R E P O R T

[인공지능 과제 1]

• 과 목 명	: 인 공 지 능
• 담 당 교 수	: 김동근 교수님
• 제 출 일	: 2023. 10. 16.
• 학 과	: 컴퓨터 공학과
• 학 번	: 201601848
• 성 명	: 이 성 남

1. 서론

- 특정지역(천안, 서울 등)의 최근 1 년치 온도 데이터(월 또는 일)데이터를 구하여 n – 차 다항식 회귀 문제를 텐서플로를 이용하여 구현하고 예측해보기.

- 기상청에서 CSV 파일을 불러와 pandas 라이브러리를 활용하여 데이터를 가져온 후, x 에 날짜 y 에 온도를 두고 텐서플로를 활용하여 신경망을 만들고 학습시켜서 손실율을 줄여나갔습니다.

2. 본론

2.1. 데이터 수집 방법

- 데이터를 수집하기 위해서 기상청의 기상자료개방포털(https://data.kma.go.kr/climate/RankState/selectRankStatisticsDivisionList.do) 사이트를 통해서 천안 지역의 22 년도 1 년치 기온(데이터)를 수집 하였습니다.

2.2. 구현 설명

- 수집한 데이터를 pandas 라이브러리를 통하여 해당 데이터 값을 가져와 x 값에는 날짜를 y 값에는 평균기온을 npArray 로 가져왔습니다. 그리고 더욱 정확한 예측을 위해서 정규화를 진행하였습니다. 그리고 Input 층과 Dense 층 2 개의 층을 가진 모델을 만들고 학습하여 실행한 결과를 pyplot 을 통하여 그래프로 출력하였습니다.

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

# 데이터에 csv파일 데이터를 불러온다. encoding을 통해 한글을 불러올수있게한다. 칼럼의 헤더는 7번라인
data = pd.read_csv(r'C:₩Users₩dltjd₩Artificial Intelligence₩month_temp.csv', encoding = 'cp949', header=6)

# csv파일의 년월 중 년도는 필요하지 않고 월만 필요하기 때문에 월의 값만 가지고온다.
x_month = data['년월'].str.split("-").str[1].astype(float).values

# csv 파일에서 월별 평균기온을 가져온다. npArray로 가져옴.
y_temp = data['평균기온(℃)'].values
```

➔ 기상자료개방포털에서 22 년도월별 기온데이터를 가져왔습니다.

x_month 에 날짜를 배열로 받아왔습니다. (1 월부터 12 월 까지)

y_temp 에는 온도를 배열로 받아왔습니다 (1 월부터 12 월까지의 평균온도)

```
# 정규화
x_month /= max(x_month)
y_temp /= max(y_temp)
```

➔ 손실율을 줄이기 위해서 받아온 배열의 최댓값으로 정규화를 진행했습니다.

```
# n-차 다항식 회귀
n = 3

# x의 데이터갯수 (20개) x n + 1개 (4개)의 배열을 만들고 값을 1로 초기화
X = np.ones(shape = (len(x_month), n+1), dtype=np.float32)

#0번째 열을 1로 초기화하고 배열 X에서 1번째 열부터 n번째 열까지 x^i 값을 차례로 넣어준다.
for i in range(1, n+1):
    X[:, i] = x_month**i

#신경망 입력 정의 n+1은 입력 데이터의 형태를 나타냄.
inputs = tf.keras.layers.Input(shape=(n+1,))
# 신경망의 출력을 정의 units = 1은 뉴런 1개 bias는 없음을 뜻함 bias는 없어도 될수있다.
# 왜냐하면, 상수항이 X배열의 1열에 1이 초기화 되어 있기때문이다.
# (inputs) 이 Dense 층의 입력은 앞에서 정의한 input 사용하겠다는
outputs = tf.keras.layers.Dense(units=1, use_bias=False)(inputs)
# 실제 신경망을 만든다.
model = tf.keras.Model(inputs=inputs, outputs=outputs)
# 신경망을 출력
model.summary()
```

➔ N- 차 다항식 회귀를 위해 x_month 값으로 배열을 만들고 신경망을 만들었습니다.

```
#rmsprop은 옵티마이저의 알고리즘의 하나  학습률은 0.1 을 opt에 저장
opt = tf.keras.optimizers.RMSprop(learning_rate = 0.1)
#옵티마이저의 알고리즘하나인 rmsprop을 저장한 opt를 옵티마이저를 사용한다.
#그리고 mse는 sum((트루값 - 예측값)^2) / t.size()
model.compile(optimizer=opt, loss='mse')

#모델을 학습하는 함수 fit() X는 입력데이터 , y_month는 정답 , 10000번 반복 ,
#batch_size를 통해 x의 값을 쪼갠다 10000번의 포문안에 6번의 포문이 계속해서 돈다.
#verbose = 2 한줄씩 출력 0은 결과만 출력
ret = model.fit(X, y_temp, epochs = 4000, batch_size=2, verbose = 2)
```

➔ 손실율을 최소화하기 위해서 RMSprop 을 사용하였고 , fit() 함수를 통해 epochs 를 4000 으로 두고 batch-size 를 2 로하여 모델을 학습시켰습니다.

```
#1: 모델 전체 저장
import os
if not os.path.exists("./RES1000"):
     os.mkdir("./RES1000")
model.save("./RES1000/1401.keras")   # HDF5, keras format

#2: 모델 구조 저장
json_string = model.to_json()
import json
file = open("./RES1000/1401.model", 'w')
json.dump(json_string, file)
file.close()

#3: 가중치 저장
model.save_weights("./RES1000/weights/1401")

#4: 학습중에 체크포인트 저장
filepath = "RES1000/ckpt/1401-{epoch:04d}.ckpt"
cp_callback = tf.keras.callbacks.ModelCheckpoint(
              filepath, verbose=0, save_weights_only=True, save_freq=50)
ret = model.fit(X, y_temp, epochs=100, callbacks = [cp_callback], verbose=0)




print("len(model.layers):", len(model.layers)) # 2

loss = ret.history['loss']
print("loss:", loss[-1])
#print(model.get_weights())  # weights
print("weights:", model.layers[1].weights[0].numpy())

plt.plot(loss)
plt.xlabel('epochs')
plt.ylabel('loss')
plt.show()

plt.scatter(x_month, y_temp)
y_pred = model.predict(X)
plt.plot(x_month, y_pred, color='red')
plt.show()
```

➔ 모델구조, 가중치 등을 저장한 후, 터미널창에 손실율, 가중치등을 출력하고 pyplot 을 통하여 그래프를 보여줬습니다.

.

3. 실험

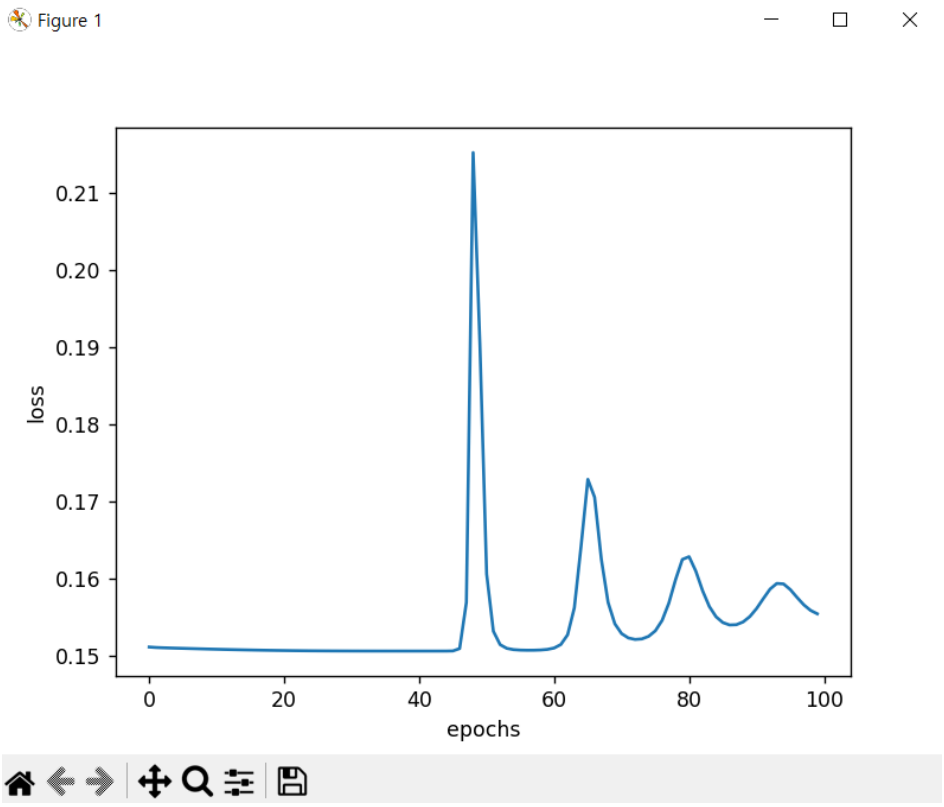


Figure 1
loss
epochs
0.21
0.20
0.19
0.18
0.17
0.16
0.15
0
20
40
60
80
100

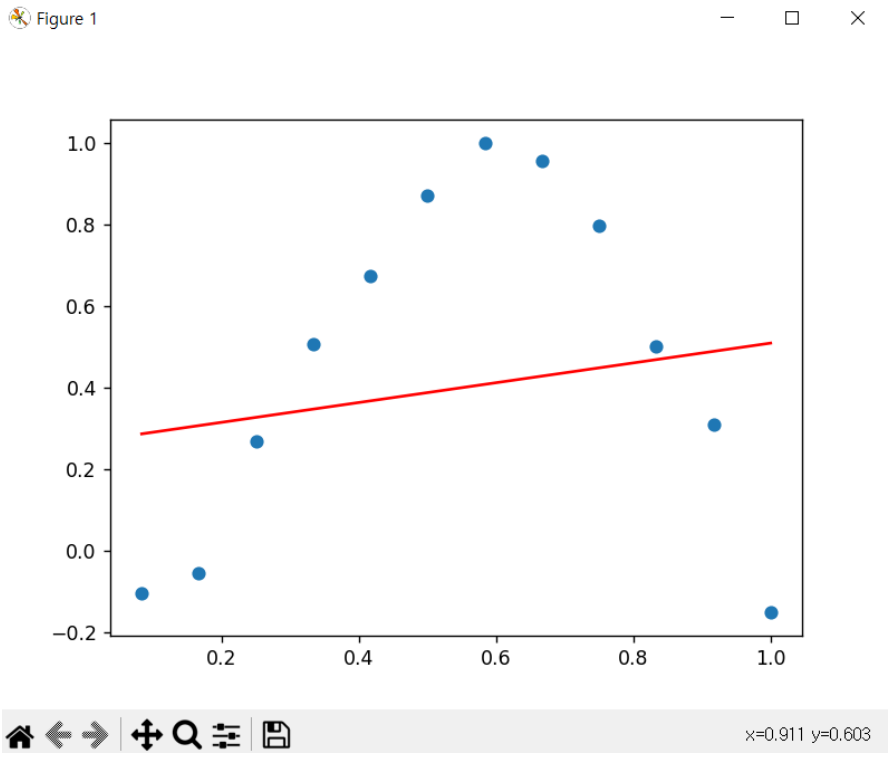
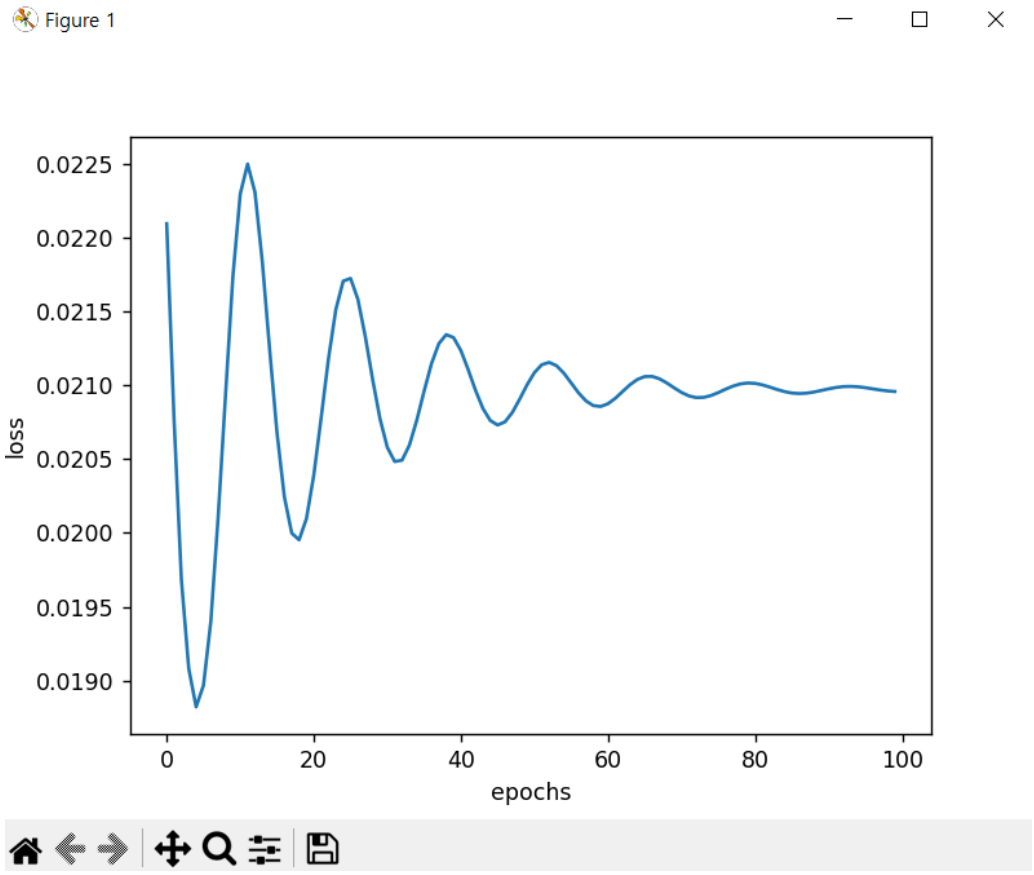
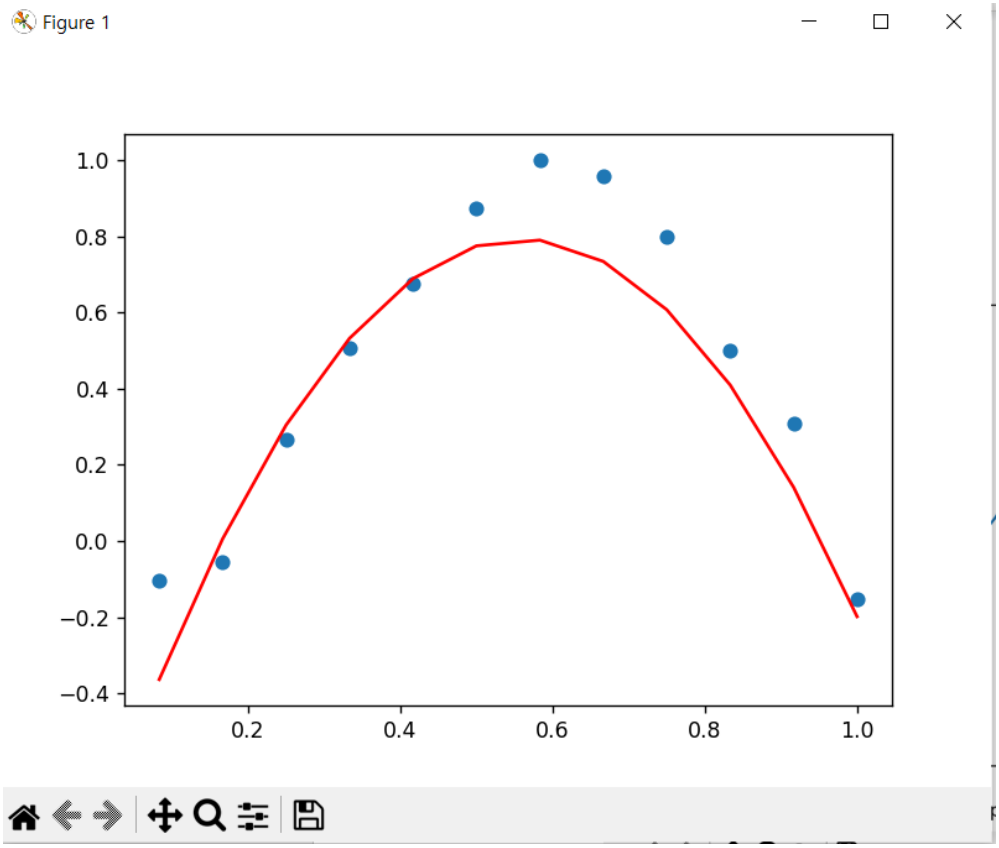


Figure 1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.2
0.4
0.6
0.8
1.0
x=0.911 y=0.603

```
len(model.layers): 2
loss: 0.1554759293794632
weights: [[0.26617476]
 [0.24291933]]
```

➔ 1 차 다항식 회귀를 실행한 손실율 그래프와 예측 그래프, 그리고 모델의 계층 제일 작은 손실율 , 가중치 출력






```
len(model.layers): 2
loss: 0.020958354696631 43
weights: [[-0.8033271]
 [ 5.7076516]
 [-5.102378 ]]
```

➔ 2 차 다항식 회귀를 실행한 손실율 그래프와 예측 그래프, 그리고 모델의 계층 제일 작은 손실율 , 가중치 출력

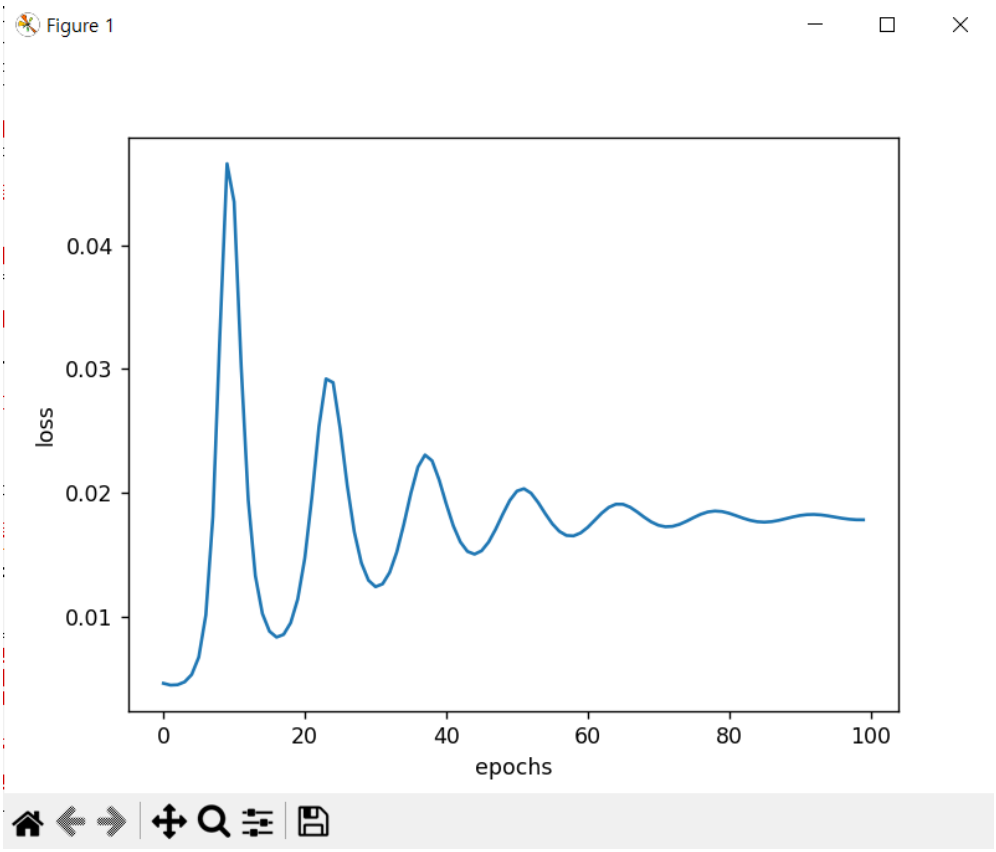


Figure 1
0.04
0.03
0.02
0.01
loss
0
20
40
60
80
100
epochs

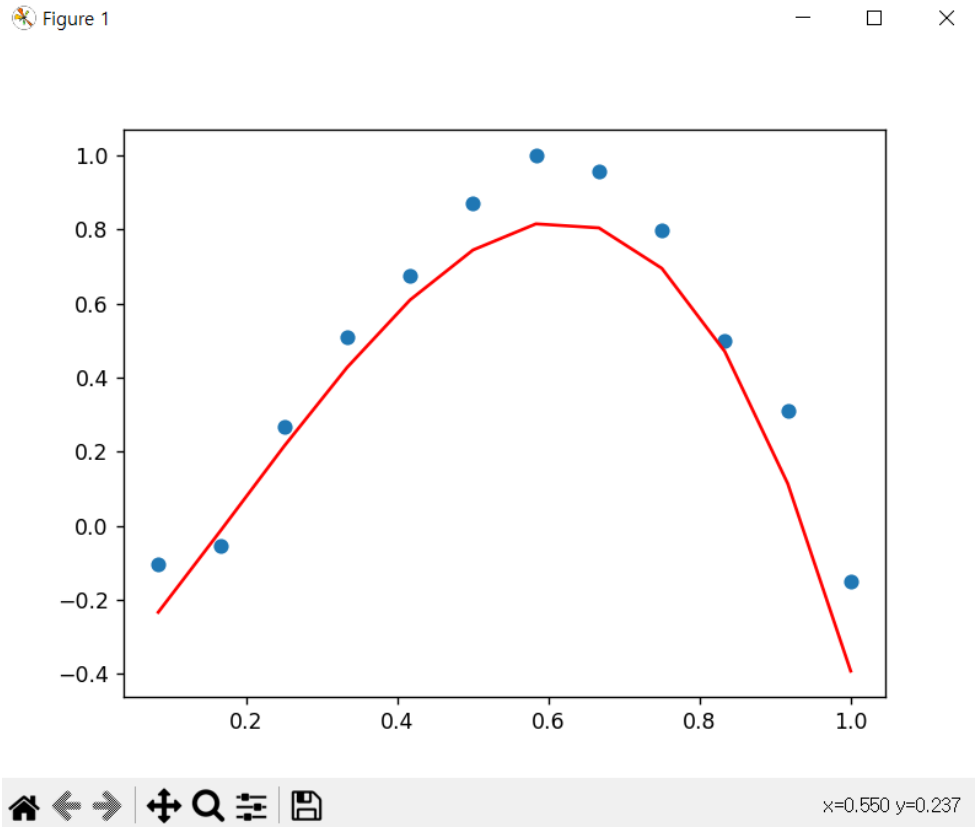


Figure 1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0.2
0.4
0.6
0.8
1.0
x=0.550 y=0.237

```
len(model.layers): 2
loss: 0.017826804891228676
weights: [[-0.43552595]
 [ 2.2299953 ]
 [ 2.7074482 ]
 [-4.8944826 ]]
```

➔ 3 차 다항식 회귀를 실행한 손실율 그래프와 예측 그래프, 그리고 모델의 계층 제일 작은 손실율 , 가중치 출력

4. 결론(느낀점)

- 이번 과제를 하면서 신경망 모델을 만들고 해당 모델을 학습하는 과정이 어떻게 진행되는지 조금 더 이해할 수 있는 계기가 되었습니다.

5. 부록(코드)

```
6.  import tensorflow as tf
7.  import numpy as np
8.  import matplotlib.pyplot as plt
9.  import pandas as pd
10. import numpy as np
11.
12. # 데이터에 csv 파일 데이터를 불러온다.encoding 을 통해 한글을 불러올수있게한
    다.칼럼의 헤더는 7 번라인
13. data = pd.read_csv(r 'C:\Users\dltjd\Artificial
    Intelligence\month_temp.csv', encoding = 'cp949', header = 6)
14.
```

```
15. # csv 파일의 년월 중 년도는 필요하지 않고 월만 필요하기 때문에 월의 값만 가지
    고온다.
16. x_month = data['년월'].str.split("-").str[1].astype(float).values
17.
18. # csv 파일에서 월별 평균기온을 가져온다.npArray 로 가져옴.
19. y_temp = data['평균기온(℃)'].values
20.
21. # 정규화
22. x_month /= max(x_month)
23. y_temp /= max(y_temp)
24.
25. # n - 차 다항식 회귀
26. n = 3
27.
28. # x 의 데이터갯수(20 개) x n + 1 개(4 개) 의 배열을 만들고 값을 1 로 초기화
29. X = np.ones(shape = (len(x_month), n + 1), dtype = np.float32)
30.
31. #0 번째 열을 1 로 초기화하고 배열 X 에서 1 번째 열부터 n 번째 열까지 x^i 값을
    차례로 넣어준다.
32. for i in range(1, n+1):
33.      X[:, i] = x_month ** i
34.
35. #신경망 입력 정의 n + 1 은 입력 데이터의 형태를 나타냄.
36. inputs = tf.keras.layers.Input(shape = (n + 1, ))
37. # 신경망의 출력을 정의 units = 1 은 뉴런 1 개 bias 는 없음을 뜻함 bias 는 없
    어도 될수있다.
38. # 왜냐하면, 상수항이 X 배열의 1 열에 1 이 초기화 되어 있기때문이다.
39. #(inputs) 이 Dense 층의 입력은 앞에서 정의한 input 사용하겠다는
40. outputs = tf.keras.layers.Dense(units = 1, use_bias = False)(inputs)
41. # 실제 신경망을 만든다.
42. model = tf.keras.Model(inputs = inputs, outputs = outputs)
43. # 신경망을 출력
44. model.summary()
45.
46.
47. #rmsprop 은 옵티마이저의 알고리즘의 하나 학습률은 0.1 을 opt 에 저장
48. opt = tf.keras.optimizers.RMSprop(learning_rate = 0.1)
```

```
#옵티마이저의 알고리즘하나인 rmsprop 을 저장한 opt 를 옵티마이저를 사용한다.
#그리고 mse 는 sum((트루값 - 예측값) ^ 2) / t.size()
model.compile(optimizer = opt, loss = 'mse')

#모델을 학습하는 함수 fit() X 는 입력데이터, y_month 는 정답, 10000 번 반복,
    #batch_size 를 통해 x 의 값을 쪼갠다 10000 번의 포문안에 6 번의 포문이 계속해서 돈다.
#verbose = 2 한줄씩 출력 0 은 결과만 출력
ret = model.fit(X, y_temp, epochs = 4000, batch_size = 2, verbose = 2)



#1: 모델 전체 저장
import os
if not os.path.exists("./RES1000"):
     os.mkdir("./RES1000")
model.save("./RES1000/1401.keras")   # HDF5, keras format

#2: 모델 구조 저장
json_string = model.to_json()
import json
file = open("./RES1000/1401.model", 'w')
json.dump(json_string, file)
file.close()

#3: 가중치 저장
model.save_weights("./RES1000/weights/1401")

# 4: 학습중에 체크포인트 저장
filepath = "RES1000/ckpt/1401-{epoch:04d}.ckpt"
cp_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath, verbose = 0, save_weights_only = True, save_freq = 50)
ret = model.fit(X, y_temp, epochs = 100, callbacks = [cp_callback],
verbose = 0)


```

```
84.
85.
86.
87. print("len(model.layers):", len(model.layers)) # 2
88.
89. loss = ret.history['loss']
90. print("loss:", loss[-1])
91. #print(model.get_weights()) # weights
92. print("weights:", model.layers[1].weights[0].numpy())
93.
94. plt.plot(loss)
95. plt.xlabel('epochs')
96. plt.ylabel('loss')
97. plt.show()
98.
99. plt.scatter(x_month, y_temp)
100.     y_pred = model.predict(X)
101.     plt.plot(x_month, y_pred, color = 'red')
102.     plt.show()
```