스트 데이터를 신경망에 입력하고 표현하는 방법
  - 원핫 코드 – 몇 가지 단점이 있음
  - 대안은 단어 임베딩word embedding

The **Embedding layer** in Keras is a layer that turns positive integers into dense vectors of fixed size [1]. It is used to create dense word encoding and can be thought of as a matrix multiplication by a one-hot encoding matrix or simply as a linear layer over the one-hot encoding matrix [2]. The layer can only be used on positive integer inputs of a fixed range [1]. The output of the layer is not a mathematical function of the input, but it is trained just like any other layer in your network architecture [3].

Here is an example of how to use the Embedding layer in Keras:

Python

```
model = keras.Sequential()
model.add(keras.layers.Embedding(input_dim=1000, output_dim=64, input_length=10))
```

AI가 생성한 코드입니다. 신중하게 검토하고 사용하세요. FAQ의 자세한 정보.

The above code creates a model with an Embedding layer that takes an integer matrix of size `(batch, input_length)` as input. The largest integer (i.e. word index) in the input should be no larger than 999 (vocabulary size). The output shape of the layer is `(batch_size, input_length, output_dim)` [1].

8.5.3 단어 임베딩

단어 임베딩이란?

- 단어를 저차원 공간의 벡터로 표현하는 기법
  - 보통 수백 차원을 사용
  - 밀집 벡터임
  - 단어의 의미를 표현
  - 신경망 학습을 통해 알아냄

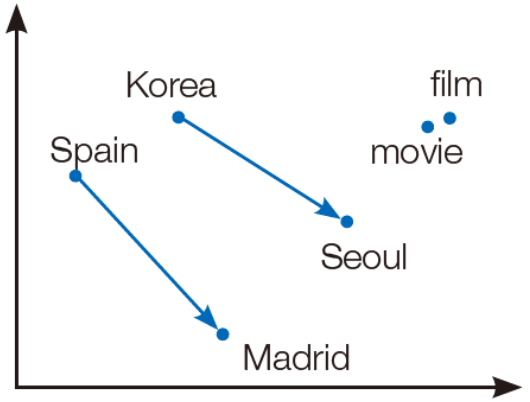


그림 8-17 단어 임베딩의 원리

[원핫 코드]

사전 크기([프로그램 8-7(a)]의 dic_siz)가 10000인 경우

```
                          10000개 요소
"python" → (1 0 0 0 0 … … 0 0 0 … 0 0 0)
```

[단어 임베딩]

단어 임베딩 공간의 차원([프로그램 8-7(b)]의 embed_space_dim)이 16인 경우

```
                    16개 요소
"python" →  (0.01 0.23 0.0 … 0.002)
```

8.5.3 단어 임베딩

- 단어 임베딩 공간에서 다층 퍼셉트론으로 IMDB 인식 [프로그램 8-7(b)]
  - 단어 임베딩 기술을 이용하여 IMDB의 샘플을 긍정과 부정으로 분류

프로그램 8-7(b) 단어 임베딩을 사용해 IMDB 데이터를 부정/긍정 평가로 분류하는 학습 모델

```
20
21  embed_space_dim=16   # 16차원의 임베딩 공간
22
23  x_train=preprocessing.sequence.pad_sequences(x_train,maxlen=sample_siz)
24  x_test=preprocessing.sequence.pad_sequences(x_test,maxlen=sample_siz)
25
```

샘플의 크기를 sample_siz(512)로 고정
512보다 작은 샘플은 pad_sequences 함수가 특수 문자로 채움

8.5.3 단어 임베딩

```
26  # 신경망 모델 설계와 학습
27  embed=Sequential()
28  embed.add(Embedding(input_dim=dic_siz,output_dim=embed_space_dim,input_
    length=sample_siz))
29  embed.add(Flatten())
30  embed.add(Dense(32,activation='relu'))
31  embed.add(Dense(1,activation='sigmoid'))
32  embed.compile(loss='binary_crossentropy',optimizer='Adam',metrics=
    ['accuracy'])
33  hist=embed.fit(x_train,y_train,epochs=20,batch_size=64,validation_data=
    (x_test,y_test),verbose=2)
34
35  embed.summary()
36
37  # 모델 평가
38  res=embed.evaluate(x_test,y_test,verbose=0)
39  print("정확률은",res[1]*100)
40
41  import matplotlib.pyplot as plt
42
43  # 학습 곡선
44  plt.plot(hist.history['accuracy'])
45  plt.plot(hist.history['val_accuracy'])
46  plt.title('Model accuracy')
47  plt.ylabel('Accuracy')
48  plt.xlabel('Epoch')
49  plt.legend(['Train','Validation'], loc='best')
50  plt.grid()
51  plt.show()
```

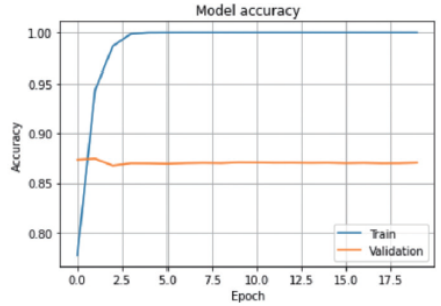
Embedding 함수는 input_dim 차원을 output_dim 차원으로 축소

input_length*output_dim 구조의 텐서를 일렬로 펼침

0(부정) 또는 1(긍정)로 분류하므로 출력 노드는 1개
손실 함수는 binary_crossentropy

8.5.3 단어 임베딩

```
Train on 25000 samples, validate on 25000 samples
Epoch 1/20
25000/25000 - 6s - loss: 0.4595 - accuracy: 0.7503 - val_loss: 0.2825 - val_accuracy:
0.8812
...
Epoch 20/20
25000/25000 - 5s - loss: 1.6844e-05 - accuracy: 1.0000 - val_loss: 0.7613 - val_
accuracy: 0.8688

Model: "sequential_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_32 (Embedding)     (None, 512, 16)           160000
_________________________________________________________________
flatten_32 (Flatten)         (None, 8192)              0
_________________________________________________________________
dense_40 (Dense)             (None, 32)                262176
_________________________________________________________________
dense_41 (Dense)             (None, 1)                 33
=================================================================
Total params: 422,209
Trainable params: 422,209
Non-trainable params: 0
_________________________________________________________________

정확률은 86.87999844551086
```



모델의 구조 Embedding-Flatten-Dense-Dense

테스트 집합에 대한 정확률

8.5.4 LSTM으로 인식: 조기 멈춤 적용

- LSTM을 사용하는 [프로그램 8-8]
  - 시계열 정보를 반영([프로그램 8-7]은 단지 단어 임베딩에 다층 퍼셉트론을 적용하여 시계열 정보를 이용하지 못함)
  - 조기 멈춤 적용([프로그램 8-7]의 실행 결과를 보면 세대 1에서 최고 성능을 이룬 후 개선 없음). 조기 멈춤early stopping은 훈련 집합에 대해 덜 수렴했더라도 검증 집합에 대해 성능 개선이 없으면 학습을 마치는 전략

프로그램 8-8 워드 임베딩을 사용하여 IMDB 데이터를 부정/긍정 평가로 분류하는 LSTM 신경망 학습

```
01  from tensorflow.keras.datasets import imdb
02  from tensorflow.keras.models import Sequential
03  from tensorflow.keras.layers import Dense,LSTM,Embedding
04  from tensorflow.keras import preprocessing
05  from tensorflow.keras.callbacks import EarlyStopping
06
07  dic_siz=10000        # 사전의 크기(사전에 있는 단어 개수)
08  sample_siz=512       # 샘플의 크기
09
10  # tensorflow가 제공하는 간소한 버전의 IMDB 읽기
11  (x_train,y_train),(x_test,y_test)=imdb.load_data(num_words=dic_siz)
12
13  embed_space_dim=16   # 16차원의 임베딩 공간
14
15  x_train=preprocessing.sequence.pad_sequences(x_train,maxlen=sample_siz)
16  x_test=preprocessing.sequence.pad_sequences(x_test,maxlen=sample_siz)
17
```

8.5.4 LSTM으로 인식: 조기 멈춤 적용

```
18  early=EarlyStopping(monitor='val_accuracy',patience=5,restore_best_
    weights=True)
19
20  # 신경망 모델의 설계와 학습(LSTM 층 포함)
21  embed=Sequential()
22  embed.add(Embedding(input_dim=dic_siz,output_dim=embed_space_dim,input_
    length=sample_siz))
23  embed.add(LSTM(units=32))
24  embed.add(Dense(1,activation='sigmoid'))
25  embed.compile(loss='binary_crossentropy',optimizer='Adam',metrics=['accuracy'])
26  hist=embed.fit(x_train,y_train,epochs=20,batch_size=64,validation_split=0.2,v
    erbose=2,callbacks=[early])
27
28  # 모델 평가
29  res=embed.evaluate(x_test,y_test,verbose=0)
30  print("정확률은",res[1]*100)
31
32  import matplotlib.pyplot as plt
33
34  # 학습 곡선
35  plt.plot(hist.history['accuracy'])
36  plt.plot(hist.history['val_accuracy'])
37  plt.title('Model accuracy')
38  plt.ylabel('Accuracy')
39  plt.xlabel('Epoch')
40  plt.legend(['Train','Validation'], loc='best')
41  plt.grid()
42  plt.show()
```

가장 높은 성능을 발휘했을 때의 가중치를 취함

5 세대 동안 성능 향상 없으면 멈춤

검증 집합에 대한 정확률을 조기 멈춤 기준으로 사용

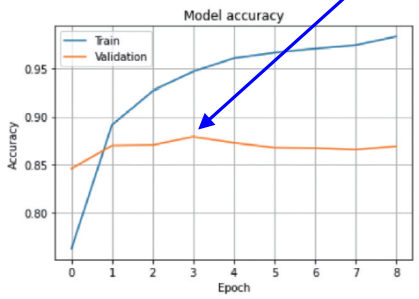
콜백 함수를 통해 조기 멈춤 적용

훈련 집합의 20%를 떼어 검증 집합으로 사용

8.5.4 LSTM으로 인식: 조기 멈춤 적용

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/20
20000/20000 - 119s - loss: 0.4790 - accuracy: 0.7628 - val_loss: 0.3398 - val_
accuracy: 0.8640
Epoch 2/20
20000/20000 - 113s - loss: 0.2587 - accuracy: 0.9004 - val_loss: 0.3117 - val_
accuracy: 0.8722
Epoch 3/20
20000/20000 - 109s - loss: 0.1976 - accuracy: 0.9288 - val_loss: 0.3347 - val_
accuracy: 0.8784
Epoch 4/20
20000/20000 - 119s - loss: 0.1684 - accuracy: 0.9401 - val_loss: 0.3417 - val_
accuracy: 0.8754
Epoch 5/20
20000/20000 - 107s - loss: 0.1239 - accuracy: 0.9588 - val_loss: 0.3424 - val_
accuracy: 0.8682
Epoch 6/20
20000/20000 - 105s - loss: 0.0987 - accuracy: 0.9685 - val_loss: 0.4095 - val_
accuracy: 0.8734
Epoch 7/20
20000/20000 - 105s - loss: 0.0908 - accuracy: 0.9709 - val_loss: 0.4010 - val_
accuracy: 0.8618
Epoch 8/20
20000/20000 - 104s - loss: 0.1116 - accuracy: 0.9596 - val_loss: 0.4849 - val_
accuracy: 0.8638
정확률은 86.76400184631348
```

세대 3 이후 다섯 세대 동안 성능 향상이 없어 학습을 멈춤



8.5.4 LSTM으로 인식: 조기 멈춤 적용

- [프로그램 8-8]과 [프로그램 8-7]의 성능 분석
  - [프로그램 8-8]은 LSTM을 사용해 시계열 특성을 반영하고 조기 멈춤을 적용해 더 유리한 상황인데 [프로그램 8-7]보다 열등. 왜?
    - 두 프로그램 모두 단어의 빈도수에 따라 분류하는 듯
      - boring, terrible, bad 등이 많으면 부정, wonderful, good 등이 자주 나타나면 긍정으로 분류
    - 문장의 의미를 파악하지 못한 채 분류
      - 예, "To me all of the movies are terrible, but this one is not.' 문장에서 terrible이 있다는 이유로 두 프로그램 모두 부정으로 분류

- 문장의 의미 이해하려면 발전된 자연어 처리 알고리즘 필요
  - 다음 절에서 word2vec과 GloVe를 간략히 소개

8.5.5 word2vec과 GloVe

- 단어를 벡터 공간에 표현하는 단어 임베딩 기술
  - 오래 전부터 연구되어온 아이디어
  - 영국의 언어학자 퍼스 "You shall know a word by the company it keeps." 단어 간의 상호작용이 매우 중요하다는 통찰
    - 예) "영화"라는 단어는 "아카데미", "흥행" 등의 단어와 함께 등장할 가능성 높음
  - 고전적인 기법들: TF(term frequency), LSA, 신경망 기법들
  - 2010년대에는 딥러닝을 활용한 단어 임베딩 기술이 주류
    - Word2vec(구글)
      - 1000억개 가량의 뉴스를 모아둔 데이터셋으로 학습. 300만개 가량의 단어를 300차원 공간에 표현
    - GloVe(스탠퍼드 대학)
      - 위키피디아 문서 데이터를 사용하여 학습. 40만개 가량의 단어를 50, 100, 200, 300 차원 공간에 표현

8.5.5 word2vec과 GloVe

- GloVe로 단어 연관 관계를 찾는 프로그래밍 실습
  - 파일 크기가 상대적으로 작은 GloVE를 가지고 실습
  - 100차원 파일인 glove.6B.100d.txt 사용

프로그램 8-9(a) GloVe를 이용해 단어 임베딩 이해하기

```
01  import os
02  import numpy as np
03  from scipy.spatial import distance
04
05  # IMDB 원본 데이터 읽기
06  fname='./glove.6B/glove.6B.100d.txt'
07  f=open(fname,encoding='utf8')
08
09  for line in f:       # 내용을 살필 목적으로 첫 번째 단어만 출력
10      print(type(line))
11      print(line)
12      break
13
```

거리 계산용 함수

100차원 짜리 사용

8.5.5 word2vec과 GloVe

```
14  # 사전 구축(딕셔너리 자료구조로 표현)
15  dictionary={}
16  for line in f:
17      li=line.split()
18      word=li[0]
19      vector=np.asarray(li[1:],dtype='float32')
20      dictionary[word]=vector
21
22  # 가장 가까운 단어를 찾아주는 함수
23  def find_closest_words(vector):
24      return sorted(dictionary.keys(), key=lambda w: distance.euclidean(dictio
        nary[w],vector))
25
26  # 가까운 단어 찾기
27  print(find_closest_words(dictionary['movie'])[:5])
28  print(find_closest_words(dictionary['school'])[:5])
29  print(find_closest_words(dictionary['oak'])[:5])
30
31  # 단어 추론
32  print(find_closest_words(dictionary["seoul"]-dictionary["korea"]+dictionary["
    spain"])[:5])
33  print(find_closest_words(dictionary["animal"]-dictionary["lion"]+dictionary["
    oak"])[:5])
34  print(find_closest_words(dictionary["queen"]-dictionary["king"]+dictionary["a
    ctress"])[:5])
```

문자열 표현을 벡터 형식으로 변환하고 딕셔너리에 저장

매개변수 vector와 가까운 순으로 정렬한 키를 반환하는 함수

'movie'와 가장 가까운 5개 단어 찾기

32~34행 단어 연관 관계 추론

```
Seoul-Korea+Spain      →      Madrid
animal-lion+oak        →      tree
queen-king+actress     →      actor
```

8.5.5 word2vec과 GloVe

```
<class 'str'>
the -0.038194 -0.24487 0.72812 -0.39961 0.083172 0.043953 -0.39141
0.3344 -0.57545 0.087459 0.28787 -0.06731 0.30906 -0.26384 -0.13231
-0.20757 0.33395 -0.33848 -0.31743 -0.48336 0.1464 -0.37304 0.34577
0.052041 0.44946 -0.46971 0.02628 -0.54155 -0.15518 -0.14107 -0.039722
0.28277 0.14393 0.23464 -0.31021 0.086173 0.20397 0.52624 0.17164
-0.082378 -0.71787 -0.41531 0.20335 -0.12763 0.41367 0.55187 0.57908
-0.33477 -0.36559 -0.54857 -0.062892 0.26584 0.30205 0.99775 -0.80481
-3.0243 0.01254 -0.36942 2.2167 0.72201 -0.24978 0.92136 0.034514
0.46745 1.1079 -0.19358 -0.074575 0.23353 -0.052062 -0.22044 0.057162
-0.15806 -0.30798 -0.41625 0.37972 0.15006 -0.53212 -0.2055 -1.2526
0.071624 0.70565 0.49744 -0.42063 0.26148 -1.538 -0.30223 -0.073438
-0.28312 0.37104 -0.25217 0.016215 -0.017099 -0.38984 0.87424 -0.72569
-0.51058 -0.52028 -0.1459 0.8278 0.27062
['movie', 'film', 'movies', 'films', 'hollywood']
['school', 'college', 'schools', 'elementary', 'students']
['oak', 'pine', 'cedar', 'walnut', 'grove']

['madrid', 'spain', 'santiago', 'seville', 'valencia']
['oak', 'trees', 'woodland', 'wood', 'organic']
['actress', 'actresses', 'dancer', 'actor', 'comedienne']
```

단어 'the'는 100차원 벡터 (-0.038194,-0.24487,...)로 표현

가까운 단어 5개 찾기 결과

단어 연관 관계 찾기 결과

8.5.5 word2vec과 GloVe

- 고차원을 2차원 또는 3차원으로 축소하여 데이터를 가시화하는 기법
  - PCA, iso-contour 등 많은 기법이 있음
  - tsne가 가장 뛰어남
- [프로그램 8-9(b)]는 tsne를 사용하여 100차원 벡터를 2차원으로 변환

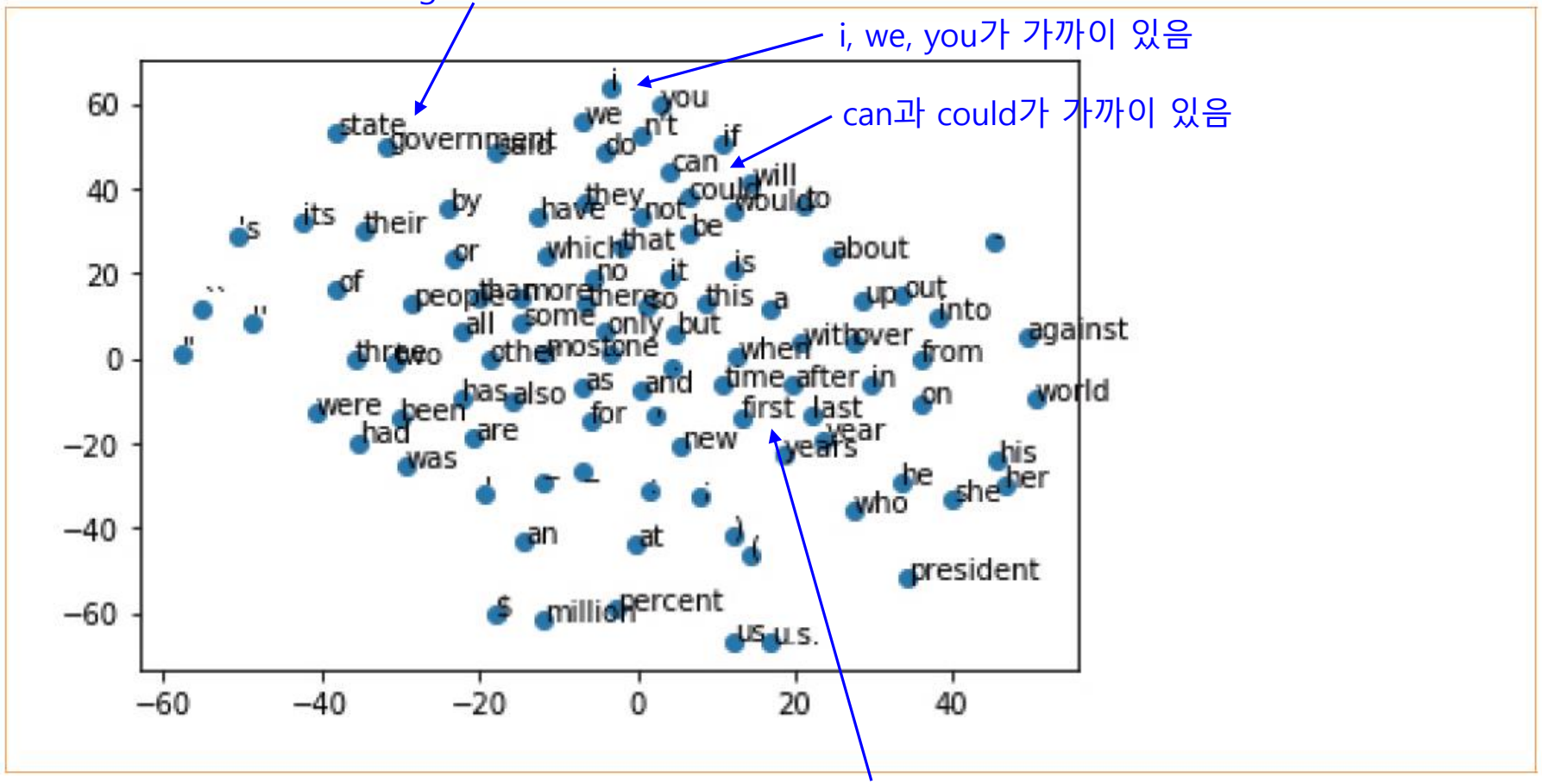
프로그램 8-9(b) tsne를 이용한 시각화

```
35
36  from sklearn.manifold import TSNE
37  import matplotlib.pyplot as plt
38
39  # tsne를 이용하여 2차원 공간으로 축소하고 시각화
40  tsne=TSNE(n_components=2,random_state=0)
41  words=list(dictionary.keys())
42  vectors=[dictionary[word] for word in words]
43  p2=tsne.fit_transform(vectors[:100])
44  plt.scatter(p2[:,0],p2[:,1])
45
46  for label,x,y in zip(words,p2[:,0],p2[:,1]):
47      plt.annotate(label,xy=(x,y))
```

2차원으로 축소하라는 지시

앞에 있는 100개만 시각화

8.5.5 word2vec과 GloVe



8.5. 자연어 처리

영화평 데이터셋 IMDB