휴스타 ICT 설계 팀프로젝트1
MLP 실습 및 DNN

이새봄

생능출판, 천인국 저「딥러닝 EXPRESS」

2022.04.12

목차

- MLP 실습
- DNN 실습

KNU

MLP실습

numpy로 구현

```python
import numpy as np

# 시그모이드 함수
def actf(x):
    return 1/(1+np.exp(-x))

# 시그모이드 함수의 미분치
def actf_deriv(x):
    return x*(1-x)

# 입력유닛의 개수, 은닉유닛의 개수, 출력유닛의 개수
inputs, hiddens, outputs = 2, 2, 1
learning_rate=0.2

# 훈련 샘플과 정답
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
T = np.array([[1], [0], [0], [1]])

W1 = np.array([[0.10,0.20], [0.30,0.40]])
W2 = np.array([[0.50],[0.60]])
B1 = np.array([0.1, 0.2])
B2 = np.array([0.3])
```

```python
# 순방향 전파 계산
def predict(x):
        layer0 = x          # 입력을 layer0에 대입한다.
        Z1 = np.dot(layer0, W1)+B1  # 행렬의 곱을 계산한다.
        layer1 = actf(Z1)       # 활성화 함수를 적용한다.
        Z2 = np.dot(layer1, W2)+B2  # 행렬의 곱을 계산한다.
        layer2 = actf(Z2)       # 활성화 함수를 적용한다.
        return layer0, layer1, layer2
```

```python
# 역방향 전파 계산
def fit():
    global W1, W2, B1, B2       # 우리는 외부에 정의된 변수를 변경해야 한다.
    for i in range(90000):      # 9만번 반복한다.
        for x, y in zip(X, T):      # 학습 샘플을 하나씩 꺼낸다.
            x = np.reshape(x, (1, -1))  # 2차원 행렬로 만든다. ①
            y = np.reshape(y, (1, -1))  # 2차원 행렬로 만든다.

            layer0, layer1, layer2 = predict(x)         # 순방향 계산
            layer2_error = layer2-y             # 오차 계산
            layer2_delta = layer2_error*actf_deriv(layer2)  # 출력층의 델타 계산
            layer1_error = np.dot(layer2_delta, W2.T)       # 은닉층의 오차 계산 ②
            layer1_delta = layer1_error*actf_deriv(layer1)  # 은닉층의 델타 계산 ③

            W2 += -learning_rate*np.dot(layer1.T, layer2_delta) # ④
            W1 += -learning_rate*np.dot(layer0.T, layer1_delta) #
            B2 += -learning_rate*np.sum(layer2_delta, axis=0)   # ⑤
            B1 += -learning_rate*np.sum(layer1_delta, axis=0)   #
```

```python
def test():
    for x, y in zip(X, T):
        x = np.reshape(x, (1, -1))  # 하나의 샘플을 꺼내서 2차원 행렬로 만든다.
        layer0, layer1, layer2 = predict(x)
        print(x, y, layer2)     # 출력층의 값을 출력해본다.
fit()
test()
```

```
[[0 0]] [1] [[0.99196032]]
[[0 1]] [0] [[0.00835708]]
[[1 0]] [0] [[0.00836107]]
[[1 1]] [1] [[0.98974873]]
```

MLP실습

sklearn로 구현

```
X = [[0,0],[0,1],[1,0],[1,1]]
y = [0,0,0,1]

p = Perceptron()
p.fit(X,y)

print(p.predict(X)) # [0 0 0 1]
```

```
[0 0 0 1]
```

AND GATE
선형 모델 분류
가능

```
X = [[0,0],[0,1],[1,0],[1,1]]
y = [0,1,1,0]

from sklearn.linear_model import Perceptron

p = Perceptron()
p.fit(X,y)

print(p.predict(X)) # [0 0 0 0]
```

```
[0 0 0 0]
```

XOR GATE
선형 모델 분류
불가능

```
from sklearn import svm

p = svm.SVC()
p.fit(X,y)

print(p.predict(X)) # [0 1 1 0]
```

```
[0 1 1 0]
```

비선형 모델
사용하여 분류

MLP실습

keras로 구현

```python
import numpy as np
import tensorflow as tf

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(units=2, input_shape=(2,), activation='sigmoid')) #①
model.add(tf.keras.layers.Dense(units=1,  activation='sigmoid')) #②
model.compile(loss='mean_squared_error', optimizer=tf.keras.optimizers.SGD(lr=0.3))

model.summary()

X = np.array([[0, 0],[0, 1],[1, 0],[1, 1]])
y = np.array([[0], [1], [1], [0]])

model.fit(X, y, batch_size=1, epochs=10000)

print( model.predict(X) )
```

```
Epoch 9993/10000
4/4 [==============================] - 0s 2ms/step - loss: 2.3207e-04
Epoch 9994/10000
4/4 [==============================] - 0s 1ms/step - loss: 2.3205e-04
Epoch 9995/10000
4/4 [==============================] - 0s 2ms/step - loss: 2.3202e-04
Epoch 9996/10000
4/4 [==============================] - 0s 2ms/step - loss: 2.3200e-04
Epoch 9997/10000
4/4 [==============================] - 0s 2ms/step - loss: 2.3197e-04
Epoch 9998/10000
4/4 [==============================] - 0s 1ms/step - loss: 2.3194e-04
Epoch 9999/10000
4/4 [==============================] - 0s 2ms/step - loss: 2.3192e-04
Epoch 10000/10000
4/4 [==============================] - 0s 1ms/step - loss: 2.3189e-04
[[0.01678485]
 [0.9854888 ]
 [0.9855789 ]
 [0.01506674]]
```

Numpy에서의 복잡한 코드를 간단히 작성 가능

DNN실습

데이터셋

Personal Key Indicators of Heart Disease

2020 annual CDC survey data of 400k adults related to their health status

Data | Code (51) | Discussion (5) | Metadata

About Dataset

Key Indicators of Heart Disease

2020 annual CDC survey data of 400k adults related to their health status

What topic does the dataset cover?

Usability ⓘ
10.00

License
CC0: Public Domain

Expected update frequency
Annually

1 사망
0 생존

	C	D	E	F	G	H	I	J	K	L	M
1	creatinine_phosphokinase	diabetes	ejection_fr	high_blood	platelets	serum_crea	serum_sodi	sex	smoking	time	DEATH_EVENT
2	582	0	20	1	265000	1.9	130	1	0	4	1
3	7861	0	38	0	263358.03	1.1	136	1	0	6	1
4	146	0	20	0	162000	1.3	129	1	1	7	1
5	111	0	20	0	210000	1.9	137	1	0	7	1
6	160	1	20	0	327000	2.7	116	0	0	8	1
7	47	0	40	1	204000	2.1	132	1	1	8	1
8	246	0	15	0	127000	1.2	137	1	0	10	1
9	315	1	60	0	454000	1.1	131	1	1	10	1
10	157	0	65	0	263358.03	1.5	138	0	0	10	1

knu

DNN실습

코드

```
import numpy as np
import pandas as pd
import tensorflow as tf

train = pd.read_csv("heart_failure_clinical_records_dataset.csv", sep=',')

# 결손치가 있는 데이터 행은 삭제
train.dropna(inplace=True)

# 2차원 배열을 1차원 배열로 평탄화
target = np.ravel(train.DEATH_EVENT)

# 사망여부를 학습 데이터에서 삭제
train.drop(['DEATH_EVENT'], inplace=True, axis=1)

# 케라스 모델을 생성
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(16, activation='relu', input_shape=(train.shape[1],)))
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

```
# 케라스 모델을 컴파일한다.
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# 케라스 모델을 학습시킨다.
model.fit(train, target, epochs=10, batch_size=1, verbose=1)
```

```
Epoch 7/10
299/299 [==============================] - 1s 2ms/step - loss: 0.6377 - accuracy: 0.6789
Epoch 8/10
299/299 [==============================] - 1s 2ms/step - loss: 0.6352 - accuracy: 0.6789
Epoch 9/10
299/299 [==============================] - 1s 3ms/step - loss: 0.6334 - accuracy: 0.6789
Epoch 10/10
299/299 [==============================] - 1s 2ms/step - loss: 0.6321 - accuracy: 0.6789
```

DNN실습

HIDDEN LAYER

1

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(16, activation='relu', inpu
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

loss: 110.0692
accuracy: 0.6054

2

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(16, activation='relu', inpu
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

loss: 0.6321
accuracy: 0.6789

4

```
# 케라스 모델을 생성
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(16, activation='relu', input
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

loss: 0.6783
accuracy: 0.6789

6

```
# 케라스 모델을 생성
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(16, activation='relu', inpu
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

loss: 1.2933
accuracy: 0.5886

일반적으로 좋아지지만, 과하면 오히려 성능이 떨어짐

DNN실습

NODE 개수

2

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(2, activation='relu', inpu
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

loss: 0.7055
accuracy: 0.3211

4

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(4, activation='relu', input
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

loss: 7.9552
accuracy: 0.6020

8

```
# 케라스 모델을 생성
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(8, activation='relu',
model.add(tf.keras.layers.Dense(1, activation='sigmoi
```

loss: 119.0219
accuracy: 0.6187

16

```
# 케라스 모델을 생성
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(16, activation='relu', in
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

loss: 228.3505
accuracy: 0.5819

일반적으로 좋아지지만, 과하면 오히려 성능이 떨어짐

knu

DNN실습

ACTIVATION FUNCTION

relu

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(16, activation='relu', inpu
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

loss: 110.0692
accuracy: 0.6054

sigmoid

```
# 케라스 모델을 생성
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(16, activation='sigmoid', i
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

loss: 0.6289
accuracy: 0.6789

본 데이터셋에 대해서는, sigmoid가 더 적합함 → 하이퍼 파라미터 튜닝 필요

참고자료

- https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease?datasetId=1936563&sortBy=voteCount

- 딥 러닝 강의자료 코드