Keras로 딥러닝 시작하기

학습 내용

- 라이브러리 불러오기
- 신경망을 위한 데이터 이해
- MNIST 데이터 셋 가져오기
- 넘파이를 활용한 텐서 조작
- 활성화 함수 알아보기

목차

01. 라이브러리 임포트

목차로 이동하기

- 설치가 안되어 있을 경우, 설치하기
  - pip install keras

In [1]:

```
import keras
import tensorflow as tf
```

In [2]:

```
print(tf.__version__)
print(keras.__version__)
```

```
2.10.0
2.10.0
```

02. 신경망을 위한 데이터 이해

목차로 이동하기

- 배열에 있는 특정 원소를 일부 선택하는 것을 슬라이싱(slicing)이라 한다.

- Tensor 자료형
  - 데이터를 위한 컨테이너(container). 거의 대부분 수치형 데이터를 다루기 위한, 숫자를 위한 컨테이너.
  - 텐서는 임의의 차원 개수를 가지는 행렬의 일반화된 모습
- 스칼라 : 하나의 숫자만 담고 있는 텐서를 스칼라라고 한다.
  - 0차원 텐서, 0D텐서

In [3]:

```
import numpy as np
x = np.array(12)
print("x의 차원 : ",x.ndim)      # 차원 확인
print("x의 shape : ",x.shape)
print("x의 값 : ", x)
```

```
x의 차원 :  0
x의 shape :  ()
x의 값 :  12
```

벡터(1D 텐서)

- 숫자의 배열을 벡터(vector)또는 1D 텐서라고 부른다. 1D 텐서는 딱 하나의 축을 가진다.

In [4]:

```
x = np.array([10,20,30,40,50])

print("x의 차원 : ",x.ndim)      # 차원 확인
print("x의 shape : ",x.shape)
print("x의 값 : ", x)
```

```
x의 차원 :  1
x의 shape :  (5,)
x의 값 :  [10 20 30 40 50]
```

- 위의 값은 5개의 원소를 가지고 있으므로 5차원 벡터라 부른다.
- 5D 벡터는 하나의 축을 따라 5개의 차원을 가지고, 5D텐서는 5개의 축을 가진것.

행렬(2D 텐서)

- 벡터의 배열을 행렬(matrix) 또는 2D텐서라 부른다. 행렬에는 2개의 축이 있다. 보통 행과 열이라 한다.

In [5]:

```
x = np.array([ [11,21,31],
               [12,22,32],
               [13,23,33] ])

print("x의 차원 : ",x.ndim)      # 차원 확인
print("x의 shape : ",x.shape)
print("x의 값 : ", x)
```

```
x의 차원 :  2
x의 shape :  (3, 3)
x의 값 :  [[11 21 31]
 [12 22 32]
 [13 23 33]]
```

3D텐서와 고차원 텐서

- 행렬들을 하나의 새로운 배열로 합치면 숫자가 채워진 직육면체 형태로 해석할 수 있는 3D텐서가 만들어진다.

In [6]:

```
x = np.array([
              [ [11,21,31],
                [12,22,32],
                [13,23,33] ],
              [ [11,21,31],
                [12,22,32],
                [13,23,33] ],
              [ [11,21,31],
                [12,22,32],
                [13,23,33] ]
             ])
print(x.ndim)
print(x.shape)
x
```

```
3
(3, 3, 3)
```

Out[6]:

```
array([[[11, 21, 31],
        [12, 22, 32],
        [13, 23, 33]],

       [[11, 21, 31],
        [12, 22, 32],
        [13, 23, 33]],

       [[11, 21, 31],
        [12, 22, 32],
        [13, 23, 33]]])
```

- 이와같이 3D텐서들을 하나의 배열로 합치면 4D텐서가 된다
- 딥러닝에서는 보통 0D에서 4D까지의 텐서를 다룬다.
- 동영상 데이터를 다룬다면 5D텐서까지도 다루어 볼 수 있다.

03. MNIST 데이터 셋

목차로 이동하기

- 배열에 있는 특정 원소를 일부 선택하는 것을 슬라이싱(slicing)이라 한다.

In [7]:

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

실습

- (1) train_images의 차원, shape, 데이터 확인
- (2) train_labels의 차원, shape, 데이터 확인
- (3) 일부 영역만 가져와보기

In [8]:

```python
# 이미지 데이터의 차원, shape, 자료형 확인
print(train_images.ndim)
print(train_images.shape)
print(train_images.dtype)
print(train_images[0])
```

```
3
(60000, 28, 28)
uint8
[[  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   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   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   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   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   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  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  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   0 249
  253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  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  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  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   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   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   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]
```

In [9]:

```
# target 데이터의 차원, shape, 자료형 확인
print(train_labels.ndim)
print(train_labels.shape)
print(train_labels.dtype)
print(train_labels[0:10])
```

```
1
(60000,)
uint8
[5 0 4 1 9 2 1 3 1 4]
```

04. 이미지를 출력해 보기

목차로 이동하기

- 배열에 있는 특정 원소를 일부 선택하는 것을 슬라이싱(slicing)이라 한다.

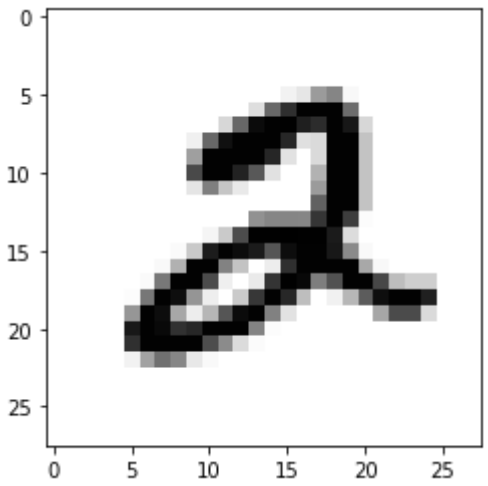
In [10]:

```
import matplotlib.pyplot as plt
```

In [11]:

```
image = train_images[5]
print(image.shape)
plt.imshow(image, cmap=plt.cm.binary)
plt.show()
```

```
(28, 28)
```



05. 넘파이를 활용한 텐서 조작

목차로 이동하기

- 배열에 있는 특정 원소를 일부 선택하는 것을 슬라이싱(slicing)이라 한다.

In [12]:

```
# 행, 열, 높이
# 행 10~50선택
my_slice = train_images[10:50]
print(my_slice.shape)
```

```
(40, 28, 28)
```

In [13]:

```
my_slice = train_images[10:50, :, :]  # 이전것과 동일
print(my_slice.shape)
```

```
(40, 28, 28)
```

In [14]:

```
my_slice = train_images[10:50, 0:28, 0:28]  # 이전것과 동일
print(my_slice.shape)
```

```
(40, 28, 28)
```

이미지의 오른쪽 아래 14 x 14 픽셀 선택

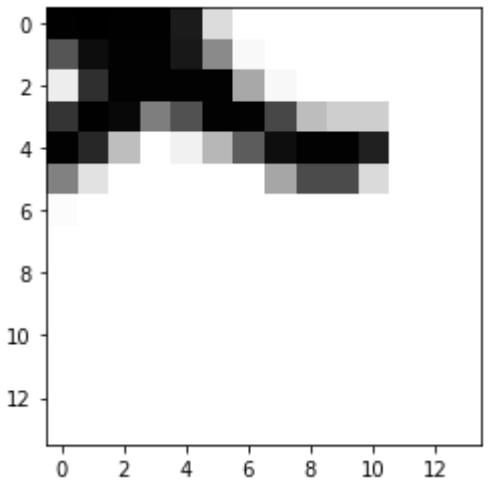
In [15]:

```
my_slice = train_images[:, 14:, 14:]
print(my_slice.shape)
```

```
(60000, 14, 14)
```

In [16]:

```
image = my_slice[5]
plt.imshow(image, cmap=plt.cm.binary)
plt.show()
```



Quiz

- 이미지를 행렬 5 ~ 23, 4 ~ 25까지 가져와 이에 대한 이미지를 출력해 보자.

06. 배치 데이터

목차로 이동하기

- 딥러닝 모델에서는 한 번에 전체 데이터를 처리하지 않는다.
- 그대신 데이터를 작은 배치(batch)로 나눈다.
- 구체적으로 말하면 MNIST 숫자 데이터에서 크기가 128인 배치 하나는 다음과 같다.

In [17]:

```
batch = train_images[ : 128]
```

In [18]:

```
# 다음 배치
batch = train_images[128:256]
```

In [19]:

```
# n번째 배치
# batch = train_images[128 * n:128 * (n+1)]
```

07. 텐서의 실제 사례

목차로 이동하기

- **벡터 데이터(sample, features)크기의 2D텐서**
- **시계열 데이터 또는 시퀀스 (sequence) 데이터 : (samples, timesteps, features)크기의 3D텐서**
- **이미지 : 경우(samples, height, width, channels) 또는 (samples, channels, height, width) 크기의 4D텐서**
- **동영상 : (samples, frames, height, width, channels) 또는 (sampels, frames, channels, height, width) 크기의 5D텐서**

이미지 데이터

- 이미지는 전형적으로 높이, 너비, 컬러 채널의 3차원으로 이루어진다.
- 흑백이미지의 channel의 차원 크기는 1입니다.
- 256 x 256 크기의 흑백 이미지에 대한 128개의 배치는 (128, 256, 256, 1)크기의 텐서
- 256 x 256 크기의 컬러 이미지에 대한 128개의 배치는 (128, 256, 256, 3)크기의 텐서

비디오 데이터

- 프레임의 연속 (frames, height, width, color_depth)의 4D텐서

- 여러 비디오의 배치(samples, frames, height, width, color_depth)의 5D텐서로 저장.

60초 짜리 144 x 256유튜브 비디오 클립을 초당 4프레임으로 샘플링하면 240프레임이 된다.

- 클립을 4개 가진 배치는 (4, 240, 144, 256, 3) 크기의 텐서에 저장.

08. 텐서의 크기 변환

목차로 이동하기

In [20]:

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

In [21]:

```
train_images = train_images.reshape((60000, 28*28))
train_images.shape
```

Out[21]:

```
(60000, 784)
```

In [22]:

```
x = np.array( [[0. , 1.],
               [2. , 3.],
               [4. , 5.]])
print(x.shape)

x1 = x.reshape((6,1))
x1.shape
```

```
(3, 2)
```

Out[22]:

```
(6, 1)
```

09. 활성화 함수 살펴보기

목차로 이동하기

In [23]:

```
import mglearn
```

In [24]:

```
# graphviz의 설치가 필요 - 콜랩 확인 가능
# 퍼셉트론 설명 이미지 확인
# mglearn.plots.plot_logistic_regression_graph()
# mglearn.plots.plot_single_hidden_layer_graph()
```

활성화 함수

Relu(렐루-rectified linear unit, ReLU), tanh(하이퍼 볼릭 탄젠트-hyperbolic tangent)

- ReLU 함수는 0이하를 잘라버림.
- tanh 함수는 낮은 입력값에 대해 -1로 수렴, 큰 입력값에 대해 +1로 수렴
- sigmoid 함수는 낮은 입력값에 대해 0에 수렴, 큰 입력값에 대해 1로 수렴

In [25]:

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

In [26]:

```
line = np.linspace(-3, 3, 100)
tanh_line = np.tanh(line)
relu_line = np.maximum(line, 0)  # 두개의 배열값 중 최대값 찾기
sig_line = 1/(1+np.exp(-line))

step_line = line.copy()
step_line[step_line <= 0] = 0
step_line[step_line > 0] = 1
```

In [27]:

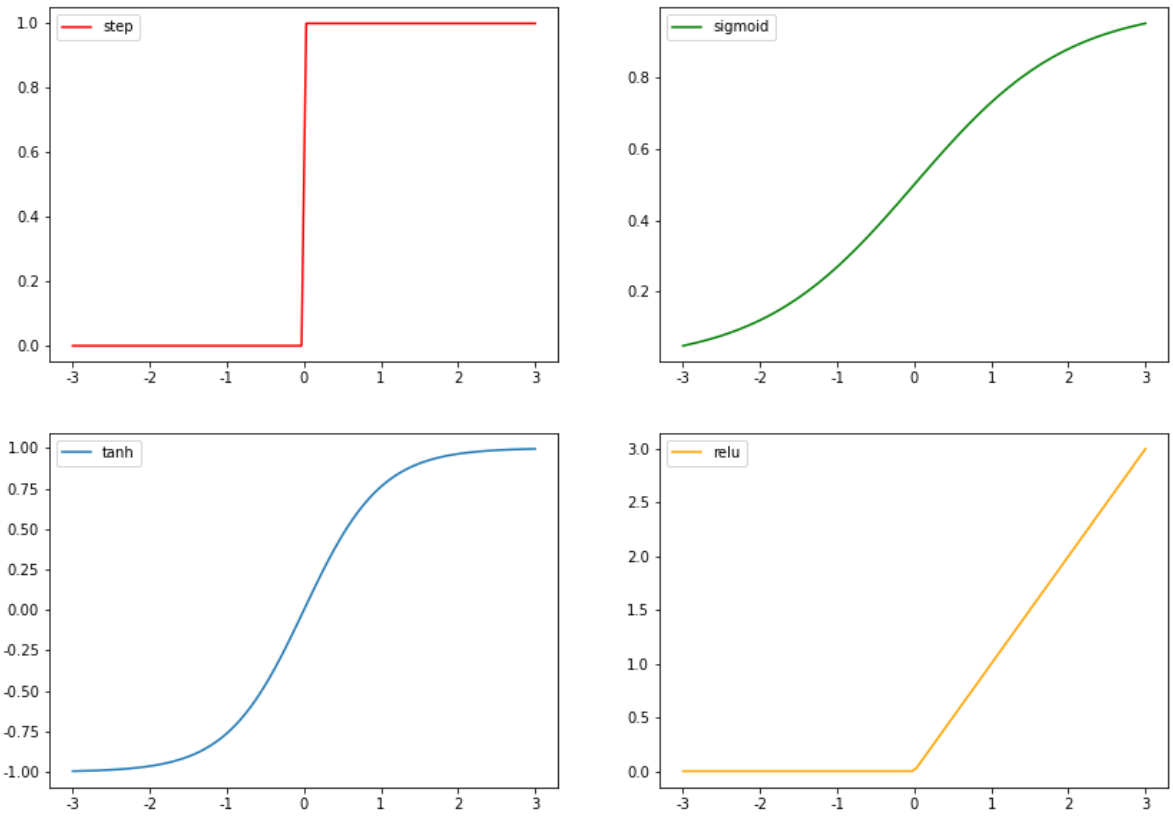
```
# 음수 표시
import matplotlib
matplotlib.rcParams['axes.unicode_minus'] = False
```

In [28]:

```
plt.rcParams["figure.figsize"] = (14,10)
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)
# step function
ax1.plot(line, step_line, label='step', color='red')
ax1.legend(loc='best')

# sigmoid function (시그모이드 함수)
ax2.plot(line, sig_line, label='sigmoid', color='green')
ax2.legend(loc='best')

# Hyperbolic Tangent(tanh)
ax3.plot(line, tanh_line, label='tanh')
ax3.legend(loc='best')

# relu
ax4.plot(line, np.maximum(line, 0), label='relu', color='orange')
ax4.legend(loc='best')
```

Out[28]:

<matplotlib.legend.Legend at 0x17c8a3f9820>



REF

- 케라스 창시자에게 배우는 딥러닝
- 파이썬 라이브러리를 활용한 데이터 분석