

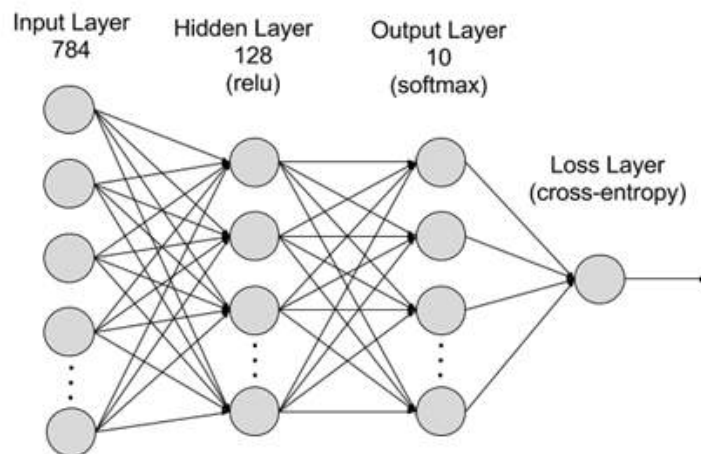
04 딥러닝 활용 맛보기

앞에서 우리는 중·고등학교 때 배운 함수들에 대해 인공 신경망을 학습시켜 근사 함수를 만들어 보았습니다. 실제로 인공 신경망 함수는 이러한 함수들로 표현하기 어려운 복잡한 형태의 입출력 데이터에 대한 근사 함수를 만들 때 사용합니다. 여기서는 인공 신경망을 학습시켜 숫자와 그림을 인식해 보도록 합니다. 일반적으로 인공 신경망 관련된 책에서 소개되는 예를 수행해 보며, 인공 신경망이 어떻게 활용되는 지 살펴봅니다.

01 딥러닝 활용 예제 살펴보기



여기서는 MNIST라고 하는 손글씨 숫자 데이터를 입력받아 학습을 수행하는 예제를 살펴봅니다. 여기서 소개하는 예제의 경우 독자 여러분은 구체적으로 어떤 데이터가 사용되는지 알기 어렵습니다. 데이터에 대해서는 다음 단원에서 자세히 살펴보도록 합니다. 여기서는 딥러닝 예제의 기본적인 구조를 살펴보도록 합니다. 다음과 같은 모양의 인공 신경망을 구성하고 학습시켜 봅니다.



1. 다음과 같이 예제를 작성합니다.

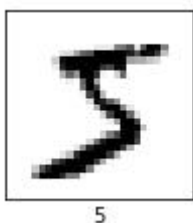
141_1.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (X, YT), (x, yt) = mnist.load_data() #60000개의 학습데이터, 10000개의 검증데이터
06
07 X, x = X/255, x/255 # 60000x28x28, 10000x28x28
08 X, x = X.reshape((60000,784)), x.reshape((10000,784))
09
10 model = tf.keras.Sequential([
11     tf.keras.Input(shape=(784,)),
12     tf.keras.layers.Dense(128, activation='relu'),
13     tf.keras.layers.Dense(10, activation='softmax')
14 ]) # 신경망 모양 결정(W, B 내부적 준비)
15
16 model.compile(optimizer='adam',
17               loss='sparse_categorical_crossentropy',
18               metrics=['accuracy'])
19
20 model.fit(X, YT, epochs=5)
21
22 model.evaluate(x, yt)
```

01 : import문을 이용하여 tensorflow 모듈을 tf라는 이름으로 불러옵니다. tensorflow 모듈은 구글에서 제공하는 인공 신경망 라이브러리입니다.

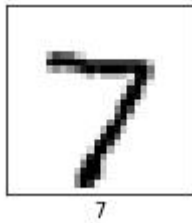
03 : mnist 변수를 생성한 후, tf.keras.datasets.mnist 모듈을 가리키게 합니다. mnist 모듈은 손글씨 숫자 데이터를 가진 모듈입니다. mnist 모듈에는 6만개의 학습용 손글씨 숫자 데이터와 1만개의 시험용 손글씨 숫자 데이터가 있습니다. 이 데이터들에 대해서는 다음 단원에서 자세히 살펴봅니다.

05 : mnist.load_data 함수를 호출하여 손글씨 숫자 데이터를 읽어와 x_train, y_train, x_test, y_test 변수가 가리키게 합니다. x_train, x_test 변수는 각각 6만개의 학습용 손글씨 숫자 데이터와 1만개의 시험용 손글씨 숫자 데이터를 가리킵니다. y_train, y_test 변수는 각각 6만개의 학습용 손글씨 숫자 라벨과 1만개의 시험용 손글씨 숫자 라벨을 가리킵니다. 예를 들어 x_train[0], y_train[0] 항목은 각각 다음과 같은 손글씨 숫자 5에 대한 그림과 라벨 5를 가리킵니다.



또, x_test[0], y_test[0] 항목은 각각 다음과 같은 손글씨 숫자 7에 대한 그림과 라벨 7을 가

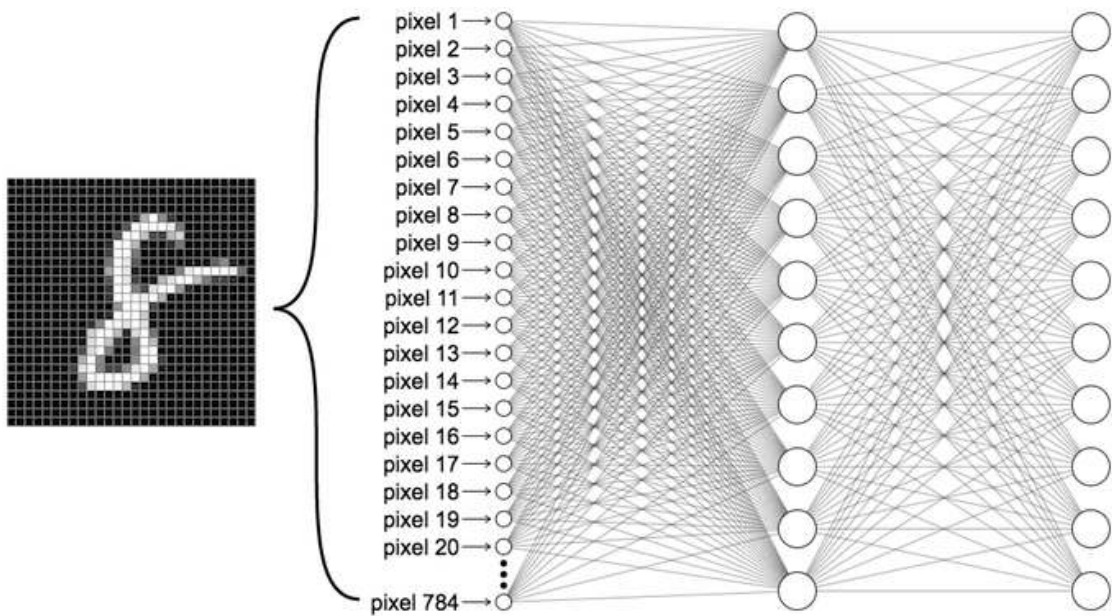
리킵니다.



06 : x_train, x_test 변수가 가리키는 6만개, 1만개의 그림은 각각 28x28 픽셀로 구성된 그림이며, 1픽셀의 크기는 8비트로 0에서 255사이의 숫자를 가집니다. 모든 픽셀의 숫자를 255.0으로 나누어 각 픽셀을 0.0에서 1.0사이의 실수로 바꾸어 인공 신경망에 입력하게 됩니다. 다음은 x_train[0] 그림의 픽셀값을 출력한 그림입니다.

[illegible]

07 : x_train, x_test 변수가 가리키는 6만개, 1만개의 그림은 각각 28x28 픽셀, 28x28 픽셀로 구성되어 있습니다. 이 예제에서 소개하는 인공 신경망의 경우 그림 데이터를 입력할 때 28x28 픽셀을 784(=28x28) 픽셀로 일렬로 세워서 입력하게 됩니다. 그래서 10줄에 있는 Input 클래스는 일렬로 세워진 784 픽셀을 입력으로 받도록 구성됩니다.



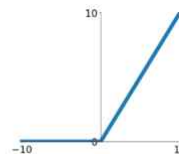
<출처 : <https://www.kdnuggets.com/2019/11/designing-neural-networks.html>>

09~13 : `tf.keras.Sequential` 클래스를 이용하여 인공 신경 망을 생성합니다. 여기서 생성한 인공 신경망은 $138(=128+10)$ 개의 인공 신경으로 구성됩니다. 입력층에 표시된 노드는 입력값의 개수를 표시하며 나머지 층에 있는 노드는 인공 신경을 나타냅니다. 인공 신경망의 내부 구조는 뒤에서 자세히 살펴봅니다. 생성된 인공 신경망은 일반적으로 모델이라고 합니다. 모델은 모형을 의미하며, 주어진 데이터에 맞추어진 원래 함수를 흉내내는 함수인 근사 함수를 의미합니다. 여기서는 손글씨 숫자를 구분하는 근사함수입니다.

10 : `tf.keras.Input` 함수를 이용하여 내부적으로 keras 라이브러리에서 제공하는 tensor를 생성하고, 입력 노드의 개수를 정해줍니다. tensor는 3차원 이상의 행렬식을 의미하며, 인공 신경망 구성시 사용하는 자료형입니다. 여기서는 784개의 입력 노드를 생성합니다.

11 : `tf.keras.layers.Dense` 클래스를 이용하여 신경 망 층을 생성합니다. 여기서는 단위 인공 신경 128개를 생성합니다. activation은 활성화 함수를 의미하며 여기서는 'relu' 함수를 사용합니다. 다음은 relu 함수를 나타냅니다.

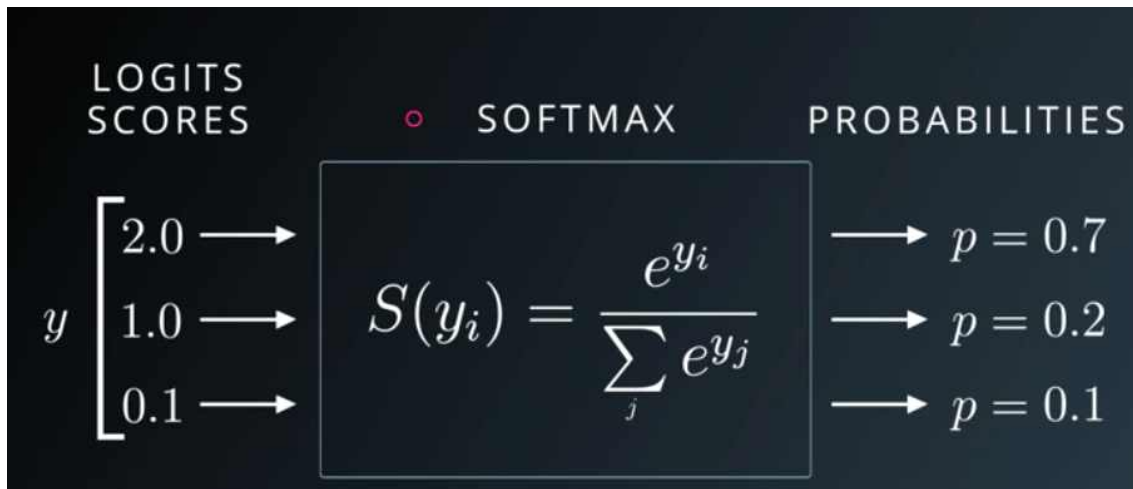
ReLU
 $\max(0, x)$



활성화 함수와 'relu' 함수에 대해서는 뒤에서 자세히 살펴보도록 합니다.

여기서 Dense는 내부적으로 $y = \text{activation}(x \cdot w + b)$ 식을 생성하게 됩니다. 이 식에 대해서는 뒤에서 실제로 구현해 보며 그 원리를 살펴 보도록 합니다.

12 : `tf.keras.layers.Dense` 클래스를 이용하여 신경 망 층을 생성합니다. 여기서는 단위 인공 신경 10개를 생성합니다. activation은 활성화 함수를 의미하며 여기서는 'softmax' 함수를 사용합니다. 다음은 softmax 함수를 나타냅니다.




<<https://laptrinhx.com/understand-the-softmax-function-in-minutes-1261253144>>

활성화 함수와 'softmax' 함수에 대해서는 뒤에서 자세히 살펴보도록 합니다.

15~17 : `model.compile` 함수를 호출하여 내부적으로 인공 신경망을 구성합니다. 인공 신경망을 구성할 때에는 적어도 2개의 함수를 정해야 합니다. `loss` 함수와 `optimizer` 함수, 즉, 손실 함수와 최적화 함수를 정해야 합니다. 손실 함수와 최적화 함수에 대해서는 뒤에서 자세히 살펴보겠습니다. 손실 함수로는 `sparse_categorical_crossentropy` 함수를 사용하고 최적화 함수는 `adam` 함수를 사용합니다. `sparse_categorical_crossentropy`, `adam` 함수는 뒤에서 살펴 보도록 합니다. `fit` 함수 로그에는 기본적으로 손실값만 표시됩니다. `metrics` 매개 변수는 학습 측정 항목 함수를 전달할 때 사용합니다. 'accuracy'는 학습의 정확도를 출력해 줍니다.

19 : `model.fit` 함수를 호출하여 인공 신경망에 대한 학습을 시작합니다. `fit` 함수에는 `x_train`, `y_train` 데이터가 입력이 되는데 인공 신경망을 `x_train`, `y_train` 데이터에 맞도록 학습한다는 의미를 갖습니다. 즉, `x_train`, `y_train` 데이터에 맞도록 인공 신경망을 조물조물, 주물주물 학습한다는 의미입니다. `fit` 함수에는 학습을 몇 회 수행할지도 입력해 줍니다. `epochs`는 학습 횟수를 의미하며, 여기서는 5회 학습을 수행하도록 합니다. 일반적으로 학습 횟수에 따라 인공 신경망 근사 함수가 정확해 집니다.

21 : `model.evaluate` 함수를 호출하여 인공 신경망의 학습 결과를 평가합니다. 여기서는 학습이 끝난 인공 신경망 함수에 `x_test` 값을 주어 학습 결과를 평가해 봅니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
Epoch 1/5
1875/1875 [=====] - 5s 3ms/step - loss: 0.4329 - accuracy: 0.8786
Epoch 2/5
1875/1875 [=====] - 5s 2ms/step - loss: 0.1204 - accuracy: 0.9645
Epoch 3/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0763 - accuracy: 0.9779
Epoch 4/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0558 - accuracy: 0.9834
Epoch 5/5
1875/1875 [=====] - 5s 2ms/step - loss: 0.0438 - accuracy: 0.9871
313/313 [=====] - 1s 1ms/step - loss: 0.0776 - accuracy: 0.9767
[0.0776418074965477, 0.9767000079154968]
```

① 손실 함수에 의해 측정된 오차 값을 나타냅니다. 학습 회수가 늘어남에 따라 오차 값이 줄어듭니다.

② 학습 진행에 따른 정확도가 표시됩니다. 처음에 87.86%에서 시작해서 마지막엔 98.71%의

정확도로 학습이 끝납니다. 즉, 100개의 손글씨가 있다면 98.71개를 맞춘다는 의미입니다.

❸ 학습이 끝난 후에, evaluate 함수로 시험 데이터를 평가한 결과입니다. 손실값이 늘어났고, 정확도가 97.67%로 약간 떨어진 상태입니다.

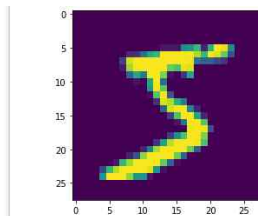
❹ 평가한 결과가 표시됩니다.

연습문제

학습 회수를 다음과 같이 50회로 늘려 정확도를 개선해 봅니다.

```
19 model.fit(X,YT,epochs=50)
```

02 손글씨 숫자 인식 예제 살펴보기



여기서는 앞에서 전체적으로 실행해 본 예제를 단계별로 살펴봅니다. 즉, 입력 데이터의 모양도 살펴보고 학습을 수행한 후, 학습에 사용하지 않은 손글씨 숫자를 얼마나 잘 인식하는지 살펴봅니다. 또 잘못 인식한 숫자를 독자 여러분이 직접 화면에 그려보며, 인공 신경망의 인식 성능을 확인해 봅니다.

데이터 모양 살펴보기

여기서는 먼저 MNIST 손글씨 숫자 데이터의 개수와 형식을 살펴보도록 합니다.

1. 다음과 같이 예제를 작성합니다.

142_1.py


```
1 import tensorflow as tf
2
3 mnist = tf.keras.datasets.mnist
4
5 (X, YT), (x, yt) = mnist.load_data() #60000개의 학습데이터, 10000개의 검증데이터
6 print(X.shape, YT.shape, x.shape, yt.shape)
```

01 : import문을 이용하여 tensorflow 모듈을 tf라는 이름으로 불러옵니다. tensorflow 모듈은 구글에서 제공하는 인공 신경망 라이브러리입니다.

03 : mnist 변수를 생성한 후, tf.keras.datasets.mnist 모듈을 가리키게 합니다. mnist 모듈은 손글씨 숫자 데이터를 가진 모듈입니다. mnist 모듈에는 6만개의 학습용 손글씨 숫자 데이터와 1만개의 시험용 손글씨 숫자 데이터가 있습니다. 이 데이터들에 대해서는 다음 단원에서 자세히 살펴봅니다.

05 : mnist.load_data 함수를 호출하여 손글씨 숫자 데이터를 읽어와 x_train, y_train, x_test, y_test 변수가 가리키게 합니다.

06, 07 : print 함수를 호출하여 x_train, y_train, x_test, y_test의 데이터 모양을 출력합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
x_train:(60000, 28, 28) y_train:(60000,) x_test:(10000, 28, 28) y_test:(10000,)
```

x_train, y_train 변수는 각각 6만개의 학습용 손글씨 숫자 데이터와 숫자 라벨을 가리킵니다. x_test, y_test 변수는 각각 1만개의 시험용 손글씨 숫자 데이터와 숫자 라벨을 가리킵니다. x_train, x_test 변수가 가리키는 6만개, 1만개의 그림은 각각 28x28 픽셀, 28x28 픽셀로 구성되어 있습니다.

학습 데이터 그림 그려보기

여기서는 학습용 데이터의 그림을 화면에 출력해 봅니다.

1. 다음과 같이 예제를 수정합니다.

142_2.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (X, YT), (x, yt) = mnist.load_data() #60000개의 학습데이터, 10000개의 검증데이터
06 print(X.shape, YT.shape, x.shape, yt.shape)
07
08 import matplotlib.pyplot as plt
09
10 plt.imshow(X[0])
11 plt.show()
12
13 print(YT[0])
14 print(tf.one_hot(YT[0], 10))
```


09 : import문을 이용하여 matplotlib.pyplot 모듈을 plt라는 이름으로 불러옵니다. 여기서는 matplotlib.pyplot 모듈을 이용하여 11~13줄에서 그래프를 그립니다.

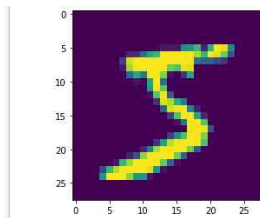
11 : plt.figure 함수를 호출하여 새로운 그림을 만들 준비를 합니다. figure 함수는 내부적으로

로 그림을 만들고 편집할 수 있게 해 주는 함수입니다.

12 : plt.imshow 함수를 호출하여 x_train[0] 항목의 그림을 내부적으로 그립니다.

13 : plt.show 함수를 호출하여 내부적으로 그린 그림을 화면에 그립니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.



x_train[0] 항목의 손글씨 숫자 그림은 5입니다.

그림 픽셀 값 출력해 보기

여기서는 앞에서 출력한 그림의 픽셀 값을 출력해 봅니다.

1. 다음과 같이 예제를 수정합니다.

142_3.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (X, YT), (x, yt) = mnist.load_data() #60000개의 학습데이터, 10000개의 검증데이터
06 print(X.shape, YT.shape, x.shape, yt.shape)
07
08 import matplotlib.pyplot as plt
09
10 plt.imshow(X[0])
11 plt.show()
12
13 print(YT[0])
14 print(tf.one_hot(YT[0], 10))
15
16 for row in range(28):
17     for col in range(28):
18         print('%4s' %X[0][row][col], end='')
19     print()
```

15 : 그림의 세로 28 줄에 대해

17 : 각 픽셀값을 출력합니다.

[illegible]

학습 데이터 그림 그려보기 2

142_4.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (X, YT), (x, yt) = mnist.load_data() #60000개의 학습데이터, 10000개의 검증데이터
06 print(X.shape, YT.shape, x.shape, yt.shape)
07
08 import matplotlib.pyplot as plt
09
10 plt.imshow(X[0])
11 plt.show()
12
13 print(YT[0])
14 print(tf.one_hot(YT[0], 10))
15
```

```

16 for row in range(28):
17     for col in range(28):
18         print('%4s' %X[0][row][col], end='')
19     print()
20
21 import numpy as np
22
23 plt.figure(figsize=(10,10))
24 for i in range(25):
25     plt.subplot(5,5,i+1)
26     plt.xticks([])
27     plt.yticks([])
28     plt.imshow(X[i],cmap=plt.cm.binary)
29     plt.xlabel(YT[i])
30 plt.show()

```

20 : plt.figure 함수를 호출하여 새로운 그림을 만들 준비를 합니다. figure 함수는 내부적으로 그림을 만들고 편집할 수 있게 해 주는 함수입니다. figsize는 그림의 인치 단위의 크기를 나타냅니다. 여기서는 가로 10인치, 세로 10인치의 그림을 그린다는 의미입니다.

21 : 0에서 24에 대해


22 : plt.subplot 함수를 호출하여 그림 창을 분할하여 하위 그림을 그립니다. 5,5는 각각 행의 개수와 열의 개수를 의미합니다. i+1은 하위 그림의 위치를 나타냅니다.

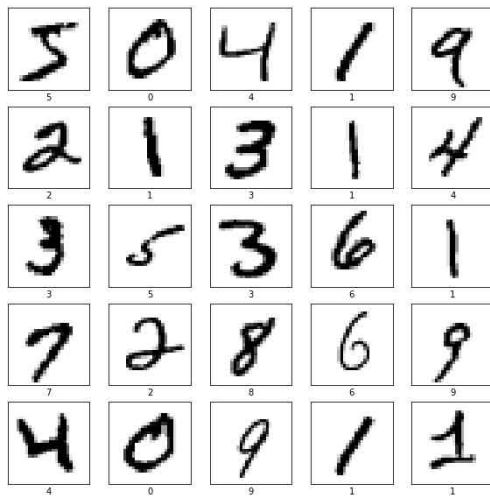
23, 24 : plt.xticks, plt.yticks 함수를 호출하여 x, y 축 눈금을 설정합니다. 여기서는 빈 목록을 주어 눈금 표시를 하지 않습니다.

25 : plt.imshow 함수를 호출하여 x_train[i] 항목의 그림을 내부적으로 그립니다. cmap는 color map의 약자로 binary는 그림을 이진화해서 표현해 줍니다.

26 : plt.xlabel 함수를 호출하여 x 축에 라벨을 붙여줍니다. 라벨의 값은 y_train[i]입니다.

27 : plt.show 함수를 호출하여 내부적으로 그린 그림을 화면에 그립니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.



x_train 변수가 가리키는 손글씨 숫자 그림 25개를 볼 수 있습니다. x_train 변수는 이런 그림을 6만개를 가리키고 있습니다.

인공 신경망 학습 시키기

여기서는 이전 예제에서 살펴본 손글씨 숫자 데이터를 이용하여 인공 신경망을 학습시켜 봅니다. 인공 신경망은 앞에서 구성했던 신경망을 그대로 사용합니다.

1. 다음과 같이 예제를 수정합니다.

142_5.py

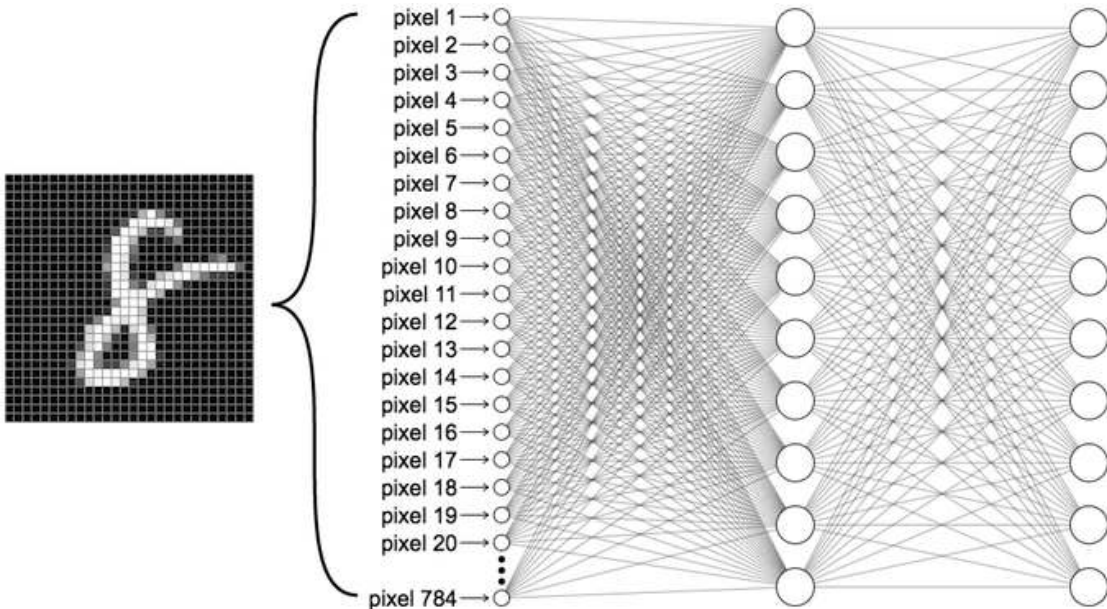
```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (X, Yt), (x, yt) = mnist.load_data() #60000개의 학습데이터, 10000개의 검증데이터
06
07 X, x = X/255, x/255 # 60000x28x28, 10000x28x28
08 X, x = X.reshape((60000,784)), x.reshape((10000,784))
09
10 model = tf.keras.Sequential([
11     tf.keras.Input(shape=(784,)),
12     tf.keras.layers.Dense(128, activation='relu'),
13     tf.keras.layers.Dense(10, activation='softmax')
14 ]) # 신경망 모양 결정(W, B 내부적 준비)
15
16 model.compile(optimizer='adam',
17               loss='sparse_categorical_crossentropy',
```

```
18         metrics=['accuracy'])
19
20 model.fit(X,YT,epochs=5)
21
22 model.evaluate(x,yt)
```

29 : x_train, x_test 변수가 가리키는 6만개, 1만개의 그림은 각각 28x28 픽셀로 구성된 그림이며, 1픽셀의 크기는 8비트로 0에서 255사이의 숫자를 가집니다. 모든 픽셀의 숫자를 255.0으로 나누어 각 픽셀을 0.0에서 1.0사이의 실수로 바꾸어 인공 신경망에 입력하게 됩니다. 다음은 x_train[0] 그림의 픽셀값을 출력한 그림입니다.

[illegible]

30 : x_train, x_test 변수가 가리키는 6만개, 1만개의 그림은 각각 28x28 픽셀, 28x28 픽셀로 구성되어 있습니다. 이 예제에서 소개하는 인공 신경망의 경우 그림 데이터를 입력할 때 28x28 픽셀을 784(=28x28) 픽셀로 일렬로 세워서 입력하게 됩니다. 그래서 33줄에 있는 Input 클래스는 일렬로 세워진 784 픽셀을 입력으로 받도록 구성됩니다.



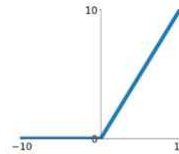
<출처 : <https://www.kdnuggets.com/2019/11/designing-neural-networks.html>>

32~36 : `tf.keras.Sequential` 클래스를 이용하여 인공 신경 망을 생성합니다. 여기서 생성한 인공 신경망은 138(=128+10)개의 인공 신경으로 구성됩니다. 입력층에 표시된 노드는 입력값의 개수를 표시하며 나머지 층에 있는 노드는 인공 신경을 나타냅니다. 인공 신경망의 내부 구조는 뒤에서 자세히 살펴봅니다. 생성된 인공 신경망은 일반적으로 모델이라고 합니다. 모델은 모형을 의미하며, 주어진 데이터에 맞추어진 원래 함수를 흉내내는 함수인 근사 함수를 의미합니다. 여기서는 손글씨 숫자를 구분하는 근사함수입니다.

33 : `tf.keras.Input` 함수를 이용하여 내부적으로 keras 라이브러리에서 제공하는 tensor를 생성하고, 입력 노드의 개수를 정해줍니다. tensor는 3차원 이상의 행렬식을 의미하며, 인공 신경망 구성시 사용하는 자료형입니다. 여기서는 784개의 입력 노드를 생성합니다.

34 : `tf.keras.layers.Dense` 클래스를 이용하여 신경 망 층을 생성합니다. 여기서는 단위 인공 신경 128개를 생성합니다. activation은 활성화 함수를 의미하며 여기서는 'relu' 함수를 사용합니다. 다음은 relu 함수를 나타냅니다.

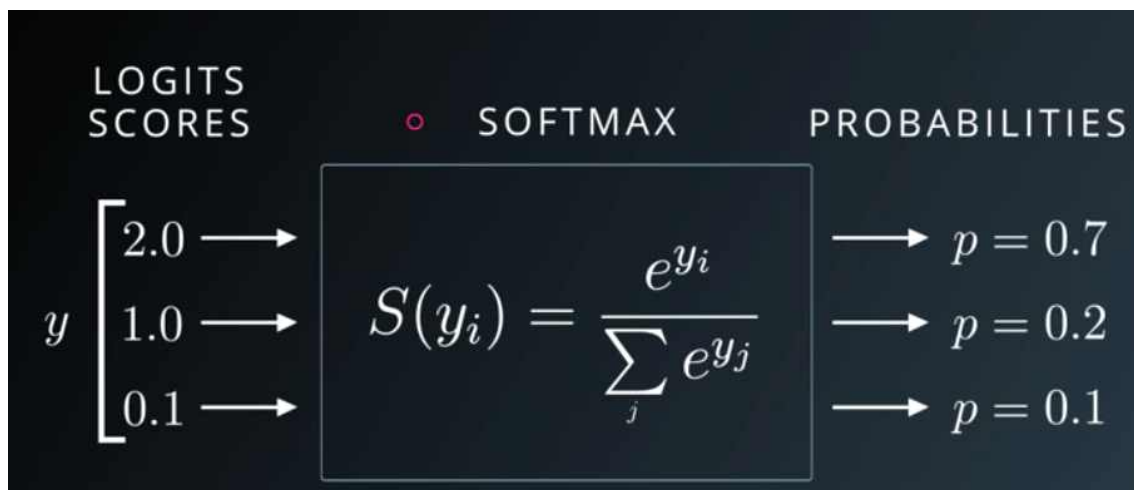
ReLU
 $\max(0, x)$



활성화 함수와 'relu' 함수에 대해서는 뒤에서 자세히 살펴보도록 합니다.

여기서 Dense는 내부적으로 $y = \text{activation}(x \cdot w + b)$ 식을 생성하게 됩니다. 이 식에 대해서는 뒤에서 실제로 구현해 보며 그 원리를 살펴 보도록 합니다.

35 : `tf.keras.layers.Dense` 클래스를 이용하여 신경 망 층을 생성합니다. 여기서는 단위 인공 신경 10개를 생성합니다. activation은 활성화 함수를 의미하며 여기서는 'softmax' 함수를 사용합니다. 다음은 softmax 함수를 나타냅니다.



<<https://laptrinhx.com/understand-the-softmax-function-in-minutes-1261253144>>


활성화 함수와 'softmax' 함수에 대해서는 뒤에서 자세히 살펴보도록 합니다.

38~40 : `model.compile` 함수를 호출하여 내부적으로 인공 신경망을 구성합니다. 인공 신경망을 구성할 때에는 적어도 2개의 함수를 정해야 합니다. loss 함수와 optimizer 함수, 즉, 손실 함수와 최적화 함수를 정해야 합니다. 손실 함수와 최적화 함수에 대해서는 뒤에서 자세히 살펴봅니다. 손실 함수로는 `sparse_categorical_crossentropy` 함수를 사용하고 최적화 함수는 `adam` 함수를 사용합니다. `sparse_categorical_crossentropy`, `adam` 함수는 뒤에서

살펴 보도록 합니다. fit 함수 로그에는 기본적으로 손실값만 표시됩니다. metrics 매개 변수는 학습 측정 항목 함수를 전달할 때 사용합니다. 'accuracy'는 학습의 정확도를 출력해 줍니다.

42 : model.fit 함수를 호출하여 인공 신경망에 대한 학습을 시작합니다. fit 함수에는 x_train, y_train 데이터가 입력이 되는데 인공 신경망을 x_train, y_train 데이터에 맞도록 학습한다는 의미를 갖습니다. 즉, x_train, y_train 데이터에 맞도록 인공 신경망을 조물조물, 주물주물 학습한다는 의미입니다. fit 함수에는 학습을 몇 회 수행할지도 입력해 줍니다. epochs는 학습 횟수를 의미하며, 여기서는 5회 학습을 수행하도록 합니다. 일반적으로 학습 횟수에 따라 인공 신경망 근사 함수가 정확해 집니다.

44 : model.evaluate 함수를 호출하여 인공 신경망의 학습 결과를 평가합니다. 여기서는 학습이 끝난 인공 신경망 함수에 x_test 값을 주어 학습 결과를 평가해 봅니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
Epoch 1/5  
1875/1875 [=====] - 5s 2ms/step - loss: 0.4268 - accuracy: 0.8779  
Epoch 2/5  
1875/1875 [=====] - 4s 2ms/step - loss: 0.1233 - accuracy: 0.9637  
Epoch 3/5  
1875/1875 [=====] - 4s 2ms/step - loss: 0.0780 - accuracy: 0.9762  
Epoch 4/5  
1875/1875 [=====] - 4s 2ms/step - loss: 0.0586 - accuracy: 0.9827  
Epoch 5/5  
1875/1875 [=====] - 4s 2ms/step - loss: 0.0447 - accuracy: 0.9860  
313/313 [=====] - 1s 1ms/step - loss: 0.0705 - accuracy: 0.9776
```

- ① 손실 함수에 의해 측정된 오차 값을 나타냅니다. 학습 회수가 늘어남에 따라 오차 값이 줄어듭니다.
- ② 학습 진행에 따른 정확도가 표시됩니다. 처음에 87.86%에서 시작해서 마지막엔 98.71%의 정확도로 학습이 끝납니다. 즉, 100개의 손글씨가 있다면 98.71개를 맞춘다는 의미입니다.
- ③ 학습이 끝난 후에, evaluate 함수로 시험 데이터를 평가한 결과입니다. 손실값이 늘어났고, 정확도가 97.67%로 약간 떨어진 상태입니다.

학습된 인공 신경망 저장하기

```
01 import tensorflow as tf  
02  
03 mnist = tf.keras.datasets.mnist  
04  
05 (X, YT), (x, yt) = mnist.load_data() #60000개의 학습데이터, 10000개의 검증데이터  
06  
07 X, x = X/255, x/255 # 60000x28x28, 10000x28x28  
08 X, x = X.reshape((60000,784)), x.reshape((10000,784))  
09  
10 model = tf.keras.Sequential([  
11     tf.keras.Input(shape=(784,)),  
12     tf.keras.layers.Dense(128, activation='relu'),  
13     tf.keras.layers.Dense(10, activation='softmax')  
14 ]) # 신경망 모양 결정(W, B 내부적 준비)
```

```

15
16 model.compile(optimizer='adam',
17               loss='sparse_categorical_crossentropy',
18               metrics=['accuracy'])
19
20 model.fit(X,YT,epochs=5)
21
22 model.evaluate(x,yt)
23
24 model.save('model.h5')

```

학습된 인공 신경망 시험하기

여기서는 학습된 신경망에 시험 데이터를 입력하여 예측해 봅니다.

1. 다음과 같이 예제를 수정합니다.

142_6.py

```

01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (X, YT), (x, yt) = mnist.load_data() #60000개의 학습데이터, 10000개의 검증데이터
06
07 X, x = X/255, x/255 # 60000x28x28, 10000x28x28
08 X, x = X.reshape((60000,784)), x.reshape((10000,784))
09
10 model = tf.keras.models.load_model('model.h5')
11
12 y=model.predict(x)
13 print(y[0])

```

46 : model_f.predict 함수를 호출하여 인공 신경망을 시험합니다. 여기서는 학습이 끝난 인공 신경망 함수에 x_test 값을 주어 그 결과를 예측해 봅니다. 예측한 결과값은 p_test 변수로 받습니다. x_test는 1만개의 손글씨 숫자를 가리키고 있으며, 따라서 1만개에 대한 예측을 수행합니다.

47 : print 함수를 호출하여 x_test[0] 손글씨 숫자의 예측값을 출력합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

p_test [0] : [1.07918666e-07 4.64393946e-09 3.81268319e-06 5.01001021e-04
2.92917357e-10 3.51789765e-07 5.53046931e-12 9.99484420e-01
9.99521490e-07 9.21584615e-06]

```

p_test[0]은 x_test[0]이 가리키는 손글씨 숫자에 대해 0~9 각각의 숫자에 대한 확률값 목록을 출력합니다. x_test[0]은 실제로 숫자 7을 가리키고 있습니다. 그래서 p_test[0]의 8번째 값의 확률이 가장 높게 나타납니다. 8번째 값은 9.99484420e-01이며 99.9%로 7이라고 예측합니다. p_test[0]의 1번째 값은 숫자 0일 확률을 나타냅니다.

예측값과 실제값 출력해 보기

여기서는 예측값과 실제값을 출력해 봅니다.


1. 다음과 같이 예제를 수정합니다.

142_7.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (X, YT), (x, yt) = mnist.load_data() #60000개의 학습데이터, 10000개의 검증데이터
06
07 X, x = X/255, x/255 # 60000x28x28, 10000x28x28
08 X, x = X.reshape((60000,784)), x.reshape((10000,784))
09
10 model = tf.keras.models.load_model("model.h5")
11
12 y=model.predict(x)
13 print(y[0])
14
15 import numpy as np
16
17 print(np.argmax(y[0]), yt[0])
```

49 : import문을 이용하여 numpy 모듈을 np라는 이름으로 불러옵니다. 여기서는 numpy 모듈의 argmax 함수를 이용하여 51 줄에서 p_test[0] 항목의 가장 큰 값의 항목 번호를 출력합니다.

51, 52 : print 함수를 호출하여 p_test[0] 항목의 가장 큰 값의 항목 번호와 y_test[0] 항목이 가리키는 실제 라벨값을 출력합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
p_test[0] : 7 y_test[0] : 7
```

p_test[0] 항목의 가장 큰 값의 항목 번호와 y_test[0] 항목이 가리키는 실제 라벨값이 같습니다. x_test[0] 항목의 경우 예측값과 실제값이 같아 인공 신경망이 옳게 예측합니다.

시험 데이터 그림 그려보기

여기서는 시험용 데이터의 그림을 화면에 출력해 봅니다.

1. 다음과 같이 예제를 수정합니다.

142_8.py


```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (X, YT), (x, yt) = mnist.load_data() #60000개의 학습데이터, 10000개의 검증데이터
06
07 X, x = X/255, x/255 # 60000x28x28, 10000x28x28
08 X, x = X.reshape((60000,784)), x.reshape((10000,784))
09
10 model = tf.keras.models.load_model("model.h5")
11
12 y=model.predict(x)
13 print(y[0])
14
15 import numpy as np
16
17 print(np.argmax(y[0]), yt[0])
18
19 import matplotlib.pyplot as plt
20
21 x=x.reshape(10000,28,28)
22
23 plt.imshow(x[0])
24 plt.show()
```

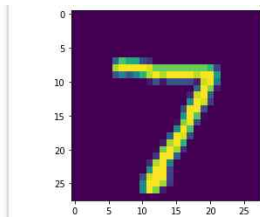
54 : reshape 함수를 호출하여 x_test가 가리키는 그림을 원래 모양으로 돌려 놓습니다. 그래야 pyplot 모듈을 이용하여 그림을 화면에 표시할 수 있습니다.

56 : plt.figure 함수를 호출하여 새로운 그림을 만들 준비를 합니다. figure 함수는 내부적으로 그림을 만들고 편집할 수 있게 해 주는 함수입니다.

57 : plt.imshow 함수를 호출하여 x_test[0] 항목의 그림을 내부적으로 그립니다.

58 : plt.show 함수를 호출하여 내부적으로 그린 그림을 화면에 그립니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.



x_test[0] 항목의 손글씨 숫자 그림은 7입니다.

시험 데이터 그림 그려보기 2

여기서는 시험 데이터의 그림 25개를 화면에 출력해 봅니다.

1. 다음과 같이 예제를 수정합니다.

142_9.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (X, YT), (x, yt) = mnist.load_data() #60000개의 학습데이터, 10000개의 검증데이터
06
07 X, x = X/255, x/255 # 60000x28x28, 10000x28x28
08 X, x = X.reshape((60000,784)), x.reshape((10000,784))
09
10 model = tf.keras.models.load_model("model.h5")
11
12 y=model.predict(x)
13 print(y[0])
14
15 import numpy as np
16
17 print(np.argmax(y[0]), yt[0])
18
19 import matplotlib.pyplot as plt
20
21 x=x.reshape(10000,28,28)
22
23 plt.imshow(x[0])
24 plt.show()
25
26 plt.figure(figsize=(10,10))
```

```

27 for i in range(25):
28     plt.subplot(5,5,i+1)
29     plt.xticks([])
30     plt.yticks([])
31     plt.imshow(x[i], cmap=plt.cm.binary)
32     plt.xlabel(np.argmax(y[i]))
33 plt.show()

```

60 : plt.figure 함수를 호출하여 새로운 그림을 만들 준비를 합니다. figure 함수는 내부적으로 그림을 만들고 편집할 수 있게 해 주는 함수입니다. figsize는 그림의 인치 단위의 크기를 나타냅니다. 여기서는 가로 10인치, 세로 10인치의 그림을 그린다는 의미입니다.

61 : 0에서 24에 대해


62 : plt.subplot 함수를 호출하여 그림 창을 분할하여 하위 그림을 그립니다. 5,5는 각각 행의 개수와 열의 개수를 의미합니다. i+1은 하위 그림의 위치를 나타냅니다.

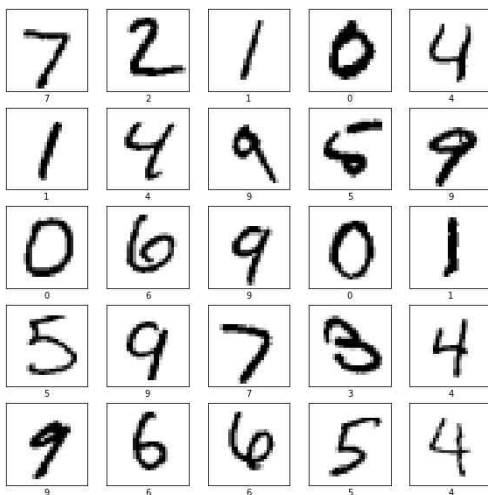
63, 64 : plt.xticks, plt.yticks 함수를 호출하여 x, y 축 눈금을 설정합니다. 여기서는 빈 목록을 주어 눈금 표시를 하지 않습니다.

65 : plt.imshow 함수를 호출하여 x_test[i] 항목의 그림을 내부적으로 그립니다. cmap은 color map의 약자로 binary는 그림을 이진화해서 표현해 줍니다.

66 : plt.xlabel 함수를 호출하여 x 축에 라벨을 붙여줍니다. 라벨의 값은 y_train[i]입니다.

67 : plt.show 함수를 호출하여 내부적으로 그린 그림을 화면에 그립니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.



x_test 변수가 가리키는 손글씨 숫자 그림 25개를 볼 수 있습니다. x_test 변수는 이런 그림을 1만개를 가리키고 있습니다.

잘못된 예측 살펴보기

여기서는 시험 데이터 중 잘못된 예측이 몇 개나 되는지 또 몇 번째 그림이 잘 못 예측되었는

지 살펴보도록 합니다.

1. 다음과 같이 예제를 수정합니다.

142_10.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (X, YT), (x, yt) = mnist.load_data() #60000개의 학습데이터, 10000개의 검증데이터
06
07 X, x = X/255, x/255 # 60000x28x28, 10000x28x28
08 X, x = X.reshape((60000,784)), x.reshape((10000,784))
09
10 model = tf.keras.models.load_model("model.h5")
11
12 y=model.predict(x)
13
14 import numpy as np
15 import matplotlib.pyplot as plt
16
17 x=x.reshape(10000,28,28)
18
19 cnt_wrong=0
20 y_wrong=[]
21 for i in range(10000):
22     if np.argmax(y[i]) != yt[i]:
23         y_wrong.append(i)
24         cnt_wrong += 1
25
26 print(cnt_wrong)
27 print(y_wrong[:10])
```

69 : cnt_wrong 변수를 선언한 후, 0으로 초기화합니다. cnt_wrong 변수는 잘못 예측된 그림의 개수를 저장할 변수입니다.

70 : p_wrong 변수를 선언한 후, 빈 목록으로 초기화합니다. p_wrong 변수는 잘못 예측된 그림의 번호를 저장할 변수입니다.

71 : 0부터 10000미만까지


72 : p_test[i] 항목의 가장 큰 값의 항목 번호와 y_test[0] 항목이 가리키는 실제 라벨값이 다르면

73 : p_wrong 목록에 해당 그림 번호를 추가하고

74 : cnt_wrong 값을 하나 증가시킵니다.

76 : print 함수를 호출하여 cnt_wrong 값을 출력합니다.

77 : print 함수를 호출하여 p_wrong에 저장된 값 중, 앞에서 10개까지 출력합니다.
p_wrong[:10]은 p_wrong 목록의 0번 항목부터 시작해서 10번 항목 미만인 9번 항목까지를 의미합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
cnt_wrong : 224  
predicted wrong 10 : [115, 247, 320, 321, 340, 381, 445, 582, 610, 619]
```

학습이 끝난 인공 신경망은 시험 데이터에 대해 10000개 중 224개에 대해 잘못된 예측을 하였습니다. 즉, 10000개 중 9776(=10000-224)개는 바르게 예측을 했으며, 나머지 224개에 대해서는 잘못된 예측을 하였습니다. 예측 정확도는 97.76%, 예측 오류도는 2.24%입니다. 잘못 예측한 데이터 번호 10개에 대해서도 확인해 봅니다. 115번 데이터로 시작해서 619번 데이터 까지 10개의 데이터 번호를 출력하고 있습니다.

잘못 예측한 그림 살펴보기

여기서는 시험 데이터 중 잘못 예측한 그림을 출력해봅니다.

1. 다음과 같이 예제를 수정합니다.

142_11.py

```
01 import tensorflow as tf  
02  
03 mnist = tf.keras.datasets.mnist  
04  
05 (X, YT), (x, yt) = mnist.load_data() #60000개의 학습데이터, 10000개의 검증데이터  
06  
07 X, x = X/255, x/255 # 60000x28x28, 10000x28x28  
08 X, x = X.reshape((60000,784)), x.reshape((10000,784))  
09  
10 model = tf.keras.models.load_model("model.h5")  
11  
12 y=model.predict(x)  
13  
14 import numpy as np  
15 import matplotlib.pyplot as plt  
16  
17 x=x.reshape(10000,28,28)  
18  
19 cnt_wrong=0
```

```

20 y_wrong=[]
21 for i in range(10000):
22     if np.argmax(y[i]) != yt[i]:
23         y_wrong.append(i)
24         cnt_wrong += 1
25
26 print(cnt_wrong)
27 print(y_wrong[:10])
28
29 plt.figure(figsize=(10,10))
30 for i in range(25):
31     plt.subplot(5,5,i+1)
32     plt.xticks([])
33     plt.yticks([])
34     plt.imshow(x[y_wrong[i]],cmap=plt.cm.binary)
35     plt.xlabel(f'{y_wrong[i]} : y{np.argmax(y[y_wrong[i]])} yt{yt[y_wrong[i]]}')
36 plt.show()

```

79 : plt.figure 함수를 호출하여 새로운 그림을 만들 준비를 합니다. figure 함수는 내부적으로 그림을 만들고 편집할 수 있게 해 주는 함수입니다. figsize는 그림의 인치 단위의 크기를 나타냅니다. 여기서는 가로 10인치, 세로 10인치의 그림을 그린다는 의미입니다.

80 : 0에서 24에 대해


81 : plt.subplot 함수를 호출하여 그림 창을 분할하여 하위 그림을 그립니다. 5,5는 각각 행의 개수와 열의 개수를 의미합니다. i+1은 하위 그림의 위치를 나타냅니다.

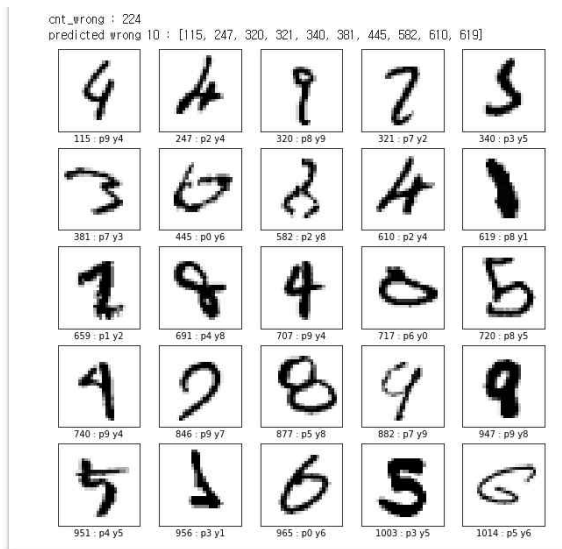
82, 83 : plt.xticks, plt.yticks 함수를 호출하여 x, y 축 눈금을 설정합니다. 여기서는 빈 목록을 주어 눈금 표시를 하지 않습니다.

84 : plt.imshow 함수를 호출하여 x_test[p_wrong[i]] 항목의 그림을 내부적으로 그립니다. cmap는 color map의 약자로 binary는 그림을 이진화해서 표현해 줍니다.

85, 86 : plt.xlabel 함수를 호출하여 x 축에 라벨을 붙여줍니다. 라벨의 값은 잘못 예측한 그림 번호, 인공 신경망이 예측한 숫자값, 라벨에 표시된 숫자값으로 구성됩니다.

87 : plt.show 함수를 호출하여 내부적으로 그린 그림을 화면에 그립니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.



첫 번째 잘못 예측한 숫자는 115번째의 숫자이며, 인공 신경망은 9로 예측하였으며, 실제 값은 4입니다. 언뜻 보면 사람이 보기에 혼동될 수 있는 형태의 숫자들입니다.

연습문제

학습 회수를 다음과 같이 50회로 늘려 정확도를 개선해 봅니다.

142_11.py

```
42 : model.fit(x_train, y_train, epochs=50)
```

이상에서 손글씨 숫자를 이용하여 인공 신경망을 학습시키고, 학습시킨 결과를 예측하는 과정을 자세히 살펴보았습니다. 일반적으로 인공 신경망을 이용할 때, 달라지는 부분은 데이터의 종류와 인공 신경망의 구성 형태입니다. 나머지 부분은 프레임워크처럼 비슷한 형태로 작성될 가능성이 높습니다.

mnist 선형 회귀

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (X, YT), (x, yt) = mnist.load_data() #60000개의 학습데이터, 10000개의 검증데이터
06
07 X, x = X/255, x/255 # 60000x28x28, 10000x28x28
08 X, x = X.reshape((60000,784)), x.reshape((10000,784))
09
10 model = tf.keras.Sequential([
11     tf.keras.Input(shape=(784,)),
12     tf.keras.layers.Dense(128, activation='relu'),
13     tf.keras.layers.Dense(1, activation='linear')
14 ]) # 신경망 모양 결정(W, B 내부적 준비)
15
16 model.compile(optimizer='adam',
17               loss='mse')
18
19 model.fit(X, YT, epochs=20)
20
21 y=model.predict(x)
22 print(y[0])
23
24 import matplotlib.pyplot as plt
25
26 x=x.reshape(10000,28,28)
27
28 plt.figure(figsize=(10,10))
29 for i in range(25):
30     plt.subplot(5,5,i+1)
31     plt.xticks([])
32     plt.yticks([])
33     plt.imshow(x[i], cmap=plt.cm.binary)
34     plt.xlabel(f'y{y[i]} yt{yt[i]}')
35 plt.show()
```


03 패션 MNIST 데이터셋 인식시켜보기

여기서는 이전에 작성했던 예제를 부분적으로 수정해 가며 또다른 데이터셋인 패션 MNIST 데이터셋을 사용해서 인공 신경망을 학습시켜봅니다. 패션 MNIST 데이터셋은 10개의 범주(category)와 70,000개의 흑백 이미지로 구성됩니다. 패션 MNIST 데이터셋의 그림은 손글씨 데이터 셋과 마찬가지로 28x28 픽셀의 해상도를 가지며 다음처럼 신발, 옷, 가방 등의 품목을 나타냅니다.



1. 다음과 같이 예제를 수정합니다.

143_1.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.fashion_mnist
04
05 (X, YT), (x, yt) = mnist.load_data() #60000개의 학습데이터, 10000개의 검증데이터
06
07 X, x = X/255, x/255 # 60000x28x28, 10000x28x28
08 X, x = X.reshape((60000,784)), x.reshape((10000,784))
09
10 model = tf.keras.Sequential([
11     tf.keras.Input(shape=(784,)),
12     tf.keras.layers.Dense(128, activation='relu'),
13     tf.keras.layers.Dense(10, activation='softmax')
14 ]) # 신경망 모양 결정(W, B 내부적 준비)
15
16 model.compile(optimizer='adam',
17               loss='sparse_categorical_crossentropy',
18               metrics=['accuracy'])
```

```
19  
20 model.fit(X,YT,epochs=5)  
21  
22 model.evaluate(x,yt)
```

20, 21 : 패션 MNIST 데이터 셋을 이루는 품목의 종류는 손글씨 MNIST 데이터 셋과 마찬가지로 10가지로 구성되며, 품목의 라벨은 숫자 0~9로 구성됩니다. 여기서는 품목의 해당 라벨 값에 품목의 이름을 대응시킵니다. `class_names` 변수를 선언한 후, 품목의 이름 10가지로 초기화합니다. 예를 들어, 품목의 라벨 값 0은 'T-shirt/top'을 의미하며, 9는 'Ankle boot'를 의미합니다.

29 : y_train[i]는 숫자 라벨을 의미하므로 class_names[y_train[i]]로 변경해 줍니다.

54 : np.argmax(p_test[0])는 x_test[0] 항목에 대해 예측한 숫자값을 의미하므로
class_names[np.argmax(p_test[0])]로 변경해 줍니다.

55 : y_test[0]는 숫자 라벨을 의미하므로 class_names[y_test[0]]로 변경해 줍니다.

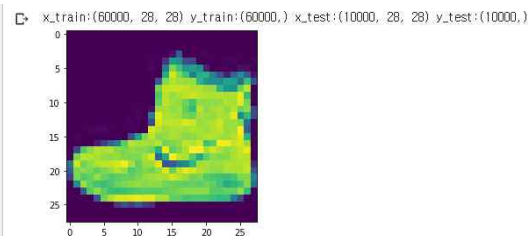
69 : np.argmax(p_test[i])는 x_test[i] 항목에 대해 예측한 숫자값을 의미하므로
class_names[np.argmax(p_test[i])]로 변경해 줍니다.

89 : np.argmax(p_test[p_wrong[i]])을 class_names[np.argmax(p_test[p_wrong[i]])]으로
변경해 줍니다.

90 : y_test[p_wrong[i]]을 class_names[y_test[p_wrong[i]]]으로 변경해 줍니다.



2. 버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.



x_train[0] 항목의 그림은 Ankle boot입니다.

[illegible]

각 픽셀의 값이 0~255 사이의 값에서 출력되는 것을 확인합니다.



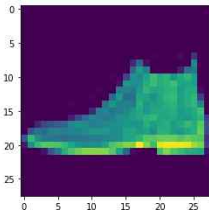
x_train 변수가 가리키는 패션 MNIST 그림 25개를 볼 수 있습니다. x_train 변수는 이런 그림을 6만개를 가리키고 있습니다.

```
Epoch 1/5
1875/1875 [=====] - 5s 3ms/step - loss: 0.6159 - accuracy: 0.7851
Epoch 2/5
1875/1875 [=====] - 5s 3ms/step - loss: 0.3814 - accuracy: 0.8625
Epoch 3/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.3335 - accuracy: 0.8781
Epoch 4/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.3142 - accuracy: 0.8852
Epoch 5/5
1875/1875 [=====] - 5s 2ms/step - loss: 0.2903 - accuracy: 0.8926
313/313 [=====] - 1s 2ms/step - loss: 0.3603 - accuracy: 0.8724
```

- ❶ 손실 함수에 의해 측정된 오차 값을 나타냅니다. 학습 회수가 늘어남에 따라 오차 값이 줄어듭니다.
- ❷ 학습 진행에 따른 정확도가 표시됩니다. 처음에 78.51%에서 시작해서 마지막엔 89.26%의 정확도로 학습이 끝납니다. 즉, 100개의 패션 MNIST 그림이 있다면 89.26개를 맞춘다는 의미입니다.
- ❸ 학습이 끝난 후에, evaluate 함수로 시험 데이터를 평가한 결과입니다. 손실값이 늘어났고, 정확도가 87.24%로 약간 떨어진 상태입니다.

```
p_test[0] : [1.0160066e-05 3.4290405e-08 3.1461993e-06 1.3471047e-09 1.1331924e-06
3.7246171e-04 2.3442435e-06 3.2645520e-02 2.5410671e-05 9.6693981e-01]
p_test[0] : 9 Ankle boot y_test[0] : 9 Ankle boot
```

p_test[0]은 x_test[0]이 가리키는 패션 MNIST 그림에 대해 0~9 각각의 숫자에 대한 확률값 목록을 출력합니다. x_test[0]은 실제로 숫자 9를 가리키고 있습니다. 그래서 p_test[0]의 9번째 값의 확률이 가장 높게 나타납니다. 9번째 값은 9.6693981e-01이며 96.7%로 9라고 예측합니다. p_test[0]의 1번째 값은 숫자 0일 확률을 나타냅니다. p_test[0] 항목의 가장 큰 값의 항목 번호와 y_test[0] 항목이 가리키는 실제 라벨값이 같습니다. x_test[0] 항목의 경우 예측값과 실제값이 같아 인공 신경망이 옳게 예측합니다.



`x_test[0]` 항목의 그림은 Ankle boot입니다.



`x_test` 변수가 가리키는 패션 MNIST 그림 25개를 볼 수 있습니다. `x_test` 변수는 이런 그림을 1만개를 가리키고 있습니다.



학습이 끝난 인공 신경망은 시험 데이터에 대해 10000개 중 1276개에 대해 잘못된 예측을 하였습니다. 즉, 10000개 중 8724(=10000-1276)개는 바르게 예측을 했으며, 나머지 1276개에 대해서는 잘못된 예측을 하였습니다. 예측 정확도는 87.24%, 예측 오류도는 12.76%입니다. 잘못 예측한 데이터 번호 10개에 대해서도 확인해 봅니다. 17번 데이터로 시작해서 67번 데이터까지 10개의 데이터 번호를 출력하고 있습니다. 첫 번째 잘못 예측한 패션 MNIST 그림은 17번째 그림이며, 인공 신경망은 Pullover로 예측하였으며, 실제 값은 Coat입니다. 두

번째 잘못 예측한 패션 MNIST 그림은 21번째의 그림이며, 인공 신경망은 Sneaker로 예측하였으며, 실제 값은 Sandal입니다. 언뜻 보면 사람이 보기에 혼동될 수 있는 형태의 품목들입니다.

연습문제

학습 회수를 다음과 같이 50회로 늘려 정확도를 개선해 봅시다.

143_1.py

```
45 : model.fit(x_train, y_train, epochs=50)
```

이상에서 이전 예제를 수정한 후, 패션 MNIST 그림을 이용하여 인공 신경망을 학습시키고, 학습시킨 결과를 예측하는 과정을 살펴보았습니다. 패션 MNIST 그림의 경우 숫자 MNIST 그림보다 인식률이 낮습니다. 패션 MNIST 그림에 대한 낮은 인식률은 인공 신경망의 구성을 바꾸어서 높일 수 있습니다.