딥러닝개론

Introduction to Deep Learning

* 동일교과지정: 무인항공시스템을 위한 딥러닝 개론

시계열 데이터와 순환 신경망

PREVIEW

- 인공지능이 다루어야 하는 또 다른 중요한 데이터는 시계열 데이터 (time series data)임
- 시계열 데이터에는 시간 정보가 들어가 있으며, 문장은 대표적인 시계열 데이터임
- 예를 들어 “세상에는 시계열 데이터가 참 많다” 라는 문장에서는 단어가 나타나는 순서, 즉 시간 정보가 중요함
- 또한 문장 마다 길이가 천차만별로 다르듯이 시계열 데이터도 샘플마다 길이가 다름
- 딥러닝은 시계열 데이터 처리에서도 혁신을 이루었음
- 데이터의 성질이 달라 완전히 새로운 구조의 신경망을 설계해야 할 것 처럼 보이는데, 다행히 다층 퍼셉트론의 은닉층에 피드백을 추가하면 시계열 데이터를 처리할 수 있으며, 이런 신경망을 순환 신경망(RNN, Recurrent Neural Network) 이라고 함

PREVIEW

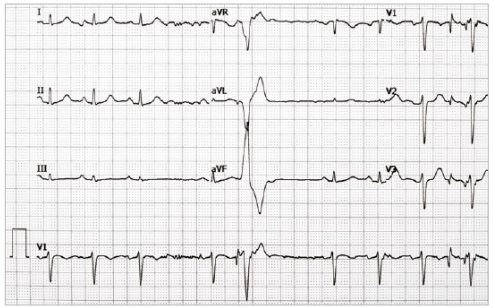
- RNN 순환 신경망은 장기 문맥을 처리하는데 한계가 있음
  - 장기 문맥이란 서로 강하게 관련이 있는 요소가 멀리 떨어져 나타나는 상황을 뜻함
- 예를 들어, 주어와 서술어는 문장의 의미를 나타내는 두 핵심 요소인데, 긴 문장에서는 둘 사이에 형용구에 해당하는 단어가 수십 개 나타나는 경우가 있음
- LSTM (Long Short Term Memory)은 순환 신경망에 **선별 기억 능력을 추가한 모델**로, 장기 문맥을 처리하는데 강력한 성능을 발휘함
- 현재 LSTM은 우월한 성능 때문에 시계열 데이터를 처리하는데 가장 널리 사용됨

8.1 시계열 데이터의 이해

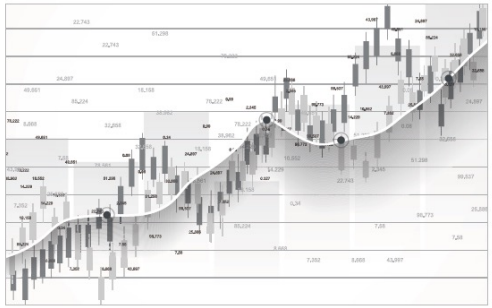
- 시계열 데이터는 시간 축을 따라 신호가 변하는 동적 데이터임
- 지금까지 배운 SVM, MLP, DNN, CNN 은 정적 데이터를 한꺼번에 입력 받으므로 시계열 데이터를 처리하는데 적합하지 않음
- 시계열 데이터를 고정 길이의 정적 데이터로 바꾸는 기법을 적용할 수도 있지만 정보 손실이 크고 변환 기법을 사람이 설계해야 하는 부담이 있음
- 딥러닝에서는 시계열 특성을 십분 활용하는 순환 신경망 또는 LSTM을 사용함

8.1 시계열 데이터의 이해

- [그림 8-1]은 다양한 시계열 데이터의 예
  - (a) 심전도 신호를 분석하는 의료 인공지능을 만들어 이용하면 더욱 정확하게 병을 진단할 수 있음
  - (b) 주식 시세를 잘 분석하는 인공지능 제품을 만들면 투자에 유리할 수 있음
  - (c) 정확률이 높은 음성 인식기를 만들면 경쟁력 있는 인공지능 제품을 만들 수 있음
  - (d) 유전자 분석하는 인공지능을 만들면 신약 개발에 활용할 수 있음



(a) 심전도



(b) 주식 시세

(c) 음성 인식 데이터

(d) 유전자 염기 서열

그림 8-1 다양한 시계열 데이터

8.1.1 시계열 데이터의 특성

- 시계열 데이터의 특성
  - 요소의 순서가 중요
  - 샘플의 길이가 다름
  - 문맥 의존성을 지님
  - 계절성을 가지는 데이터가 많음

* 각 특성에 대해서 다음 장에 있음

8.1.1 시계열 데이터의 특성

- **요소의 순서가 중요**
  - “세상에는 시계열 데이터가 참 많다” 라는 문장을 “데이터가 시계열 많다 참 세상에는” 으로 바꾸면 의미가 크게 훼손됨
  - 반면 시간 정보가 없는 데이터는 특징의 순서를 바꿔 표현해도 아무런 문제가 없음
  - 예를 들어, iris 데이터에서는 샘플이 꽃잎 길이, 꽃잎 너비, 꽃받침 길이, 꽃받침 너비라는 특징 벡터로 표현되는데, 벡터의 요소 즉 특징의 순서를 바꿔 꽃잎 너비, 꽃받침 너비, 꽃잎 길이, 꽃받침 길이로 표현해도 상관없음

- **샘플의 길이가 다름**
  - 인공지능 이란 단어를 짧게 발음하는 사람도 있고 인~공~지~능으로 길게 발음하는 사람도 있음
  - 심전도를 잴 때도 심각한 환자는 더욱 정밀하게 진단하기 위해 더 길게 축정하기도 함
  - 염기 서열에서도 잡음이 섞이거나 중간에 빠지는 염기가 발생해 측정할 때마다 길이가 조금씩 다를 수 있음

8.1.1 시계열 데이터의 특성

- **문맥 의존성을 지님**
  - “시계열은 앞에서 말한 바와 같이 데이터를 구성하는 요소의 순서, 즉 나타나는 순서가 중요하다는 특성이 있다” 라는 문장에서 주어인 “시계열은”과 서술어인 “특성이 있다”는 밀접한 연관성이 있으며, 이런 연관성을 문맥 의존성 (context dependency)라고 함
  - 시계열을 처리하는 기계 학습 모델은 문맥 의존성을 스스로 발견하고 표현하는 능력이 있어야 함
  - 예시 문장에서 주어와 서술어 사이에 단어 12개가 있으므로 주어와 서술어는 12만큼 떨어져 있으며, 이처럼 연관성이 깊은 요소가 멀리 떨어져서 나타나는 상황을 장기 문맥 의존성 (long-term context dependency)라고 함

- **계절성을 가지는 데이터가 많음**
  - 상추 판매량은 야외 바비큐에 적절한 계절에 매출이 치솟는 경향이 있음
  - 수제 맥주 전문점은 금요일에 매출이 치솟는 경향이 있음
  - 미세먼지 수치나 항공권 판매량은 계절에 영향이 있음

8.1.1 시계열 데이터의 특성

- **시계열 데이터(동적데이터)의 정의**

$$\mathbf{x}=\left(\mathbf{a}^1\ \mathbf{a}^2\cdots\mathbf{a}^t\right) \qquad (8.1)$$

  - 데이터의 길이가 가변적이므로 d 대신 t로 표기함
  - 벡터의 요소를 고려하여 굵은 글씨체로 표기함

- **시계열 데이터의 예**
  - 매일 기온, 습도, 미세먼지 농도를 기록한다면,
    첫째날은 $\mathbf{a}^1=(23.5,42,0.1)$, 둘째날은 $\mathbf{a}^2=(25.5,45,0.08)$ 로 표현 가능
  - 심전도의 경우 10개의 채널 값이 들어오고, 30초 분량의 심전도를 초당 20번 샘플링한다면
    t=600인 샘플 X가 생성되는데, x의 요소 a^i는 10차원 벡터에 해당됨

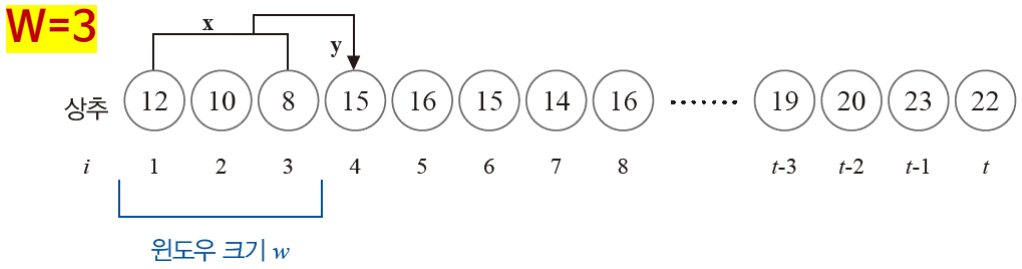
8.1.2 미래 예측을 위한 데이터 준비

- 순환 신경망은 유연한 구조라 여러 문제에 적용 가능
  - 미래를 예측(prediction vs. forecasting)하는 문제
    - Prediction 예측 : 새로운 숫자의 패턴이 입력되면 어떤 숫자인지 분류하는 일
    - Forecasting 예측(예보) : 시계열 데이터를 보고 미래를 추정하는 일

  - 시계열 데이터를 보고 미래를 추정하는 forecasting의 예
    - 내일의 주가 예측
    - 내일의 날씨 예측
    - 내일의 판매량 예측

8.1.2 미래 예측을 위한 데이터 준비

- 시계열 패턴 하나로 미래를 예측하는 신경망을 어떻게 학습할 것인가?
  - 미래를 예측하는 문제에서는 아래와 같이 일정한 길이로 패턴을 잘라 여러 샘플을 만듬
  - 이때 이전 요소 몇 개를 볼 것인지 나타내는 윈도우 크기인 w 설정이 필요함



(a) 입력 데이터 샘플링

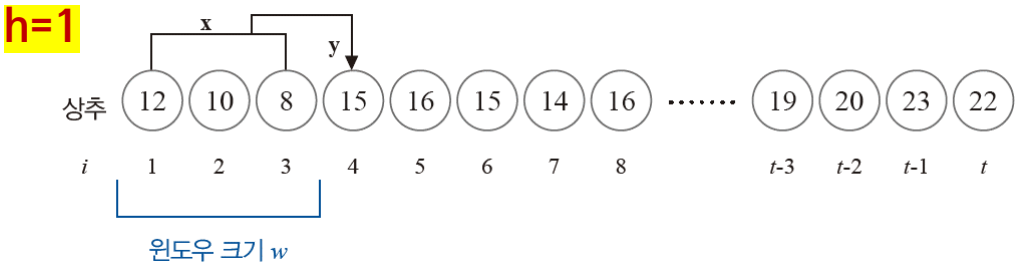
샘플	x	y
1	(12, 10, 8)	15
2	(10, 8, 15)	16
3	(8, 15, 16)	15
4	(15, 16, 15)	14
...	...	...
t-w	(19, 20, 23)	22

(b) 입력 패턴에서 생성한 샘플

- t= (1, 2, 3) 순간의 값을 보고 t=4 순간을 예측하고, (2,3,4) 순간의 값을 보고 5 순간을 예측하는 방식을 반복하면서 샘플링함

8.1.2 미래 예측을 위한 데이터 준비

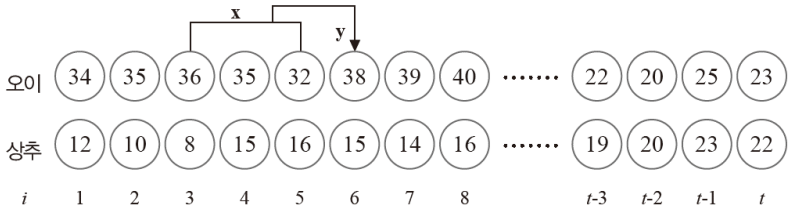
- 시계열 패턴 하나로 미래를 예측하는 신경망을 어떻게 학습할 것인가?
  - 이전 3일의 판매량을 보고, 이틀 또는 일주일 후의 판매량을 예측해야 하는 상황이 있음
  - 이때 얼마나 먼 미래를 예측할 것인지 지정하는 요인을 수평선 계수(horizon factor)라고 함
  - 일반적으로 수평선 계수 h가 클수록 어려운 문제가 됨



(a) 입력 데이터 샘플링

8.1.2 미래 예측을 위한 데이터 준비

- 시계열 패턴 하나로 미래를 예측하는 신경망을 어떻게 학습할 것인가?
  - 품목이 2개 이상인 시계열 데이터를 동시에 처리해야 하는 다중 채널인 경우도 있음



(a) 입력 패턴

샘플	x	y
1	((34, 12), (35, 10), (36, 8))	(35, 15)
2	((35, 10), (36,8), (35, 15))	(32, 16)
3	((36, 8), (35, 15), (32, 16))	(38, 15)
4	((35, 15), (32, 16), (38, 15))	(39, 14)
…	…	…
t - *w*	((22, 19), (20, 20), (25, 23))	(23, 22)

(b) 입력 패턴에서 생성한 샘플

그림 8-3 미래 예측 문제에서 샘플 생성하기(다중 채널)

8.1.3 시계열 데이터 사례: 비트코인 가격

- 비트코인 가격 데이터 다뤄보기
  - 디지털 통화에 관한 뉴스 사이트 코인데스크에서 비트코인 데이터를 다운로드

(a) 코인데스크에서 데이터를 다운로드하기

	A	B	C	D	E	F	G
1	Currency	Date	Closing Pr	24h Open	24h High	24h Low (USD)	
2	BTC	2019-02-28	3772.94	3796.64	3824.17	3666.52	
3	BTC	2019-03-01	3799.68	3773.44	3879.23	3753.8	
4	BTC	2019-03-02	3811.61	3799.37	3840.04	3788.92	
5	BTC	2019-03-03	3804.42	3806.69	3819.19	3759.41	
6	BTC	2019-03-04	3782.66	3807.85	3818.7	3766.24	
7	BTC	2019-03-05	3689.86	3783.36	3804.35	3663.48	
8	BTC	2019-03-06	3832.08	3701.05	3866.72	3688.7	
9	BTC	2019-03-07	3848.96	3832.59	3881.97	3802.52	
10	BTC	2019-03-08	3859.84	3848.96	3890.75	3827.67	
11	BTC	2019-03-09	3828.37	3859.8	3918	3778.52	
12	BTC	2019-03-10	3898.87	3841.89	3948.88	3832.14	
13	BTC	2019-03-11	3899.66	3916.76	3921.93	3865.92	
14	BTC	2019-03-12	3851.25	3899.46	3913.46	3819.93	

(b) 비트코인 가격이 저장된 데이터 프레임

그림 8-4 코인데스크 사이트에서 다운로드한 비트코인 가격 데이터

*다운로드 사이트의 화면이나 메뉴가 바뀌는 경우는 빈번하여, 책과 다른 화면이 나올 수 있음

8.1.3 시계열 데이터 사례: 비트코인 가격

- 비트코인 가격 데이터 다뤄보기
  - 다운로드한 파일을 열어보면, 아래의 그림과 같이 6개의 열이 있음
    - [Currency, Date, Closing Price(USD),24h Open(USD), 24h High(USD), 24h Low(USD)] 로 구성

	A	B	C	D	E	F	G
1	Currency	Date	Closing Pr	24h Open	24h High	24h Low (USD)	
2	BTC	2019-02-28	3772.94	3796.64	3824.17	3666.52	
3	BTC	2019-03-01	3799.68	3773.44	3879.23	3753.8	
4	BTC	2019-03-02	3811.61	3799.37	3840.04	3788.92	
5	BTC	2019-03-03	3804.42	3806.69	3819.19	3759.41	
6	BTC	2019-03-04	3782.66	3807.85	3818.7	3766.24	
7	BTC	2019-03-05	3689.86	3783.36	3804.35	3663.48	
8	BTC	2019-03-06	3832.08	3701.05	3866.72	3688.7	
9	BTC	2019-03-07	3848.96	3832.59	3881.97	3802.52	
10	BTC	2019-03-08	3859.84	3848.96	3890.75	3827.67	
11	BTC	2019-03-09	3828.37	3859.8	3918	3778.52	
12	BTC	2019-03-10	3898.87	3841.89	3948.88	3832.14	
13	BTC	2019-03-11	3899.66	3916.76	3921.93	3865.92	
14	BTC	2019-03-12	3851.25	3899.46	3913.46	3819.93	

(b) 비트코인 가격이 저장된 데이터 프레임

8.1.3 시계열 데이터 사례: 비트코인 가격

- **프로그램 8-1 (a) 비트코인 가격 데이터 읽기**
  - https://www.kaggle.com/code/yukyungchoi/2024-dl-btc-p1

[18]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

coindesk_data_train = pd.read_csv("/kaggle/input/2024-1-dl-w-11-btc-price-prediction/traindata.csv",header=0)
coindesk_data_test = pd.read_csv("/kaggle/input/2024-1-dl-w-11-btc-price-prediction/testdata.csv",header=0)
```

```
coindesk_data_train.head(5)
```

[19…]

	date	closing price	open price	high price	low price
0	2019-02-28	3816.6	3814.6	3883.7	3783.3
1	2019-03-01	3821.9	3816.7	3855.8	3816.4
2	2019-03-02	3823.1	3821.9	3843.2	3783.6
3	2019-03-03	3809.5	3823.2	3836.6	3789.7
4	2019-03-04	3715.9	3809.7	3828.4	3681.8

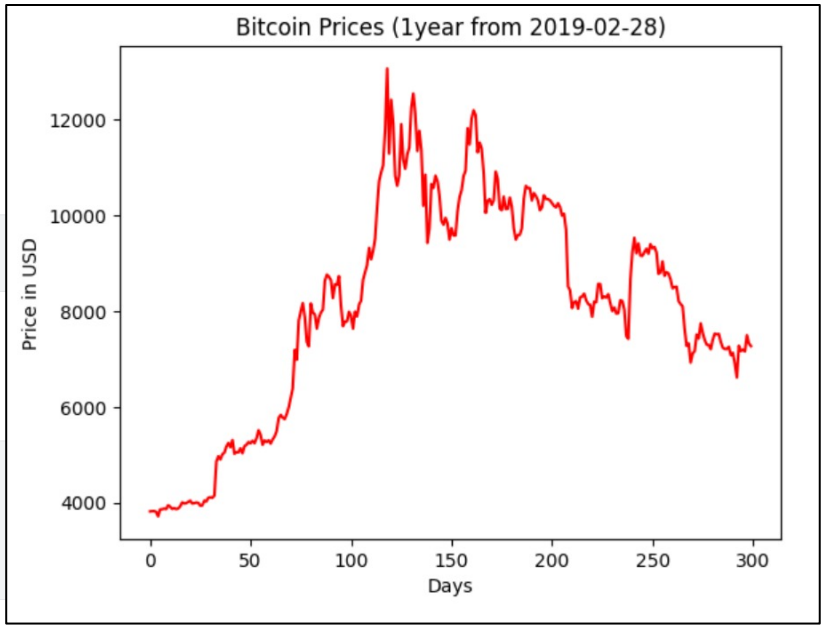
+ Code + Markdown

[20]:
```
seq = coindesk_data_train[['closing price']].to_numpy()
print('데이터 길이:', len(seq), '\n앞쪽 5개 값:', seq[0:5])
```

종가

```
데이터 길이: 300
앞쪽 5개 값: [[3816.6]
 [3821.9]
 [3823.1]
 [3809.5]
 [3715.9]]
```

[22]:
```
# 그래프로 데이터 확인
plt.plot(seq,color='red')
plt.title('Bitcoin Prices (1year from 2019-02-28)')
plt.xlabel('Days')
plt.ylabel('Price in USD')
plt.show()
```



8.1.3 시계열 데이터 사례: 비트코인 가격

- **프로그램 8-1 (a) 비트코인 가격 데이터 읽기**
  - https://www.kaggle.com/code/yukyungchoi/2024-dl-btc-p1

[18]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

coindesk_data_train = pd.read_csv("/kaggle/input/2024-1-dl-w-11-btc-price-prediction/traindata.csv",header=0)
coindesk_data_test = pd.read_csv("/kaggle/input/2024-1-dl-w-11-btc-price-prediction/testdata.csv",header=0)
```

```
coindesk_data_train.head(5)
```

[19…]

	date	closing price	open price	high price	low price
0	2019-02-28	3816.6	3814.6	3883.7	3783.3
1	2019-03-01	3821.9	3816.7	3855.8	3816.4
2	2019-03-02	3823.1	3821.9	3843.2	3783.6
3	2019-03-03	3809.5	3823.2	3836.6	3789.7
4	2019-03-04	3715.9	3809.7	3828.4	3681.8

+ Code + Markdown

[20]:
```
seq = coindesk_data_train[['closing price']].to_numpy()
print('데이터 길이:', len(seq), '\n앞쪽 5개 값:', seq[0:5])
```

```
데이터 길이: 300
앞쪽 5개 값: [[3816.6]
 [3821.9]
 [3823.1]
 [3809.5]
 [3715.9]]
```

[22]:
```
# 그래프로 데이터 확인
plt.plot(seq,color='red')
plt.title('Bitcoin Prices (1year from 2019-02-28)')
plt.xlabel('Days')
plt.ylabel('Price in USD')
plt.show()
```

NOTE [[ ]]와 [ ]의 차이

[프로그램 8-1(a)]에서 08행의 coindesk_data[['Closing Price (USD)']] 구문에 [[…]]을 사용했다. 08행을 coindesk_data['Closing Price (USD)']로 바꾸어 […] 방식을 사용하면 무슨 차이가 있을까? 간단한 예를 들면, [[…]] 방식은 [[3] [5] [7] [6]]과 같이 표현되고 […] 방식은 [3 5 7 6]으로 표현된다. [프로그램 8-1(a)]는 매일 형성되는 네 가지 주식 시세인 종가, 시가, 고가, 저가 중에서 종가만 취하여 실험을 하기 때문에 벡터의 벡터인 [[…]]와 벡터인 […]의 차이가 두드러지지 않는다. 네 값을 모두 사용하는 다중 채널의 경우는 반드시 벡터의 벡터인 [[…]] 표현을 사용해야 한다. 종가만 사용하는 프로그램에서 [[…]] 표현으로 코딩해두면 다중 채널로 확장이 쉬워지기 때문에 더 좋은 프로그램이 된다. 다중 채널로 확장하는 작업은 [프로그램 8-3]에서 다룬다.

[[]] vs []의 차이는 각자 확인 할 것!

8.1.3 시계열 데이터 사례: 비트코인 가격

- **프로그램 8-2(b) 1년치 비트코인 가격 데이터를 윈도우로 자르기**
  - https://www.kaggle.com/code/yukyungchoi/2024-dl-btc-p1
  - 이제 시계열 데이터를 윈도우 단위로 잘라 샘플링 해야함

[23]:
```
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)
```

```
# 윈도우 크기 7, 수평선 계수 1
w=7; h=1

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

```
(293, 7, 1) (293, 1)
[[3816.6]
 [3821.9]
 [3823.1]
 [3809.5]
 [3715.9]
 [3857.2]
 [3863. ]] [3875.1]
[[6613.3]
 [7276. ]
 [7165.5]
 [7196.4]
 [7156.2]
 [7495.8]
 [7322.8]] [7268.3]
```

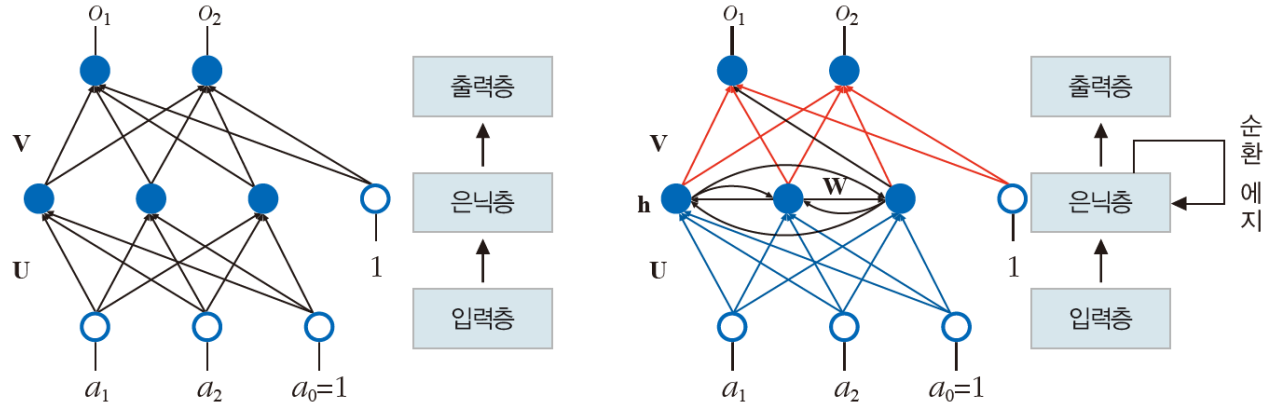
+ Code + Markdown

8.2 순환 신경망

- 시계열 데이터를 처리하는 신경망을 설계할 때는 시간에 따라 값이 하나씩 순차적으로 들어온다는 사실을 고려해야 함
- 컨볼루션 신경망(CNN)과 깊은 다층 퍼셉트론(MLP)에서는 데이터가 맨 왼쪽에 있는 입력층으로 한꺼번에 들어온다는 사실을 감안하면, 새로운 신경망은 이전 신경망과 구조 측면에서 본질적으로 달라야 할 것임
- 순환 신경망은 다행스럽게도 다층 퍼셉트론(MLP)을 약간 고쳐서 사용 가능함

8.2.1 구조와 동작, 학습 알고리즘

- 시계열 데이터를 처리하는 신경망
  - [그림8-5(a)]는 [그림4-13]을 다시 그린 것으로, **순환 신경망(RNN, Recurrent Neural Network)**과 비교하기 위해 왼쪽에서 오른쪽으로 데이터가 흐르던 방식을 아래에서 위로 흐르게 90도 회전해 그렸음
  - 또한 U^1과 U^2로 표기하던 가중치 집합을 U와 V로 바꾸어 표기함



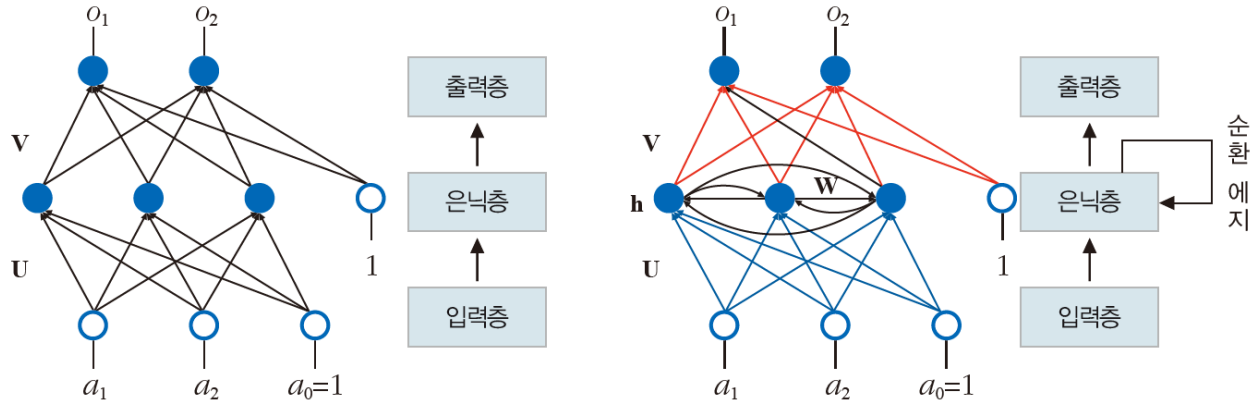
(a) 다층 퍼셉트론([그림 4-13])

(b) 순환 신경망([그림 4-13]에 순환 에지 추가)

그림 8-5 다층 퍼셉트론과 순환 신경망의 비교

8.2.1 구조와 동작, 학습 알고리즘

- 시계열 데이터를 처리하는 신경망
  - **순환 신경망은 다층 퍼셉트론과 한 가지만 빼고 같음**
  - 순환 신경망에는 은닉층에 있는 노드끼리 에지로 연결되어 있음
  - 이 에지는 은닉층 내에서 정보를 순환하므로 **순환 에지(recurrent edge)**라고 함
  - 순환 에지라는 단순한 아이디어로 다층 퍼셉트론이 시계열 데이터 처리에 적합한 순환 신경망으로 변신한 것임
  - 순환 신경망은 순환 에지를 통해 시간 정보를 입력하고 가변 길이를 다루고 문맥 정보를 처리하는 능력을 갖출 수 있음



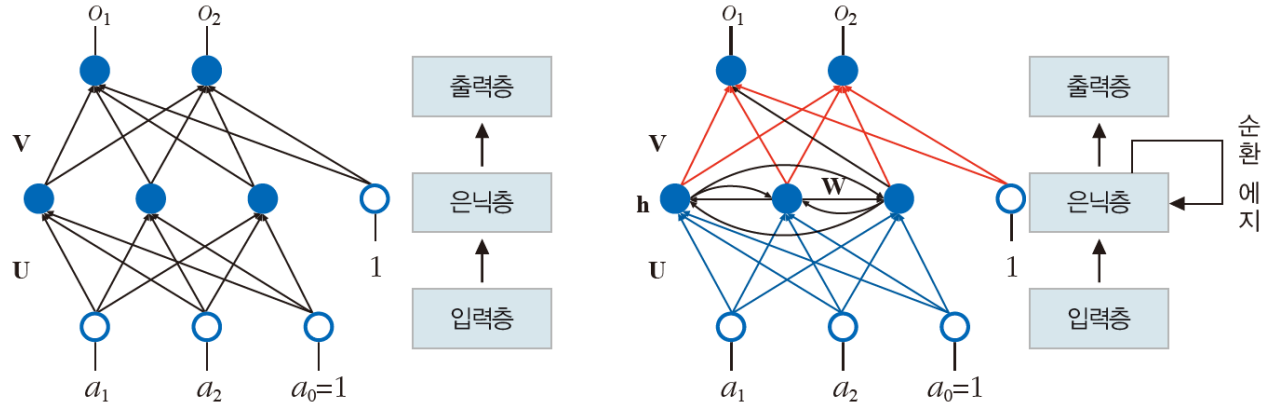
(a) 다층 퍼셉트론([그림 4-13])

(b) 순환 신경망([그림 4-13]에 순환 에지 추가)

그림 8-5 다층 퍼셉트론과 순환 신경망의 비교

8.2.1 구조와 동작, 학습 알고리즘

- 시계열 데이터를 처리하는 신경망
  - 순환 신경망이 다층 퍼셉트론과 두드러지게 다른 점은 가중치 집합임
    - 다층 퍼셉트론 {U, V} vs. 순환 신경망 {U, V, **W**}



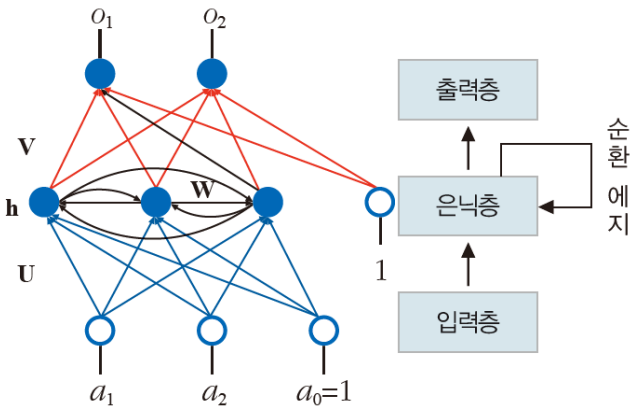
(a) 다층 퍼셉트론([그림 4-13])

(b) 순환 신경망([그림 4-13]에 순환 에지 추가)

그림 8-5 다층 퍼셉트론과 순환 신경망의 비교

8.2.1 구조와 동작, 학습 알고리즘

- 시계열 데이터를 처리하는 신경망
  - 순환 에지는 어떻게 시간 정보를 처리할 수 있을까?
  - [그림 8-6]은 [그림8-5(b)]의 순환 신경망을 단지 펼친 것
  - [그림8-6]의 신경망에는 1이라는 순간, 2라는 순간, …, t라는 순간이 있음
  - i라는 순간에는 입력층 a^i가 입력되고 은닉층의 상태가 h^i로 변하고 출력층에서 o^i가 출력됨
  - 그러나, 가중치 U, V, W를 보면 시간을 나타내는 첨자가 없으며, 이는 순환 신경망이 모든 순간 가중치를 공유한다는 것을 의미함



(b) 순환 신경망([그림 4-13]에 순환 에지 추가)

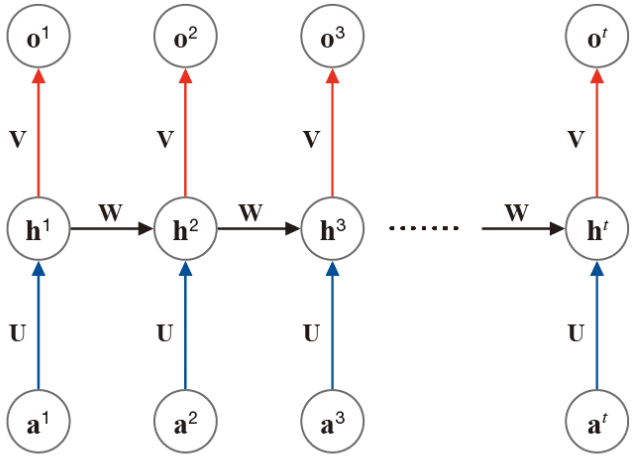


그림 8-6 펼친 순환 신경망

8.2.1 구조와 동작, 학습 알고리즘

- 순환 신경망의 동작
  - i 순간의 a^i는 가중치 **U**를 통해 은닉층의 상태 h^i에 영향을 미치고, h^i는 가중치 **V**를 통해 출력값 o^i에 영향을 미침. h^{i-1}는 가중치 **W**를 통해 h^i에 영향을 미침
  - 은닉층에서 일어나는 계산

$$\mathbf{h}^i = \tau_1\left(\mathbf{W}\mathbf{h}^{i-1} + \mathbf{U}\mathbf{a}^i\right) \qquad (8.2)$$

  - 출력층에서 일어나는 계산

$$\mathbf{o}^i = \tau_2\left(\mathbf{V}\mathbf{h}^i\right) \qquad (8.3)$$

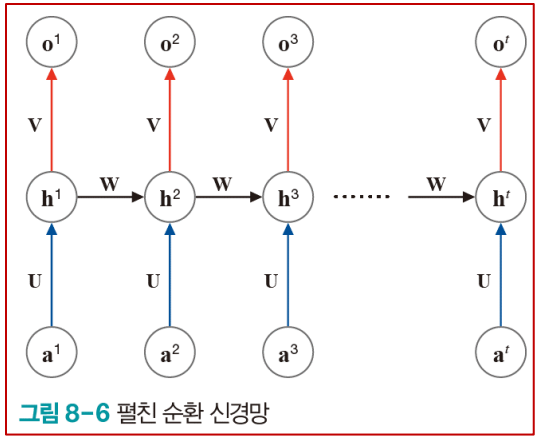
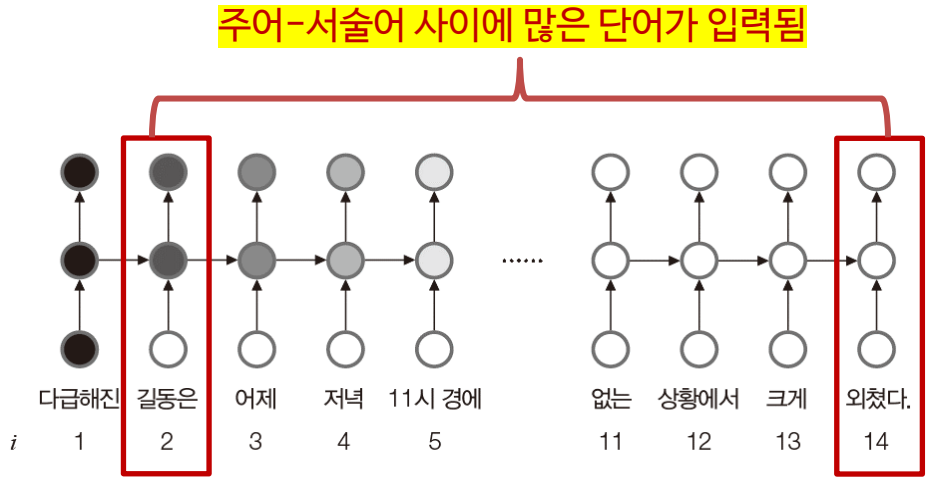


그림 8-6 펼친 순환 신경망

  - 식 (8.2)에서 $\mathbf{W}h^{i-1}$ 항을 제외하면 다층 퍼셉트론과 동일
    - 순환 신경망은 이 항을 통해 이전 순간의 은닉층 상태 h^{i-1}를 현재 순간의 은닉층 상태 h^i로 전달하여 시간성을 처리함
  - 순환 신경망의 학습 알고리즘을 BPTT(Back-propagation Through Time)라고 함

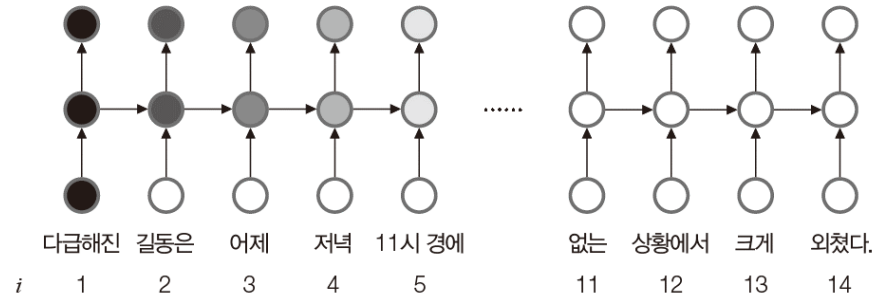
8.2.2 선별 기억력을 갖춘 LSTM

- 순환 신경망(RNN)은 이전 순간의 은닉층 상태를 다음 순간으로 넘기는 기능을 통해 과거를 기억하는 능력이 있으나, 장기 문맥 의존성을 제대로 처리하지 못하는 단점이 있음
- **장기 문맥 의존성 (long-term context dependency)**은 떨어져 있는 두 요소가 밀접한 상호작용을 하는 성질을 말하며, 많은 시계열 데이터가 장기 문맥 의존성을 가짐
- 장기 문맥 의존성을 제대로 처리하지 못하는 이유는 계속 들어오는 입력의 영향으로 기억력이 감퇴하는 것이며, 이는 시간이 지남에 따라 기억이 점점 희미해지는 사람 뇌에 비유할 수 있음



8.2.2 선별 기억력을 갖춘 LSTM

- 순환 신경망(RNN)은 이전 순간의 은닉층 상태를 다음 순간으로 넘기는 기능을 통해 과거를 기억하는 능력이 있으나, 장기 문맥 의존성을 제대로 처리하지 못하는 단점이 있음

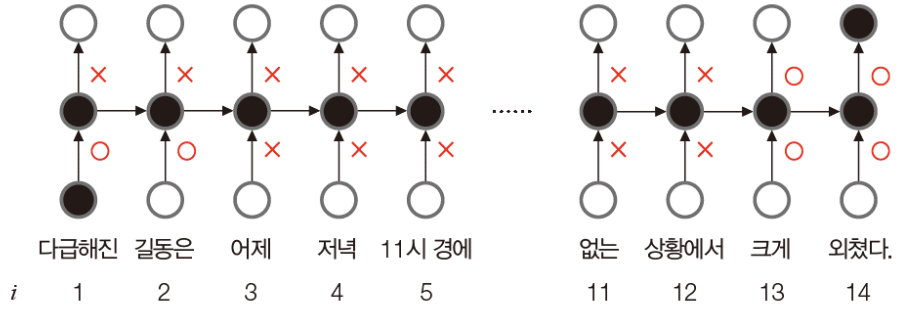


(a) 선별 기억 능력이 없는 순환 신경망

t=2 에 입력된 길동은"이라는 단어는 t=3,4,5 에는 기억이 남아 있지만, 이후에는 거의 사라지는 현상이 나타남

8.2.2 선별 기억력을 갖춘 LSTM

- 사람은 선별 기억(selective memory) 능력이 있어 인상적인 일은 아주 오래 전 기억도 간직함
- LSTM(long short-term memory)은 순환 신경망에 선별 기억 능력을 추가한 신경망임
- LSTM은 게이트(gate)라는 개념을 통해 선별 기억 능력을 확보함



(b) 선별 기억 능력을 지닌 LSTM

그림 8-7 순환 신경망과 LSTM

그림 8-7(b)는 순환신경망에 빨간 원으로 표시한 게이트를 추가한 LSTM임

O는 열린 상태의 게이트로서 신호를 전송하고, X는 닫힌 상태의 게이트로서 신호를 차단함

8.2.2 선별 기억력을 갖춘 LSTM

- [그림8-7(b)]는 LSTM을 아주 간단화한 설명임
- 실제로는 열거나 닫는 두 가지 상태만 있는 것이 아니라 여닫는 정도를 조절할 수 있도록 게이트는 0~1사이의 실수 값을 가짐
- 게이트의 여닫는 정도는 LSTM의 가중치로 표현되며 가중치 값은 학습으로 알아냄
- LSTM의 가중치는 순환 신경망의 {U, V, W}에 4개를 추가하여 {U, U^i, U^o, W, W^i, W^o, V}에 해당 되며, i는 입력 게이트, o는 출력 게이트를 의미함

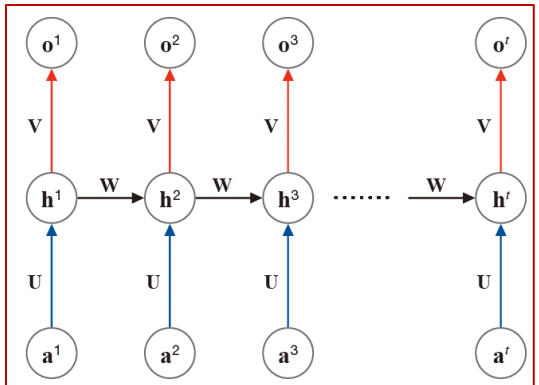
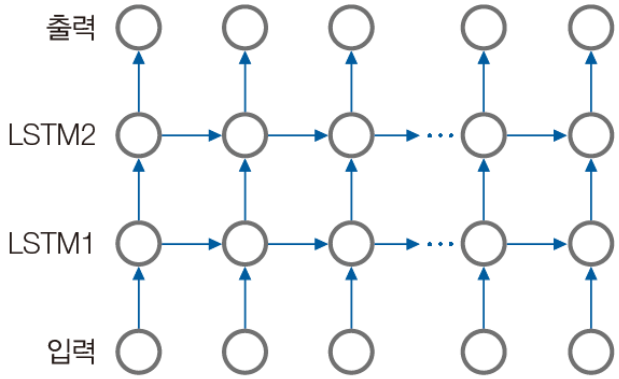


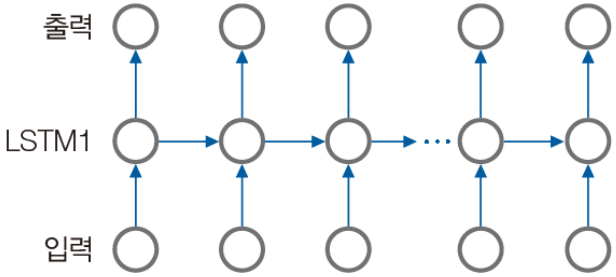
그림 8-6 펼친 순환 신경망

8.2.3 LSTM의 유연한 구조

- 순환 신경망은 다양하게 변형할 수 있음 (단층 ➔ 적층)
- [그림 8-8(a)]는 은닉층을 2개 쌓은 적층 LSTM (stacked LSTM) 임
- 적층 LSTM은 층을 깊게 쌓는 형태이며, 복잡도가 높아 효율적인 특징을 지닌다.



(a) 적층 LSTM



(a') 단층 LSTM

8.2.3 LSTM의 유연한 구조

- 순환 신경망은 다양하게 변형할 수 있음 (단방향 ➔ 양방향)
- 시계열 데이터 중에는 오른쪽 방향과 왼쪽 방향을 모두 살펴야 문맥을 제대로 파악할 수 있는 경우가 많음
- 예) "잘 **달리는** 이 **차**는 유럽에서 생산했다" vs "고산 지대에서 생산한 이 **차**는 **향**이 좋다"
- [그림 8-8(b)]는 첫 번째 은닉층에서는 원래처럼 오른쪽으로 데이터가 흐르고 두 번째 은닉층에서는 데이터가 왼쪽으로 흐름. 이런 신경망을 양방향 LSTM (bi-directional LSTM)이라고 함

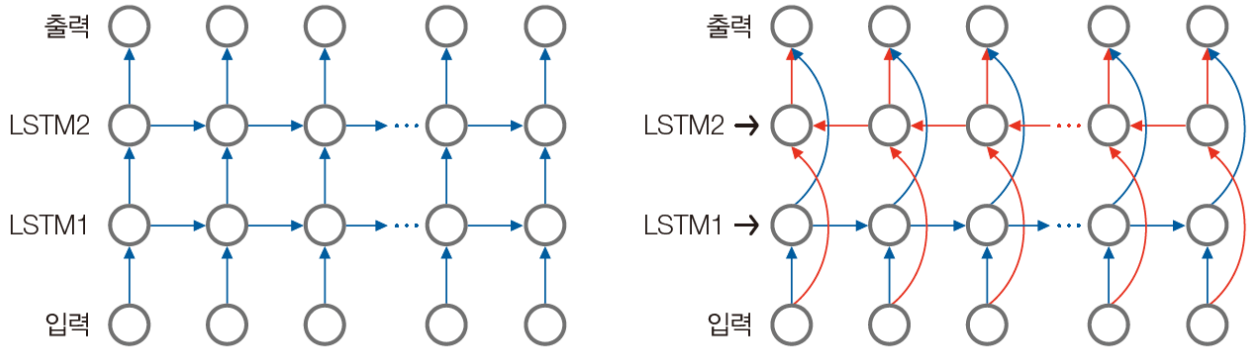


그림 8-8 적층 LSTM과 양방향 LSTM

8.2.3 LSTM의 유연한 구조

- [그림 8-9]는 여러 응용 문제를 위한 다양한 LSTM 신경망 구조를 보여줌
  - [그림 8-9(a)] 미래 예측, 비트코인 가격 예측, 심전도 데이터를 통한 환자 예측 문제에 적합한 신경망 구조
  - [그림 8-9(b)] 비디오를 구성하는 프레임별 영상 내용 분류 응용에 적합한 신경망 구조
  - [그림 8-9(c)] 언어 번역 응용에 적합한 신경망 구조
    - 앞부분에 한국어 문장을 구성하는 단어가 순서대로 입력되면 구문과 의미를 파악하고, 뒤에서는 영어 문장을 생성해 출력하는 방식

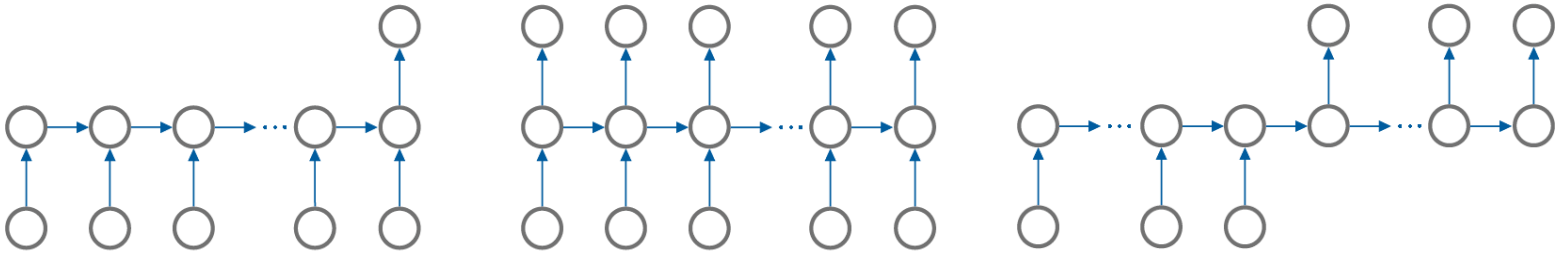


그림 8-9 다양한 문제에 적용할 수 있는 LSTM

8.3 LSTM으로 시계열 예측하기

- 이 절에서는 시계열 데이터를 보고 미래를 예측하는 프로그래밍 실습을 진행함
- [프로그램 8-1]에서 다룬 비트코인 가격 데이터를 가지고 LSTM을 학습하며, 성능 측정 기준에 따라 LSTM의 예측 능력을 평가함

- (1) 종가 정보만 있는 단일 채널 시계열 데이터에 적용
- (2) (종가, 시가, 고가, 저가)의 다중 채널 시계열 데이터에 적용

8.3.1 단일 채널 비트코인 가격 예측

- [그림 8-4(b)]를 보면 비트코인 가격 데이터에는 (종가, 시가, 고가, 저가)의 네 열이 있으며, 해당 데이터는 입력으로 모두 사용될 수 있음
- **하지만, 프로그램 8-2 에서는 종가만을 사용하여 입력이 1개인 단일 채널 데이터를 다루기로 함**

(a) 코인데스크에서 데이터를 다운로드하기

	A	B	C	D	E	F	G
1	Currency	Date	Closing Pr	24h Open	24h High	24h Low (USD)	
2	BTC	2019-02-28	3772.94	3796.64	3824.17	3666.52	
3	BTC	2019-03-01	3799.68	3773.44	3879.23	3753.8	
4	BTC	2019-03-02	3811.61	3799.37	3840.04	3788.92	
5	BTC	2019-03-03	3804.42	3806.69	3819.19	3759.41	
6	BTC	2019-03-04	3782.66	3807.85	3818.7	3766.24	
7	BTC	2019-03-05	3689.86	3783.36	3804.35	3663.48	
8	BTC	2019-03-06	3832.08	3701.05	3866.72	3688.7	
9	BTC	2019-03-07	3848.96	3832.59	3881.97	3802.52	
10	BTC	2019-03-08	3859.84	3848.96	3890.75	3827.67	
11	BTC	2019-03-09	3828.37	3859.8	3918	3778.52	
12	BTC	2019-03-10	3898.87	3841.89	3948.88	3832.14	
13	BTC	2019-03-11	3899.66	3916.76	3921.93	3865.92	
14	BTC	2019-03-12	3851.25	3899.46	3913.46	3819.93	

(b) 비트코인 가격이 저장된 데이터 프레임

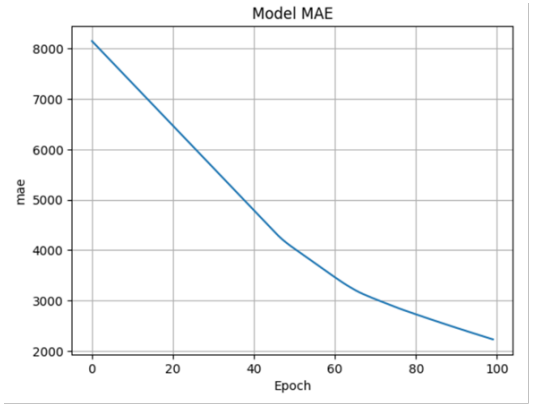
그림 8-4 코인데스크 사이트에서 다운로드한 비트코인 가격 데이터

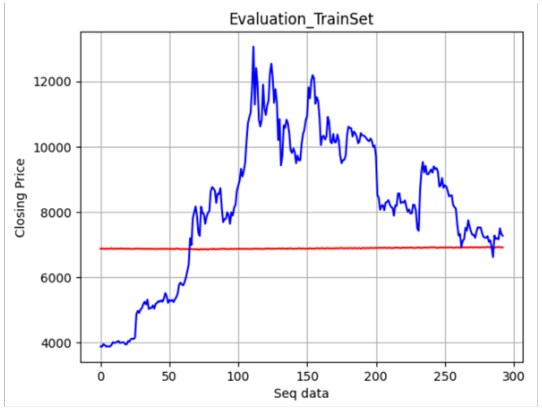
8.3.1 단일 채널 비트코인 가격 예측

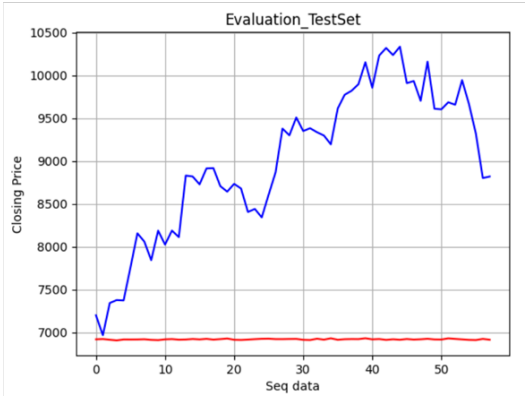
- [프로그램 8-2] 단일 채널 비트코인 가격 예측
  - 리더보드
    - https://www.kaggle.com/competitions/2024-1-dl-w-11-p-1-1-btc-price-prediction
  - (1) 파이토치 내 lstm을 사용하는 솔루션
    - https://www.kaggle.com/code/yukyungchoi/2024-dl-lstm-1-1
  - (2) 파이토치 내 lstm의 활성함수를 relu로 사용하는 솔루션
    - https://www.kaggle.com/code/yukyungchoi/2024-dl-lstm-1-3
  - (3) 파이토치 내 lstm을 사용하고 데이터 정규화를 사용하는 솔루션
    - https://www.kaggle.com/code/yukyungchoi/2024-dl-lstm-1-2

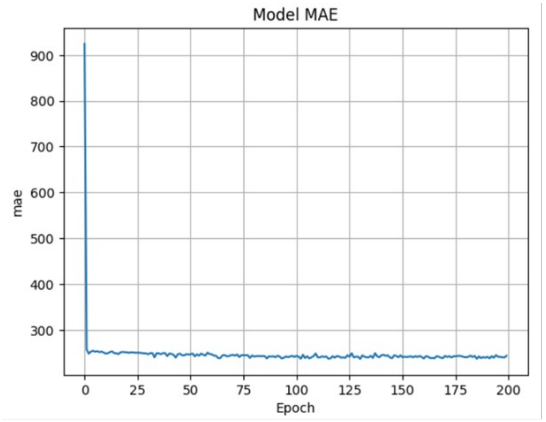
8.3.1 단일 채널 비트코인 가격 예측

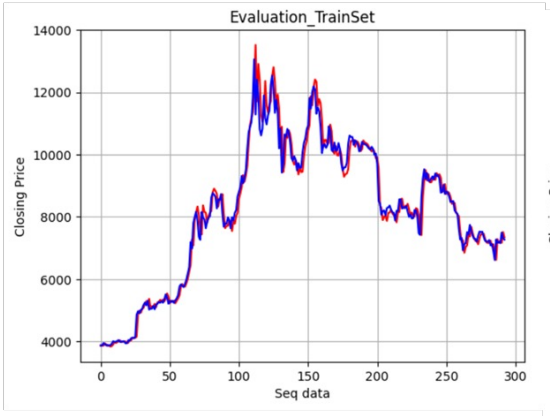
- Pytorch의 LSTM의 경우 내부 활성함수로 tanh를 사용 중이며, 이를 교안과 같이 ReLU로 커스터마이징할 경우 학습 결과는 아래와 같음

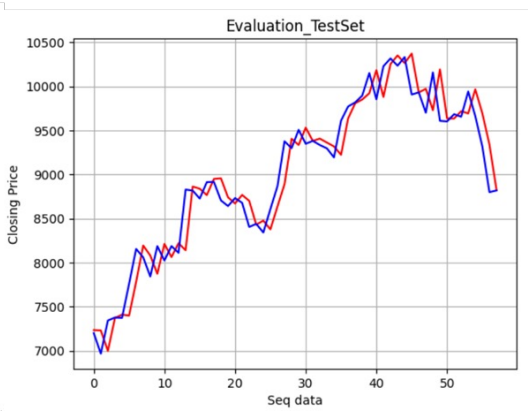












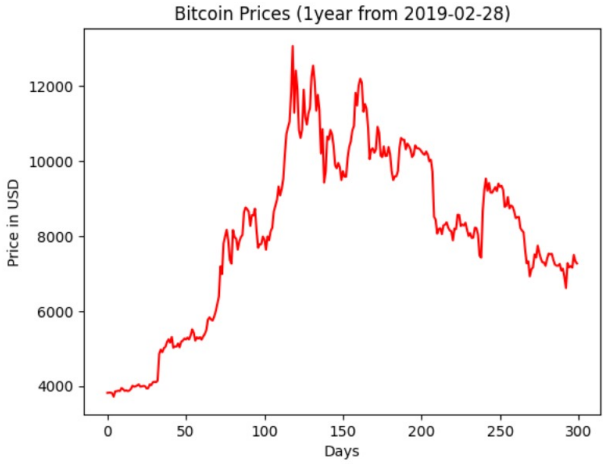
8.3.1 단일 채널 비트코인 가격 예측

- [프로그램 8-2] 단일 채널 비트코인 가격 예측 : **(1) 파이토치 내 lstm을 사용하는 솔루션**
  - 학습 및 테스트 데이터 준비

[17]:
```
pd_train = pd.read_csv("/kaggle/input/2024-1-dl-w-11-p-1-1-btc-price-prediction/traindata.csv")
pd_test = pd.read_csv("/kaggle/input/2024-1-dl-w-11-p-1-1-btc-price-prediction/testdata.csv")
```

[18]:
```
# 데이터 가공
np_train = pd_train[['closing price']].to_numpy()
np_test = pd_test[['closing price']].to_numpy()


import matplotlib.pyplot as plt
plt.plot(np_train,color='red')
plt.title('Bitcoin Prices (1year from 2019-02-28)')
plt.xlabel('Days')
plt.ylabel('Price in USD')
plt.show()
```



	A	B	C	D	E	F	G
1	Currency	Date	Closing Pr	24h Open	24h High	24h Low (USD)	
2	BTC	2019-02-28	3772.94	3796.64	3824.17	3666.52	
3	BTC	2019-03-01	3799.68	3773.44	3879.23	3753.8	
4	BTC	2019-03-02	3811.61	3799.37	3840.04	3788.92	
5	BTC	2019-03-03	3804.42	3806.69	3819.19	3759.41	
6	BTC	2019-03-04	3782.66	3807.85	3818.7	3766.24	
7	BTC	2019-03-05	3689.86	3783.36	3804.35	3663.48	
8	BTC	2019-03-06	3832.08	3701.05	3866.72	3688.7	
9	BTC	2019-03-07	3848.96	3832.59	3881.97	3802.52	
10	BTC	2019-03-08	3859.84	3848.96	3890.75	3827.67	
11	BTC	2019-03-09	3828.37	3859.8	3918	3778.52	
12	BTC	2019-03-10	3898.87	3841.89	3948.88	3832.14	
13	BTC	2019-03-11	3899.66	3916.76	3921.93	3865.92	
14	BTC	2019-03-12	3851.25	3899.46	3913.46	3819.93	

8.3.1 단일 채널 비트코인 가격 예측

- [프로그램 8-2] 단일 채널 비트코인 가격 예측
  - 윈도우에 맞게 시계열 데이터 준비 (w = 7, h = 1)

[19]:

```
# 학습데이터와 테스트데이터가 나눠 제공되기 때문에, 교재와 차이가 있음
def seq2dataset(seq, window, horizon, isTrain=True):
    """
    seq: np.array format
    window: sampling step
    """
    X=[]; Y=[]

    for i in range(len(seq)-(window+horizon)+1):
        x=seq[i:(i+window)]
        X.append(x);

        if isTrain:
            y=(seq[i+window+horizon-1])
            Y.append(y)

    if isTrain:
        return np.array(X), np.array(Y)
    else:
        return np.array(X)
```

+ Code + Markdown

[20]:

```
# 시계열 데이터 준비

w = 7
h = 1


X_train, Y_train = seq2dataset(np_train, w, h)
X_test = seq2dataset(np_test,w,h, isTrain=False)

print(X_train.shape, Y_train.shape)
print(X_test.shape)
```

```
(293, 7, 1) (293, 1)
(58, 7, 1)
```

[15…

	closing price	open price	high price	low price
0	3816.6	3814.6	3883.7	3783.3
1	3821.9	3816.7	3855.8	3816.4
2	3823.1	3821.9	3843.2	3783.6
3	3809.5	3823.2	3836.6	3789.7
4	3715.9	3809.7	3828.4	3681.8
5	3857.2	3715.9	3873.2	3705.7
6	3863.0	3857.2	3887.3	3816.7
7	3875.1	3863.1	3907.4	3847.9
8	3865.9	3875.1	3929.0	3810.7
9	3944.3	3865.6	3964.0	3859.7

Train_X

```
array([[[3816.6, 3814.6, 3883.7, 3783.3],
        [3821.9, 3816.7, 3855.8, 3816.4],
        [3823.1, 3821.9, 3843.2, 3783.6],
        ...,
        [3715.9, 3809.7, 3828.4, 3681.8],
        [3857.2, 3715.9, 3873.2, 3705.7],
        [3863. , 3857.2, 3887.3, 3816.7]],

       [[3821.9, 3816.7, 3855.8, 3816.4],
        [3823.1, 3821.9, 3843.2, 3783.6],
        [3809.5, 3823.2, 3836.6, 3789.7],
        ...,
        [3857.2, 3715.9, 3873.2, 3705.7],
        [3863. , 3857.2, 3887.3, 3816.7],
        [3875.1, 3863.1, 3907.4, 3847.9]],

       [[3823.1, 3821.9, 3843.2, 3783.6],
        [3809.5, 3823.2, 3836.6, 3789.7],
        [3715.9, 3809.7, 3828.4, 3681.8],
        ...,
        [3863. , 3857.2, 3887.3, 3816.7],
        [3875.1, 3863.1, 3907.4, 3847.9],
        [3865.9, 3875.1, 3929. , 3810.7]],

       ...,
```

Train_Y

[32…

```
array([[3875.1, 3863.1, 3907.4, 3847.9],
       [3865.9, 3875.1, 3929. , 3810.7],
       [3944.3, 3865.6, 3964. , 3859.7],
       ...,
       [7495.8, 7156.3, 7501.1, 7142. ],
       [7322.8, 7496.2, 7684. , 7276.2],
       [7268.3, 7322.4, 7433.2, 7185.7]])
```

8.3.1 단일 채널 비트코인 가격 예측

- [프로그램 8-2] 단일 채널 비트코인 가격 예측
  - 모델 정의

[10]:

```python
# LSTM 모델 정의 : input 4, hidden 128, output 4
# 단일 LSTM 모델 : num_layers = 1

class LSTMModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, num_layers=1):
        super().__init__()
        self.num_layers = num_layers
        self.hidden_dim = hidden_dim
        self.lstm = nn.LSTM(input_dim, self.hidden_dim, num_layers=self.num_layers, batch_first=True)
        self.fc = nn.Linear(self.hidden_dim, output_dim, bias=True)

    def forward(self, x):

        self.h0 = torch.randn(self.num_layers, x.size(0), self.hidden_dim).cuda()
        self.c0 = torch.randn(self.num_layers, x.size(0), self.hidden_dim).cuda()
        x, _ = self.lstm(x, (self.h0, self.c0))

        x = x[:, -1, :]
        x = self.fc(x)
        return x
```

Num_layers = 2 이상이 되면, stacked LSTM 모델이 됨

+ Code + Markdown

8.3.1 단일 채널 비트코인 가격 예측

- [프로그램 8-2] 단일 채널 비트코인 가격 예측
  - 모델 학습을 위한 데이터 배치형 가공

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, TensorDataset
from torch.utils.data import DataLoader

# 데이터 토치텐서로 변경
train_dataset = TensorDataset(torch.FloatTensor(X_train), torch.FloatTensor(Y_train))
test_dataset = TensorDataset(torch.FloatTensor(X_test))


# 데이터 배치형 데이터로 변경
# DataLoader 파라미터 shuffle은 default 값이 false
train_loader = DataLoader(train_dataset, batch_size=1)
test_loader = DataLoader(test_dataset, batch_size=1)


# 배치형 데이터 확인법
#train_features, train_labels = next(iter(train_loader))
```

8.3.1 단일 채널 비트코인 가격 예측

- [프로그램 8-2] 단일 채널 비트코인 가격 예측
  - 모델 학습

```
# 모델 생성
model = LSTMModel(input_dim=1, hidden_dim=128, output_dim=1)
criterion = nn.L1Loss(reduction='mean')
optimizer = optim.Adam(model.parameters(),lr=1e-2, eps=1e-7)


# 학습 준비
epoch = 100
model = model.cuda()

hist = []
display_Y = []

# 학습 루프
for epoch in range(epoch):
    # 데이터 준비
    avg_loss = 0
    model.train()
    for idx, data in enumerate(train_loader):

      train_X = data[0].cuda()
      train_Y = data[1].cuda()

      # 모델 예측
      pred_Y = model(train_X)

     # 예측 값 시각화를 위한 저장
      if epoch == 99:
            display_Y.append(pred_Y.cpu().detach().numpy())


      # 손실 계산 및 역전파
      loss = criterion(pred_Y, train_Y)
      optimizer.zero_grad()
      loss.backward()
      avg_loss += loss

    # 최적화 수행
      optimizer.step()


    hist.append(avg_loss.cpu().detach().numpy()/len(train_loader))

    # 손실 출력 (인터벌 5로 출력)
    if epoch%5==0:
        print(f"Epoch {epoch + 1} | Loss: {(avg_loss/len(train_loader)).item():.4f}")
```

8.3.1 단일 채널 비트코인 가격 예측

- [프로그램 8-2] 단일 채널 비트코인 가격 예측
  - 학습된 LSTM 모델을 이용한 테스트 데이터 예측

[29]:

```
model.eval()


results = []
with torch.no_grad():
    for idx, data in enumerate(test_loader):
        test_X = data[0].cuda()
        pred_Y = model(test_X)

        # 아래 타입변경과 쌍임
        results.append(pred_Y.cpu())
        #print(test_X[-1].mean(),pred_Y.mean())


# 타입변경
results = np.array(results).reshape(-1,1)
```

8.3.1 단일 채널 비트코인 가격 예측

- [프로그램 8-2] 단일 채널 비트코인 가격 예측
  - 캐글 리더보드 제출을 위한 csv 파일 만들기

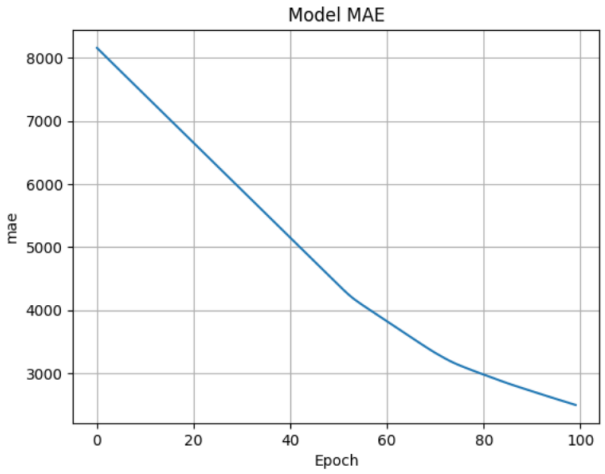
[30]:

```
# 결과 저장 및 제출 파일 준비
submit = pd.read_csv("/kaggle/input/2024-1-dl-w-11-p-1-1-btc-price-prediction/sample_submit.csv")
submit[['closing price']] = results
submit.to_csv("result.csv",index=False)
```

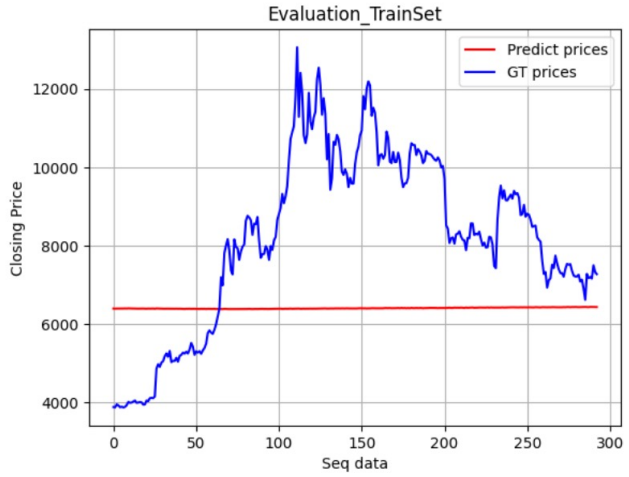
8.3.1 단일 채널 비트코인 가격 예측

- [프로그램 8-2] 단일 채널 비트코인 가격 예측
  - 시각화

[27]:
```
# 학습 과정 내 손실 값 시각화
plt.plot(hist)
plt.title('Model MAE')
plt.xlabel('Epoch')
plt.ylabel('mae')
plt.grid()
plt.show()
```



[28]:
```
# 학습 데이터 종가과 예측된 종가 시각화
plt.title('Evaluation_TrainSet')
plt.xlabel('Seq data')
plt.ylabel('Closing Price')
plt.plot(np.array(display_Y).reshape(-1), color='red')
plt.plot(np.array(Y_train).reshape(-1), color='blue')
plt.legend(['Predict prices', 'GT prices'], loc='best')
plt.grid()
plt.show()
```



8.3.1 단일 채널 비트코인 가격 예측

- [프로그램 8-2] 단일 채널 비트코인 가격 예측 :
  - (2) 파이토치 내 lstm의 활성함수를 relu로 사용하는 솔루션
    - https://www.kaggle.com/code/yukyungchoi/2024-dl-lstm-1-3

  - 모델 학습

```
# LSTM 모델 정의 : input 1, hidden 128, output 1
# 단일 LSTM 모델 : num_layers = 1

class LSTMModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, num_layers=1):
        super().__init__()
        self.num_layers = num_layers
        self.hidden_dim = hidden_dim
        #
        # self.lstm = nn.LSTM(input_dim, self.hidden_dim, num_layers=self.num_layers, batch_first=True)
        self.lstm = CustomLSTMCell(input_dim, self.hidden_dim, self.num_layers, dropout=0, activation='relu')
        self.fc = nn.Linear(self.hidden_dim, output_dim, bias=True)

    def forward(self, x):

        self.h0 = torch.randn(self.num_layers, x.size(0), self.hidden_dim).cuda()
        self.c0 = torch.randn(self.num_layers, x.size(0), self.hidden_dim).cuda()
        x, _ = self.lstm(x, (self.h0, self.c0))

        if len(x.shape)==4:
            x = x.squeeze(0)

        x = x[:, -1, :]
        x = self.fc(x)
        return x
```

8.3.1 단일 채널 비트코인 가격 예측

- [프로그램 8-2] 단일 채널 비트코인 가격 예측 :
  - 모델 학습 (모델 정의)

```
# ReLU 기반 LSTM 모델 정의 : input 1, hidden 128, output 1
# ReLU 기반 단일 LSTM 모델 : num_layers = 1

import torch.nn.functional as F

class CustomLSTMCell(nn.Module):
    def __init__(self, input_size, hidden_size, nlayers, dropout, activation='tanh'):
        """Constructor of the class"""
        super(CustomLSTMCell, self).__init__()

        self.nlayers = nlayers
        self.dropout = nn.Dropout(p=dropout)

        ih, hh = [], []

        for i in range(nlayers):
            ih.append(nn.Linear(input_size, 4 * hidden_size))
            hh.append(nn.Linear(hidden_size, 4 * hidden_size))

        self.w_ih = nn.ModuleList(ih)
        self.w_hh = nn.ModuleList(hh)

        if activation =='relu':
            self.outputact = F.relu
        elif activation == 'tanh':
            self.outputact = F.tanh
        else:
            assert print("No defined activation function")
```

```
def forward(self, input, hidden):
    """Defines the forward computation of the LSTMCell"""
    hy, cy = [], []

    for i in range(self.nlayers):
        B, L, D = input.shape
        input = input.view(B*L, -1)

        hx, cx = hidden[0][i], hidden[1][i].view(B, 1, -1)

        gates = self.w_ih[i](input).view(B,L,-1) + self.w_hh[i](hx).view(B,1,-1)
        i_gate, f_gate, c_gate, o_gate = gates.chunk(4, 2)

        i_gate = F.sigmoid(i_gate)
        f_gate = F.sigmoid(f_gate)
        c_gate = self.outputact(c_gate)
        o_gate = F.sigmoid(o_gate)

        ncx = (f_gate * cx) + (i_gate * c_gate)
        nhx = o_gate * self.outputact(ncx)
        cy.append(ncx)
        hy.append(nhx)
        input = self.dropout(nhx)

    hy, cy = torch.stack(hy, 0), torch.stack(cy, 0)

    return hy, cy
```

8.3.1 단일 채널 비트코인 가격 예측

- [프로그램 8-2] 단일 채널 비트코인 가격 예측 :
  - (3) 파이토치 내 lstm을 사용하고 데이터 정규화를 사용하는 솔루션
    - https://www.kaggle.com/code/yukyungchoi/2024-dl-lstm-1-2

  - 데이터 정규화

In [5]:

```
# 데이터 가공
np_train = pd_train[['closing price']].to_numpy()
np_test = pd_test[['closing price']].to_numpy()


# 데이터 전처리
scaler = Scaler()
scaler.set_(X=np_train, ax=0)
np_train = scaler.MinMaxScaler(np_train)
np_test = scaler.MinMaxScaler(np_test)


import matplotlib.pyplot as plt
plt.plot(np_train,color='red')
plt.title('Bitcoin Prices (1year from 2019-02-28)')
plt.xlabel('Days')
plt.ylabel('Price in USD')
plt.show()
```

8.3.1 단일 채널 비트코인 가격 예측

- [프로그램 8-2] 단일 채널 비트코인 가격 예측 :
  - 데이터 정규화

In [4]:

```
class Scaler:
    def __init__(self):
        self.max_ = 0
        self.min_ = 0
        self.ax = 0

    def set_(self, X, ax):
        self.ax = ax
        self.max_ = X.max(self.ax)
        self.min_ = X.min(self.ax)

    def MinMaxScaler(self, X):
        return (X - self.min_) / (self.max_ - self.min_)

    def inverseMinMaxScaler(self, X_pp):
        return (X_pp * (self.max_ - self.min_)) + self.min_
```

8.3.2 성능 평가

- 평균절댓값오차mean absolute error(MAE): 스케일 문제에 대처하지 못함
  - 농산물의 판매량 예측에서 복숭아 MAE 1.8 의 결과가 수박 MAE 18 보다 성능이 뛰어나다고 말할 수 없음

$$\text{평균절댓값오차: } \mathrm{MAE} = \frac{1}{|M|}\sum_{x\in M}|y-o| \qquad (8.4)$$

- 평균절댓값백분율오차: 스케일 문제에 대처

$$\text{평균절댓값백분율오차: } \mathrm{MAPE} = \frac{1}{|M|}\sum_{x\in M}\left|\frac{y-o}{y}\right| \qquad (8.5)$$

표 8-1 평균절댓값오차와 평균절댓값백분율오차

데이터 스케일	데이터(y_test는 참값, pred는 예측값)	평균절댓값오차(MAE)	평균절댓값백분율오차(MAPE)
두 자릿수	y_test [12 20 18 22 28]	(2+1+2+2+2)/5 =1.8	(2/12+1/20+2/18+2/22+2/28)/5 =0.0980
	pred [10 21 16 20 26]		
세 자릿수	y_test [120 200 180 220 280]	(20+10+20+20+20)/5 =18	(20/120+10/200+20/180+20/220+ 20/280)/5=0.0980
	pred [100 210 160 200 260]		

8.3.2 성능 평가

- 등락 정확률
  - 등락을 얼마나 정확하게 맞히는지 측정
  - 맞힌 경우의 수를 전체 샘플 수로 나눔

표 8-2 등락 정확률

샘플 i	x_test[i]	y_test[i]	pred[i]	맞힘
1	[.., .., .., .., 21]	23 업	24 업	o
2	[.., .., .., .., 25]	20 다운	26 업	x
3	[.., .., .., .., 22]	24 업	23 업	o
4	[.., .., .., .., 28]	25 다운	26 다운	o
5	[.., .., .., .., 21]	18 다운	17 다운	o
6	[.., .., .., .., 32]	31 다운	33 업	x
7	[.., .., .., .., 35]	36 업	37 업	o
8	[.., .., .., .., 20]	19 다운	22 업	x
				등락 정확률 = 5/8 = 62.5%

8.3.3 다중 채널 비트코인 가격 예측

- [그림 8-4(b)]를 보면 비트코인 가격 데이터에는 (종가, 시가, 고가, 저가)의 네 열이 있으며, 해당 데이터는 입력으로 모두 사용될 수 있음
- 프로그램 8-3 에서는 입력이 4개인 다중 채널 데이터를 다루기로 함

(a) 코인데스크에서 데이터를 다운로드하기

	A	B	C	D	E	F	G
1	Currency	Date	Closing Pr	24h Open	24h High	24h Low (USD)	
2	BTC	2019-02-28	3772.94	3796.64	3824.17	3666.52	
3	BTC	2019-03-01	3799.68	3773.44	3879.23	3753.8	
4	BTC	2019-03-02	3811.61	3799.37	3840.04	3788.92	
5	BTC	2019-03-03	3804.42	3806.69	3819.19	3759.41	
6	BTC	2019-03-04	3782.66	3807.85	3818.7	3766.24	
7	BTC	2019-03-05	3689.86	3783.36	3804.35	3663.48	
8	BTC	2019-03-06	3832.08	3701.05	3866.72	3688.7	
9	BTC	2019-03-07	3848.96	3832.59	3881.97	3802.52	
10	BTC	2019-03-08	3859.84	3848.96	3890.75	3827.67	
11	BTC	2019-03-09	3828.37	3859.8	3918	3778.52	
12	BTC	2019-03-10	3898.87	3841.89	3948.88	3832.14	
13	BTC	2019-03-11	3899.66	3916.76	3921.93	3865.92	
14	BTC	2019-03-12	3851.25	3899.46	3913.46	3819.93	

(b) 비트코인 가격이 저장된 데이터 프레임

그림 8-4 코인데스크 사이트에서 다운로드한 비트코인 가격 데이터

8.3.3 다중 채널 비트코인 가격 예측

- [프로그램 8-3] 다중 채널 비트코인 가격 예측
  - 리더보드
    - https://www.kaggle.com/competitions/2024-1-dl-w-11-btc-price-prediction/overview
  - 단일 채널 비트코인 가격 예측 문제를 기반으로 해결 하시오

8.3.3 다중 채널 비트코인 가격 예측

- [프로그램 8-3] 다중 채널 비트코인 가격 예측
  - 학습 및 테스트 데이터 준비

```
pd_train = pd.read_csv("/kaggle/input/2024-1-dl-w-11-btc-price-prediction/traindata.csv")
pd_test = pd.read_csv("/kaggle/input/2024-1-dl-w-11-btc-price-prediction/testdata.csv")

#print(pd_train.shape, pd_test.shape)
```

```
(300, 5) (65, 5)
```

+ Code + Markdown

[19]:

```
# 학습 데이터 확인 후 불필요한 정보 삭제
pd_train.head(3)
```

	A	B	C	D	E	F	G
1	Currency	Date	Closing Pr	24h Open	24h High	24h Low (USD)	
2	BTC	2019-02-28	3772.94	3796.64	3824.17	3666.52	
3	BTC	2019-03-01	3799.68	3773.44	3879.23	3753.8	
4	BTC	2019-03-02	3811.61	3799.37	3840.04	3788.92	
5	BTC	2019-03-03	3804.42	3806.69	3819.19	3759.41	
6	BTC	2019-03-04	3782.66	3807.85	3818.7	3766.24	
7	BTC	2019-03-05	3689.86	3783.36	3804.35	3663.48	
8	BTC	2019-03-06	3832.08	3701.05	3866.72	3688.7	
9	BTC	2019-03-07	3848.96	3832.59	3881.97	3802.52	
10	BTC	2019-03-08	3859.84	3848.96	3890.75	3827.67	
11	BTC	2019-03-09	3828.37	3859.8	3918	3778.52	
12	BTC	2019-03-10	3898.87	3841.89	3948.88	3832.14	
13	BTC	2019-03-11	3899.66	3916.76	3921.93	3865.92	
14	BTC	2019-03-12	3851.25	3899.46	3913.46	3819.93	

+ Code + Markdown

[5]:

```
pd_train = pd_train.drop('date', axis=1)
pd_test = pd_test.drop('date', axis=1)
print(pd_train.shape, pd_test.shape)
```

```
(300, 4) (65, 4)
```

8.3.3 다중 채널 비트코인 가격 예측

- [프로그램 8-3] 다중 채널 비트코인 가격 예측
  - 윈도우에 맞게 시계열 데이터 준비 (w = 7, h = 1)

```
# 학습데이터와 테스트데이터가 나눠 제공되기 때문에, 교재와 차이가 있음
def seq2dataset(seq, window, horizon, isTrain=True):
    """
    seq: np.array format
    window: sampling step
    """
    X=[]; Y=[]

    for i in range(len(seq)-(window+horizon)+1):
        x=seq[i:(i+window)]
        X.append(x);

        if isTrain:
            y=(seq[i+window+horizon-1])
            Y.append(y)

    if isTrain:
        return np.array(X), np.array(Y)
    else:
        return np.array(X)
```

+ Code + Markdown

[7]:

```
w = 7
h = 1

np_train = pd_train.to_numpy()
np_test = pd_test.to_numpy()

X_train, Y_train = seq2dataset(np_train, w, h)
X_test = seq2dataset(np_test,w,h, isTrain=False)

print(X_train.shape, Y_train.shape)
print(X_test.shape)
```

```
(293, 7, 4) (293, 4)
(58, 7, 4)
```

[15...

	closing price	open price	high price	low price
0	3816.6	3814.6	3883.7	3783.3
1	3821.9	3816.7	3855.8	3816.4
2	3823.1	3821.9	3843.2	3783.6
3	3809.5	3823.2	3836.6	3789.7
4	3715.9	3809.7	3828.4	3681.8
5	3857.2	3715.9	3873.2	3705.7
6	3863.0	3857.2	3887.3	3816.7
7	3875.1	3863.1	3907.4	3847.9
8	3865.9	3875.1	3929.0	3810.7
9	3944.3	3865.6	3964.0	3859.7

Train_X

```
array([[[3816.6, 3814.6, 3883.7, 3783.3],
        [3821.9, 3816.7, 3855.8, 3816.4],
        [3823.1, 3821.9, 3843.2, 3783.6],
        ...,
        [3715.9, 3809.7, 3828.4, 3681.8],
        [3857.2, 3715.9, 3873.2, 3705.7],
        [3863. , 3857.2, 3887.3, 3816.7]],

       [[3821.9, 3816.7, 3855.8, 3816.4],
        [3823.1, 3821.9, 3843.2, 3783.6],
        [3809.5, 3823.2, 3836.6, 3789.7],
        ...,
        [3857.2, 3715.9, 3873.2, 3705.7],
        [3863. , 3857.2, 3887.3, 3816.7],
        [3875.1, 3863.1, 3907.4, 3847.9]],

       [[3823.1, 3821.9, 3843.2, 3783.6],
        [3809.5, 3823.2, 3836.6, 3789.7],
        [3715.9, 3809.7, 3828.4, 3681.8],
        ...,
        [3863. , 3857.2, 3887.3, 3816.7],
        [3875.1, 3863.1, 3907.4, 3847.9],
        [3865.9, 3875.1, 3929. , 3810.7]],

       ...,
```

Train_Y

[32...

```
array([[3875.1, 3863.1, 3907.4, 3847.9],
       [3865.9, 3875.1, 3929. , 3810.7],
       [3944.3, 3865.6, 3964. , 3859.7],
       ...,
       [7495.8, 7156.3, 7501.1, 7142. ],
       [7322.8, 7496.2, 7684. , 7276.2],
       [7268.3, 7322.4, 7433.2, 7185.7]])
```