

인공지능 포트폴리오

20192623 이준모

목차

- 1. 인공지능과 딥러닝
 - 인공지능의 기본 정보
- 2. 텐서플로
 - 텐서플로의 기본정보
- 3. 텐서플로 난수
- 4.MNIST

인공지능의 시작

앨런튜링은 생각하는 기계의 구현 가능성에 관한 논문을 발표하였고 대화를 주고 받을 때 대화 상대가 기계인지 사람인지 구별하는 튜링 테스트를 고안하여 튜링 테스트를 통과하면 기계가 생각할 수 있다고 말할 근거가 된다고 하였다. 인공지능이 처음 사용된 것은 1956년 다트머스대 학술회에서 최초의 AI 프로그램인 논리 연산기를 발표하였다.

AI의 암흑기와 전성기

- AI는 1940년대부터 시작된 분야이다
- AI의 첫번째 암흑기는 1969-1980년에 마빈 민스키의 인공 신경망인 퍼셉트론에 대한 비판으로 촉발되었다.
- AI의 두번째 암흑기는 1987-1993년에 규칙 기반의 전문가시스템의 의구심과 한계에 의해 촉발되었다.
- 2010년대에 들어서는 이러한 문제점들이 해결되어 AI의 전성기가 시작되었다.

인공 지능

컴퓨터가 인간처럼 지적 능력을 갖게 하거나 행동하도록 하는 모든 기술

머신 러닝

머신러닝은 기계가 스스로 학습할 수 있도록 하는 인공지능의 한 연구 분야

스스로 기술을 학습하여 터득하므로 명시적으로 프로그래밍을 해주지 않아도 컴퓨터가 학습을 할 수 있도록 해주는 인공지능의 한 형태

데이터가 많을수록 학습량이 많아지므로 시간이 지날수록 작업 수행 능력이 더욱 상승한다

머신러닝의 종류

■ 지도 학습

- 한 쌍의 문제와 답으로 이루어진 학습데이터를 기반으로 학습시키는 방법

■ 비지도 학습

- 정답이 없는 한 데이터를 주어 이 안에 있는 관계를 찾아내는 방법
- 군집, 시각화와 차원 축소, 연관규칙 학습 등

■ 강화 학습

- 행동에 보상과 벌을 주는 방식으로 학습시키는 방법

인공신경망(퍼셉트론)

- 1957년 코넬대 교수, 심리학자인 프랭크 로젠블랫이 최초의 인공신경망을 제안하였다.
- 원하는 결과를 얻기 위해 신경망에서 많은 데이터를 정확하게 구분하도록 학습시킨다.
- 인공신경망은 인간의 신경세포인 뉴런을 모방하여 만든 가상의 신경이다.

인공신경망 구조

- 입력층과 출력층, 중간의 은닉층으로 이루어져있다.
- 중간의 은닉층이 많아질수록 학습 성능이 증가한다.
- 다중 계층인 심층 신경망을 사용하며 여러 단계의 심층 신경망을 거쳐 사고하며 결론을 도출한다.

텐서플로

딥러닝 라이브러리로 텐서플로, 케라스, 파이토치 등이 있으며 주로 사용하는 언어는 파이썬이다. 그 중 텐서플로는 구글에서 개발한 오픈소스 라이브러리로 딥러닝 개발에 최적화되어있다.

텐서란?

- 텐서는 모든 데이터를 의미한다
 - 차원이 없는 텐서 - 스칼라
10
 - 1차원 텐서 - 벡터
[10, 20]
 - n차원 텐서 - n차원 행렬
[[10, 20], [10, 20]] (2차원 텐서)

텐서플로 흐름

- 그래프라는 객체에 저장되어 실행되고 그래프 계산은 외부컴퓨터에 정보를 주어 값을 받아오는 형태로 진행된다.
- 이 과정은 session이 담당하며 생성, 사용, 종료 과정이 필요하다.
- 텐서형태의 데이터들이 딥러닝 모델을 따라 연산이 일어난다.

텐서플로 불러오기

```
[ ] try:
    %tensorflow_version 1.x
except Exception:
    pass

import tensorflow as tf
tf.__version__
```

▶ '1.15.2'

%tensorflow_version 1.x로 1.x버전 텐서플로를 불러오기 를 하였고 tf.__version__으로 현재 텐서플로의 버전을 확인하였다.

ctrl + m을 눌러 런타임을 다시 시작할 수 있고 다시 시작 하면서 텐서플로의 새로운 버전을 불러올 수 있다.

import tensorflow로 텐서플로를 불러온다면 최신버전인 2.3.0이 자동으로 불러와진다.

텐서플로 기본적인 함수들

```
[ ] c = tf.constant('Hello World!')  
    print(c)  
    print(c.numpy())
```

```
tf.Tensor(b'Hello World!', shape=(), dtype=string)  
b'Hello World!'
```

- 텐서플로 결과값에서 값만을 보고 싶다면 numpy를 사용하면 된다.

```
[ ] x = tf.constant(1.)  
    bool = tf.constant(True)  
    res = tf.cond(bool, lambda: tf.add(x, 1. ), lambda: tf.add(x, 10. ))  
  
    print(res)  
    print(res.numpy())
```

```
tf.Tensor(2.0, shape=(), dtype=float32)  
2.0
```

- cond 함수는 첫 번째 값을 검사하여 참이면 앞의 값을 반환하고 거짓이면 뒤의 값을 반환한다.

브로드캐스팅

```
[ ] x = tf.constant([[0], [10], [20], [30]])  
    y = tf.constant([0, 1, 2])
```

```
print((x+y).numpy())
```

```
[[ 0  1  2]  
 [10 11 12]  
 [20 21 22]  
 [30 31 32]]
```

- 행렬의 크기가 달라도 계산이 될 수 있도록 확장해주는 것이 브로드 캐스팅이다.

- $x = \begin{bmatrix} 0 \\ 10 \\ 20 \\ 30 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 0 & 0 \\ 10 & 10 & 10 \\ 20 & 20 & 20 \\ 30 & 30 & 30 \end{bmatrix}$

- $y = \begin{bmatrix} 0 & 1 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix}$

arrange 와 ones

```
[ ] import numpy as np
    print(np.arange(3))
    print(np.ones((3, 3)))
    print()

    x = tf.constant ((np.arange(3)))
    y = tf.constant([5], dtype=tf.int64)
    print(x)
    print(y)
    print(x+y)

[0 1 2]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]


tf.Tensor([0 1 2], shape=(3,), dtype=int64)
tf.Tensor([5], shape=(1,), dtype=int64)
tf.Tensor([5 6 7], shape=(3,), dtype=int64)
```

- arrange는 주어진 값까지 자동으로 행렬을 생성해주는 함수이다.
- ones는 주어진 형태로 행렬을 생성하며 값으로는 전부 1이 들어가는 함수이다.


행렬의 곱셈

- 행렬의 곱셈은 a와 b를 곱한다고 하면 a의 1행과 b의 1열이 곱해진 값이 결과값의 1행 1열이 된다.

a11	a12
a21	a22



b11	b12
b21	b22



a11b11+a12b21	a11b12+a12b22
a21b11+a22b21	a21b12+a22b22

- 이 때 사용하는 함수는 텐서플로의 matmul 함수이다.

행렬과 관련된 함수들

```
[ ] my_image = tf.zeros([2, 5, 5, 3])  
my_image.shape
```

```
TensorShape([2, 5, 5, 3])
```

```
[ ] tf.rank(my_image)
```

```
<tf.Tensor: shape=(), dtype=int32, numpy=4>
```

```
[ ] rank_three_tensor = tf.ones([3, 4, 5])  
rank_three_tensor.shape
```

```
TensorShape([3, 4, 5])
```

```
[ ] matrix = tf.reshape(rank_three_tensor, [6, 10])  
matrix
```

```
<tf.Tensor: shape=(6, 10), dtype=float32, numpy=  
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],  
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],  
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],  
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],  
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],  
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]], dtype=float32)>
```

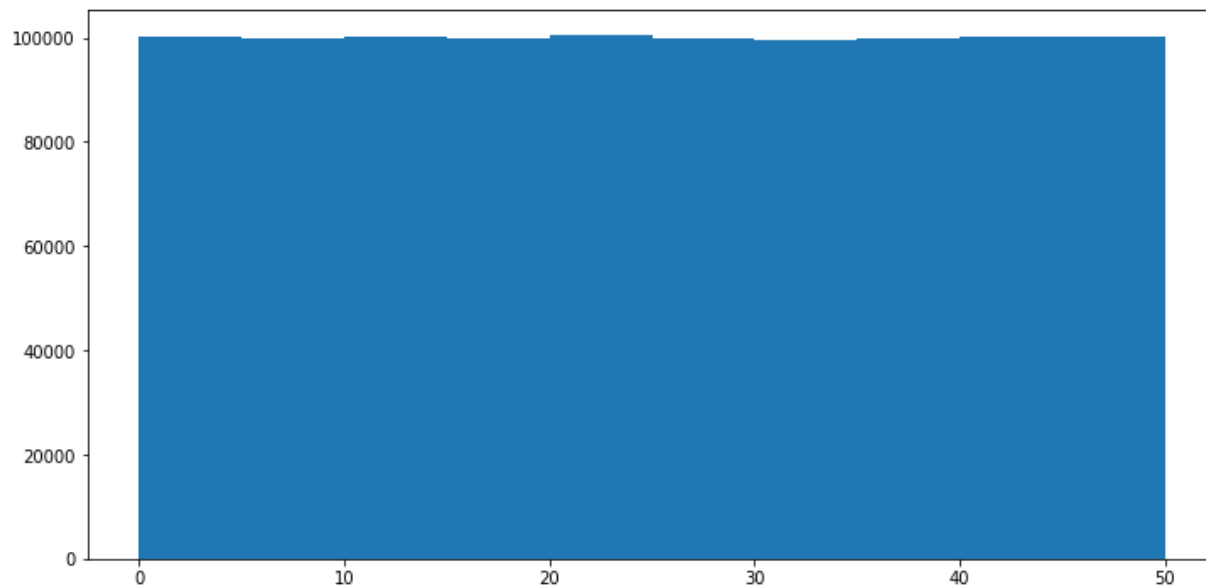
- rank 함수는 행렬의 랭크값을 반환해 준다.
- shape 함수는 함수의 형태를 반환해 준다.
- reshape 함수는 주어진 형태로 함수의 형태를 변경해주는 함수이다.
 - 만약 주어진 형태보다 원소의 개수가 적다면 오류가 발생한다.

텐서플로
난수

균등분포

```
[ ] import matplotlib.pyplot as plt
    rand = tf.random.uniform([1000000], 0, 50)
    plt.hist(rand, bins=10)
```

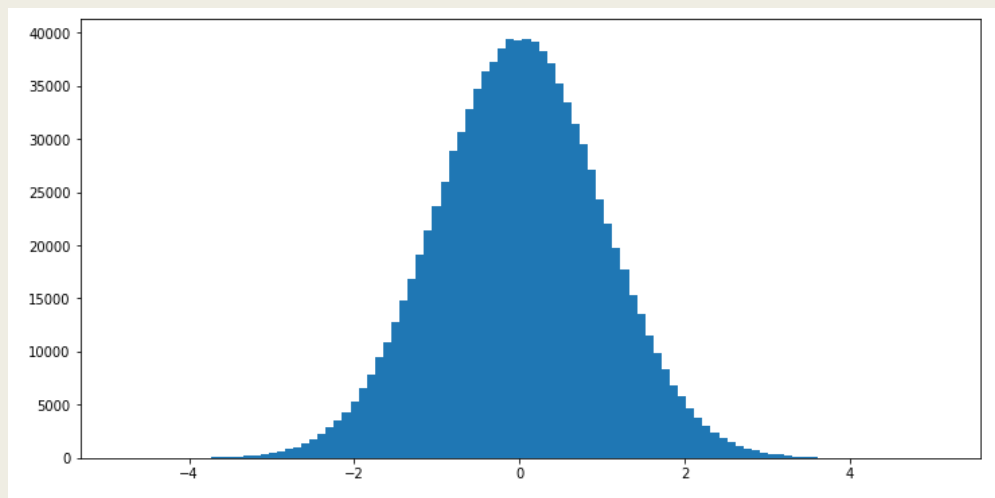
```
(array([100038., 99850., 100124., 99740., 100456., 99717., 99607.,
        99952., 100199., 100317.]),
 array([1.1920929e-05, 5.0000033e+00, 9.9999952e+00, 1.4999987e+01,
        1.9999979e+01, 2.4999969e+01, 2.9999962e+01, 3.4999954e+01,
        3.9999943e+01, 4.4999935e+01, 4.9999928e+01], dtype=float32),
 <a list of 10 Patch objects>)
```



- random은 난수 함수이며 uniform은 균등 분포를 생성해주는 함수이다.
- uniform의 첫 번째 변수는 값의 개수고 두 번째 변수는 값의 최솟값 세 번째 변수는 값의 최댓값이다.
- hist 함수는 분포를 시각적으로 그려주며 bins는 분포의 막대 개수를 의미한다.

정규분포

```
[ ] import matplotlib.pyplot as plt  
    rand = tf.random.normal([1000000], 0, 1)  
    plt.hist(rand, bins=100)
```



- normal은 정규 분포를 생성해주는 함수이다.
- normal의 변수들은 uniform과 같이 첫 번째 변수는 값의 개수고 두 번째 변수는 값의 최솟값 세 번째 변수는 값의 최댓값이다.

MNIST

- MNIST는 딥러닝 손글씨 인식에 사용되는 데이터셋이다.

MNIST란

- 미국 국립 표준 기술원이 만든 손글씨 숫자 데이터의 집합이다.
- 훈련용 데이터와 테스트용 데이터가 존재한다.
- 손글씨 패턴은 0~9까지 존재한다.

```
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 0. 30. 36. 94. 154. 170.
253. 253. 253. 253. 253. 225. 172. 253. 242. 195. 64. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 49. 238. 253. 253. 253. 253.
253. 253. 253. 253. 251. 93. 82. 82. 56. 39. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 18. 219. 253. 253. 253. 253.
253. 198. 182. 247. 241. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 0. 80. 156. 107. 253. 253.
205. 11. 0. 43. 154. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 14. 1. 154. 253.
90. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 139. 253.
190. 2. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 11. 190.
253. 70. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 35.
241. 225. 160. 108. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
81. 240. 253. 253. 119. 25. 0. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 45. 186. 253. 253. 150. 27. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 16. 93. 252. 253. 187. 0. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 249. 253. 249. 64. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 46. 130. 183. 253. 253. 207. 2. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 39.
148. 229. 253. 253. 253. 250. 182. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 24. 114. 221.
253. 253. 253. 253. 201. 78. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 0. 23. 66. 213. 253. 253.
253. 253. 198. 81. 2. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 18. 171. 219. 253. 253. 253. 253.
195. 80. 9. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 55. 172. 226. 253. 253. 253. 253. 244. 133.
11. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 136. 253. 253. 253. 212. 135. 132. 16. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0.
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

- 손글씨 구조
 - 28 * 28 의 정사각형 형태의 구조
 - 내부는 0~255값으로 0은 값이 없는 거고 255로 갈 수록 글씨가 진해진다.

딥러닝 과정



MNIST 불러오기

```
[ ] import tensorflow as tf

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
[ ] x_train.shape

(60000, 28, 28)
```

```
[ ] y_train.shape

(60000,)
```

- `tf.keras.datasets.mnist`로 mnist 데이터를 불러올 수 있다.
- mnist의 훈련용 데이터의 문제는 60000개가 들어있고 28*28형태이며 답은 문제와 일치하는 60000개가 들어있다.

모델 만들기

- `tf.keras.models.Sequential`에 층을 쌓아 신경망을 구성
- `Flatten`은 값을 평탄화 해주는 층이다.
- `Dense`는 완전연결 층이며 `activation`에 어떤 활성화 함수를 사용할 것인지 정한다.
- `Dropout`은 손실함수로 훈련에 효과를 높여준다.

```
[ ] model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

```
[ ] model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 784)	0
dense_4 (Dense)	(None, 128)	100480
dropout_2 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 10)	1290

Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0

모델 학습

```
[ ] model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5  
1875/1875 [=====] - 4s 2ms/step - loss: 0.2985 - accuracy: 0.9130  
Epoch 2/5  
1875/1875 [=====] - 4s 2ms/step - loss: 0.1409 - accuracy: 0.9580  
Epoch 3/5  
1875/1875 [=====] - 4s 2ms/step - loss: 0.1050 - accuracy: 0.9676  
Epoch 4/5  
1875/1875 [=====] - 4s 2ms/step - loss: 0.0873 - accuracy: 0.9738  
Epoch 5/5  
1875/1875 [=====] - 3s 2ms/step - loss: 0.0724 - accuracy: 0.9772  
<tensorflow.python.keras.callbacks.History at 0x7f9c056a75c0>
```

- 주어진 학습용 데이터로 fit 함수로 학습을 시킴
- epochs는 학습의 횟수이며 많이 할 수록 모델의 문제 해결 능력이 좋아짐
- 5번 째 학습 값을 보면 앞의 학습보다 결과값이 좋은 것을 확인할 수 있음

모델 평가

```
[ ] model.evaluate(x_test, y_test)
```

```
313/313 [=====] - 0s 1ms/step - loss: 0.0766 - accuracy: 0.9770  
[0.07659716159105301, 0.9769999980926514]
```

- evaluate 함수로 mnist에 있는 테스트 데이터로 테스트 진행
- 약 97프로의 정확도로 손글씨를 맞춤
- 만약 학습을 더 시켰다면 더 좋은 값이 나왔을 것이다.



감사합니다

