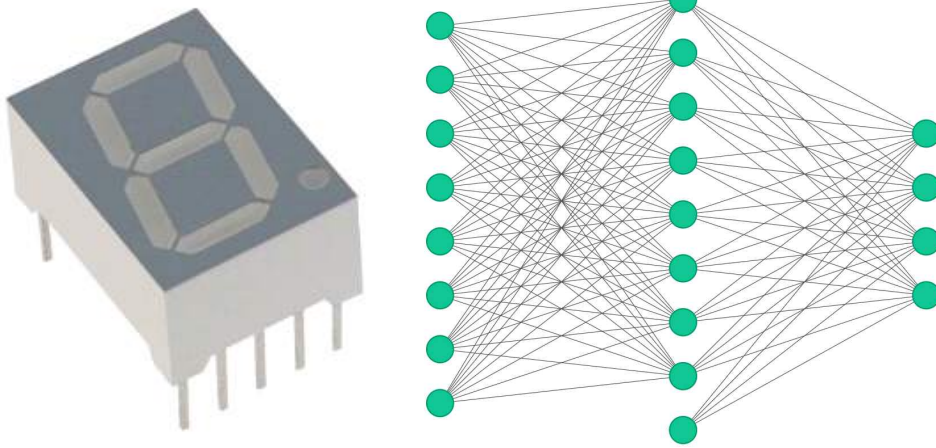


06 Tensorflow 활용하기

여기서는 Tensorflow를 활용하여 7 segment에 대한 인공 신경망을 학습시켜 봅니다. 이 과정에서 Tensorflow 신경망에 적용할 수 있는 입력 데이터와 출력 데이터의 형식을 이해하고 활용할 수 있도록 합니다.

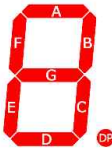


01 7 세그먼트 인공 신경망

여기서는 7 세그먼트에 숫자 값에 따라 표시되는 LED의 ON, OFF 값을 입력으로 받아 2 진수로 출력하는 인공 신경망을 구성하고 학습시켜 봅니다. 다음은 7 세그먼트 디스플레이 2 진수 연결 진리표입니다.

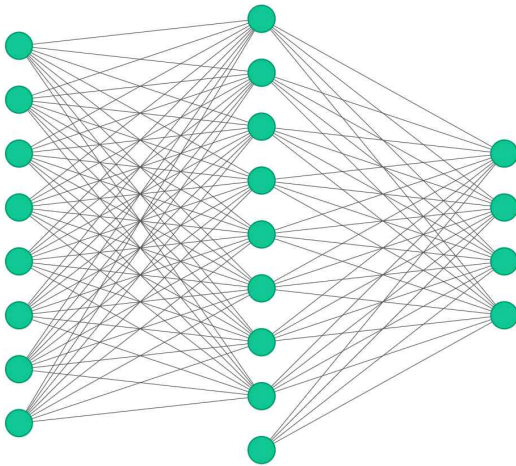
7 세그먼트 2 진수 연결 진리표

	A	B	C	D	E	F	G				
	In	In	In	In	In	In	In	Out	Out	Out	Out
0	1	1	1	1	1	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	1
2	1	1	0	1	1	0	1	0	0	1	0
3	1	1	1	1	0	0	1	0	0	1	1
4	0	1	1	0	0	1	1	0	1	0	0
5	1	0	1	1	0	1	1	0	1	0	1
6	0	0	1	1	1	1	1	0	1	1	0
7	1	1	1	0	0	0	0	0	1	1	1
8	1	1	1	1	1	1	1	1	0	0	0
9	1	1	1	0	0	1	1	1	0	0	1



5 = 1011011 → 0101

그림에서 7 세그먼트에 5로 표시되기 위해 7개의 LED가 1011011(1-ON, 0-OFF)의 비트열에 맞춰 켜지거나 꺼져야 합니다. 해당 비트열에 대응하는 이진수는 0101입니다. 여기서는 다음 그림과 같이 7개의 입력, 8개의 은닉층, 4개의 출력층으로 구성된 인공 신경망을 학습시켜 봅니다.



입력층, 은닉층의 맨 하단의 노드는 편향 노드입니다.

numpy 배열로 데이터 초기화하기

먼저 입력값과 목표값을 numpy 배열로 초기화합니다.

1. 다음과 같이 예제를 작성합니다.

_7seg_data.py

```
01 import numpy as np
02
03 np.set_printoptions(precision=1, suppress=True)
04
05 X=np.array([
06     [ 1, 1, 1, 1, 1, 1, 0 ], # 0
07     [ 0, 1, 1, 0, 0, 0, 0 ], # 1
08     [ 1, 1, 0, 1, 1, 0, 1 ], # 2
09     [ 1, 1, 1, 1, 0, 0, 1 ], # 3
10     [ 0, 1, 1, 0, 0, 1, 1 ], # 4
11     [ 1, 0, 1, 1, 0, 1, 1 ], # 5
12     [ 0, 0, 1, 1, 1, 1, 1 ], # 6
13     [ 1, 1, 1, 0, 0, 0, 0 ], # 7
14     [ 1, 1, 1, 1, 1, 1, 1 ], # 8
15     [ 1, 1, 1, 0, 0, 1, 1 ]  # 9
16 ])
17 YT=np.array([
18     [ 0, 0, 0, 0 ],
19     [ 0, 0, 0, 1 ],
20     [ 0, 0, 1, 0 ],
21     [ 0, 0, 1, 1 ],
22     [ 0, 1, 0, 0 ],
23     [ 0, 1, 0, 1 ],
24     [ 0, 1, 1, 0 ],
25     [ 0, 1, 1, 1 ],
26     [ 1, 0, 0, 0 ],
27     [ 1, 0, 0, 1 ]
28 ])
```

03 : np.set_printoptions 함수를 호출하여 print 함수로 numpy 배열을 출력할 때, 실수 출력 형식을 조절합니다. precision 인자는 소수점 이하 자리수를 설정하며, 여기서는 소수점 이하 1 자리까지 출력합니다. suppress 인자는 실수 표기 방식을 설정하며, True로 설정할 경우 고정 소수점 표기법(Fixed point notation)을 사용하며, False로 설정할 경우 과학적 표기법(Scientific notation)을 사용합니다. 예를 들어, 0.000514는 고정 소수점 표기법이고, 5.14e-04는 과학적 표기법입니다. 여기서는 고정 소수점 표기법을 사용합니다.

05~16 : X 변수를 선언하고, 진리표의 입력 값으로 초기화된 2차 numpy 배열을 할당합니다.

17~28 : YT 변수를 선언하고, 진리표의 목표 값으로 초기화된 2차 numpy 배열을 할당합니다.

2. 계속해서 다음과 같이 예제를 작성합니다.

_461.py

```
01 from _7seg_data import X, YT
02
03 print(X.shape)
04 print(YT.shape)
```

01 : _7seg_data 모듈로부터 X, YT를 불러옵니다.

03 : X의 모양을 출력합니다.

04 : YT의 모양을 출력합니다.

3. 다음과 같이 예제를 실행합니다.

```
$ python _461.py
```

다음은 실행 결과 화면입니다.

```
(10, 7)
(10, 4)
```

X는 10*7 크기의 2차 배열입니다.

YT는 10*4 크기의 2차 배열입니다.

딥러닝 모델 학습시키기

다음은 딥러닝 모델을 생성한 후, 학습을 시켜 봅니다.

1. 다음과 같이 예제를 작성합니다.

_461_2.py

```
01 import tensorflow as tf
02 from _7seg_data import X, YT
03
04 model=tf.keras.Sequential([
05     tf.keras.Input(shape=(7,)),
06     tf.keras.layers.Dense(8, activation='relu'),
07     tf.keras.layers.Dense(4, activation='sigmoid')
08 ])
09
10 model.compile(optimizer='adam', loss='mse')
11
12 model.fit(X,YT,epochs=10000)
13
14 Y=model.predict(X)
15 print(Y)
```

04~08 : 입력층의 노드 수는 7개, 은닉층의 노드 수는 8개, 출력층의 노드 수는 4개인 신경망 모델을 생성합니다. 은닉 층의 활성화 함수는 'relu', 출력 층의 활성화 함수는 'sigmoid'로 설정합니다.

10 : 학습 함수를 'adam', 오차 함수를 'mse'로 설정합니다. 'adam'은 가장 많이 사용하는 학습 함수입니다. 'mse'는 평균 제곱 오차 함수입니다.

12 : 학습을 10000 회 수행합니다.

14 : X에 대해 예측을 수행해 봅니다.

15 : 예측 값 Y를 출력합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ python _461_2.py
```

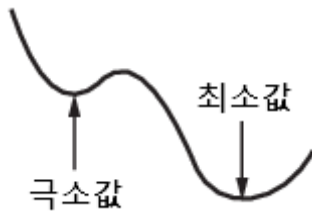
다음은 실행 결과 화면입니다.

```
Epoch 10000/10000
1/1 [=====] - 0s 5ms/step - loss: 1.5465e-05
[[0. 0. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 1. 0.]
 [0. 0. 1. 1.]
 [0. 1. 0. 0.]
 [0. 1. 0. 1.]
 [0. 1. 1. 0.]
 [0. 1. 1. 1.]
 [1. 0. 0. 0.]
 [1. 0. 0. 1.]]
```

YT값과 비교해 봅니다.

국소해의 문제 해결해 보기

앞의 예제에서 은닉층 노드의 개수가 8일 경우 7에 대한 학습이 제대로 되지 않는 경우가 있는데 이런 현상은 국소해의 문제로 발생합니다. 예를 들어, 다음 그림에서 신경망의 학습 과정에서 최소값 지점을 찾지 못하고 극소값 지점에 수렴하는 경우입니다. 국소해의 문제가 발생할 경우엔 재학습을 수행해 보거나 은닉층의 노드수를 변경해 봅니다. 여기서는 은닉층 노드의 개수를 16으로 늘려봅니다.



1. 다음과 같이 예제를 수정합니다.

_461_3.py

```
01 import tensorflow as tf
02 from _7seg_data import X, YT
03
04 model=tf.keras.Sequential([
05     tf.keras.Input(shape=(7,)),
06     tf.keras.layers.Dense(16, activation='relu'),
07     tf.keras.layers.Dense(4, activation='sigmoid')
08 ])
09
10 model.compile(optimizer='adam', loss='mse')
11
12 model.fit(X,YT,epochs=10000)
13
14 Y=model.predict(X)
15 print(Y)
```

06 : 은닉층이 노드 개수를 16으로 늘립니다.

2. 다음과 같이 예제를 실행합니다.

```
$ python _461_3.py
```

다음은 실행 결과 화면입니다.

```
Epoch 10000/10000
1/1 [=====] - 0s 5ms/step - loss: 7.2903e-06
[[0. 0. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 1. 0.]
 [0. 0. 1. 1.]
 [0. 1. 0. 0.]
 [0. 1. 0. 1.]
 [0. 1. 1. 0.]
 [0. 1. 1. 1.]
 [1. 0. 0. 0.]
 [1. 0. 0. 1.]]
```

출력층에 linear 함수 적용해 보기

여기서는 출력층 함수에 linear 함수를 적용해 봅니다. linear 함수는 출력단의 값을 그대로 내보내는 함수입니다. linear 함수를 적용하여 학습을 수행할 경우 선형 회귀라고 합니다.

1. 다음과 같이 예제를 수정합니다.

_461_4.py

```
01 import tensorflow as tf
02 from _7seg_data import X, YT
03
04 model=tf.keras.Sequential([
05     tf.keras.Input(shape=(7,)),
06     tf.keras.layers.Dense(16, activation='relu'),
07     tf.keras.layers.Dense(4, activation='linear')
08 ])
09
10 model.compile(optimizer='adam', loss='mse')
11
12 model.fit(X,YT,epochs=10000)
13
14 Y=model.predict(X)
15 print(Y)
```

07 : 출력층의 활성화 함수를 linear 함수로 변경합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ python _461_4.py
```

다음은 실행 결과 화면입니다.

```
Epoch 10000/10000
1/1 [=====] - 0s 5ms/step - loss: 1.1816e-12
[[ 0. -0.  0.  0.]
 [ 0. -0.  0.  1.]
 [ 0. -0.  1.  0.]
 [ 0. -0.  1.  1.]
 [ 0.  1.  0.  0.]
 [ 0.  1.  0.  1.]
 [ 0.  1.  1.  0.]
 [ 0.  1.  1.  1.]
 [ 1.  0.  0.  0.]
 [ 1. -0.  0.  1.]]
```

loss(오차)가 이전 예제보다 더 줄어드는 것을 확인합니다.

목표값 변경해 보기

여기서는 목표값의 형식을 2진수에서 10진수로 변경해 봅니다.

1. 다음과 같이 _7seg_data.py 라이브러리를 수정합니다.

_7seg_data.py

```
01 import numpy as np
02
03 np.set_printoptions(precision=1, suppress=True)
04
05 X=np.array([
06     [ 1, 1, 1, 1, 1, 1, 0 ], # 0
07     [ 0, 1, 1, 0, 0, 0, 0 ], # 1
08     [ 1, 1, 0, 1, 1, 0, 1 ], # 2
09     [ 1, 1, 1, 1, 0, 0, 1 ], # 3
10     [ 0, 1, 1, 0, 0, 1, 1 ], # 4
11     [ 1, 0, 1, 1, 0, 1, 1 ], # 5
12     [ 0, 0, 1, 1, 1, 1, 1 ], # 6
13     [ 1, 1, 1, 0, 0, 0, 0 ], # 7
14     [ 1, 1, 1, 1, 1, 1, 1 ], # 8
15     [ 1, 1, 1, 0, 0, 1, 1 ]  # 9
16 ])
17 YT=np.array([
18     [ 0, 0, 0, 0 ],
19     [ 0, 0, 0, 1 ],
20     [ 0, 0, 1, 0 ],
21     [ 0, 0, 1, 1 ],
22     [ 0, 1, 0, 0 ],
23     [ 0, 1, 0, 1 ],
24     [ 0, 1, 1, 0 ],
25     [ 0, 1, 1, 1 ],
26     [ 1, 0, 0, 0 ],
27     [ 1, 0, 0, 1 ]
28 ])
29 YT_1=np.array([
30     0,
31     1,
32     2,
33     3,
34     4,
```

```

35     5,
36     6,
37     7,
38     8,
39     9
40 ])

```

29~40 : YT_1 변수를 선언하고, 10진수 형식의 목표값으로 초기화된 1차 numpy 배열을 할당합니다.
을 합니다.

2. 다음과 같이 예제를 수정합니다.
_461_5.py

```

01 import tensorflow as tf
02 from _7seg_data import X, YT_1
03
04 model=tf.keras.Sequential([
05     tf.keras.Input(shape=(7,)),
06     tf.keras.layers.Dense(16, activation='relu'),
07     tf.keras.layers.Dense(1, activation='linear')
08 ])
09
10 model.compile(optimizer='adam', loss='mse')
11
12 model.fit(X,YT_1,epochs=10000)
13
14 Y=model.predict(X)
15 print(Y)

```

02 : _7seg_data 모듈로부터 X, YT_1을 불러옵니다.
07 : 출력층 노드의 개수를 1개로 줄입니다.

3. 다음과 같이 예제를 실행합니다.

```
$ python _461_5.py
```

다음은 실행 결과 화면입니다.

```

Epoch 10000/10000
1/1 [=====] - 0s 5ms/step - loss: 4.5082e-12
[[0.]
 [1.]
 [2.]
 [3.]
 [4.]
 [5.]
 [6.]
 [7.]
 [8.]
 [9.]]

```

결과가 잘 나오는 것을 확인합니다.

입력층과 목표층 바꿔보기

다음은 이전 예제의 입력층과 목표층을 바꿔 인공 신경망을 학습 시켜봅니다. 다음과 같이 2진수가 입력 되면 해당되는 7 세그먼트의 켜지고 꺼져야 할 LED의 비트열을 출력합니다.

2 진수 7 세그먼트 연결 진리표

In	In	In	In	Out	Out	Out	Out	Out	Out	Out
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	0	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1

0101 ➡ 1011011 = 5

예를 들어, “숫자 5에 맞게 7 세그먼트 LED를 켜줘!” 하고 싶을 때, 사용할 수 있는 인공 신경망입니다.

1. 다음과 같이 예제를 수정합니다.

_461_6.py

```
01 import tensorflow as tf
02 from _7seg_data import X, YT
03
04 X, YT = YT, X
05
06 model=tf.keras.Sequential([
07     tf.keras.Input(shape=(4,)),
08     tf.keras.layers.Dense(16, activation='relu'),
09     tf.keras.layers.Dense(7, activation='linear')
10 ])
11
12 model.compile(optimizer='adam', loss='mse')
13
14 model.fit(X,YT,epochs=10000)
15
16 Y=model.predict(X)
17 print(Y)
```

01 : _7seg_data 모듈로부터 X, YT을 불러옵니다.

04 : YT, X를 X, YT로 변경합니다. 즉, 입력과 출력을 바꿔줍니다.

07 : 입력층 노드의 개수를 4로 바꿉니다.

09 : 출력층 노드의 개수를 7로 바꿉니다.

2. 다음과 같이 예제를 실행합니다.

```
$ python _461_6.py
```

다음은 실행 결과 화면입니다.

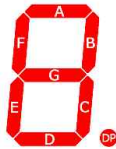
```
Epoch 10000/10000
1/1 [=====] - 0s 5ms/step - loss: 9.1047e-12
[[ 1.  1.  1.  1.  1.  1.  0.]
 [ 0.  1.  1.  0.  0.  0.  0.]
 [ 1.  1. -0.  1.  1.  0.  1.]
 [ 1.  1.  1.  1.  0.  0.  1.]
 [ 0.  1.  1.  0.  0.  1.  1.]
 [ 1. -0.  1.  1.  0.  1.  1.]
 [ 0. -0.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  0.  0.  0.  0.]
 [ 1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  0.  0.  1.  1.]]
```

02 은닉층 늘려보기

여기서는 은닉층을 늘려 봅니다. 일반적으로 은닉층의 개수가 2개 이상일 때 심층 인공 신경망이라고 합니다. 데이터는 다음과 같이 원래 데이터를 사용합니다.

7 세그먼트 2 진수 연결 진리표

	A	B	C	D	E	F	G				
	In	In	In	In	In	In	In	Out	Out	Out	Out
0	1	1	1	1	1	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	1
2	1	1	0	1	1	0	1	0	0	1	0
3	1	1	1	1	0	0	1	0	0	1	1
4	0	1	1	0	0	1	1	0	1	0	0
5	1	0	1	1	0	1	1	0	1	0	1
6	0	0	1	1	1	1	1	0	1	1	0
7	1	1	1	0	0	0	0	0	1	1	1
8	1	1	1	1	1	1	1	1	0	0	0
9	1	1	1	0	0	1	1	1	0	0	1



5 = 1011011 → 0101

1. 다음과 같이 예제를 작성합니다.

_462.py

```

01 import tensorflow as tf
02 from _7seg_data import X, YT
03
04 model=tf.keras.Sequential([
05     tf.keras.Input(shape=(7,)),
06     tf.keras.layers.Dense(16, activation='relu'),
07     tf.keras.layers.Dense(16, activation='relu'),
08     tf.keras.layers.Dense(4, activation='sigmoid')
09 ])
10
11 model.compile(optimizer='adam', loss='mse')
12
13 model.fit(X,YT,epochs=10000)
14
15 Y=model.predict(X)
16 print(Y)

```

07 : 은닉층을 하나 더 늘립니다. 노드의 개수는 16으로 합니다.

08 : 출력층의 활성화 함수가 sigmoid인 것을 확인합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ python _462.py
```

다음은 실행 결과 화면입니다.

```
Epoch 10000/10000
1/1 [=====] - 0s 6ms/step - loss: 6.0022e-07
[[0. 0. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 1. 0.]
 [0. 0. 1. 1.]
 [0. 1. 0. 0.]
 [0. 1. 0. 1.]
 [0. 1. 1. 0.]
 [0. 1. 1. 1.]
 [1. 0. 0. 0.]
 [1. 0. 0. 1.]]
```

학습 시키고 모델 내보내기

다음은 인공 신경망을 준비하여 학습을 수행한 후, 수행 결과를 저장합니다.

1. 다음과 같이 예제를 수정합니다.

_462_2.py

```
01 import tensorflow as tf
02 from _7seg_data import X, YT
03
04 model=tf.keras.Sequential([
05     tf.keras.Input(shape=(7,)),
06     tf.keras.layers.Dense(16, activation='relu'),
07     tf.keras.layers.Dense(16, activation='relu'),
08     tf.keras.layers.Dense(4, activation='sigmoid')
09 ])
10
11 model.compile(optimizer='adam', loss='mse')
12
13 model.fit(X,YT,epochs=10000)
14
15 Y=model.predict(X)
16 print(Y)
17
18 model.save('model.h5')
```

18 : save 함수를 이용하여 학습된 인공 신경망을 'model.h5' 파일로 저장해 줍니다.

2. 다음과 같이 예제를 실행합니다.

```
$ python _462_2.py
```

3. 학습이 끝난 후, 다음과 같이 model.h5 파일이 생성된 것을 확인합니다.

```
linaro@linaro-alip:~/pyLabs$ ls -l model.h5
-rw-r--r-- 1 linaro linaro 36640 Feb  1 00:57 model.h5
```

모델 불러와 예측하기 1

다음은 학습된 인공 신경망을 불러와 예측을 수행해 봅니다.

1. 다음과 같이 예제를 작성합니다.

_462_3.py

```
1 import tensorflow as tf
2 from _7seg_data import X, YT
3
4 model = tf.keras.models.load_model("model.h5")
5
6 Y=model.predict(X)
7 print(Y)
```

4 : tf.keras.models.load_model 함수를 호출하여 학습된 모델을 불러옵니다.

6 : 예측을 수행합니다.

7 : 예측 결과를 출력합니다.

2. 다음과 같이 예제를 실행합니다.

```
$ python _462_3.py
```

다음은 실행 결과 화면입니다.

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


모델 불러와 예측하기 2

여기서는 1개 데이터에 대한 예측을 해 봅니다.

다음과 같이 예제를 수정합니다.

_462_4.py

```
01 import tensorflow as tf
02 from _7seg_data import X
03
04 model = tf.keras.models.load_model("model.h5")
05
06 x=X[:1]
07 print(x.shape)
08
09 Y=model.predict(x)
10 print(Y)
```

06 : x 변수를 생성한 후, X[:1]로 초기화합니다. 이렇게 하면 X의 0번 항목의 numpy 배열이 할당됩니다.

07 : x의 모양을 출력해 봅니다.

2. 다음과 같이 예제를 실행합니다.

```
$ python _462_4.py
```

다음은 실행 결과 화면입니다.

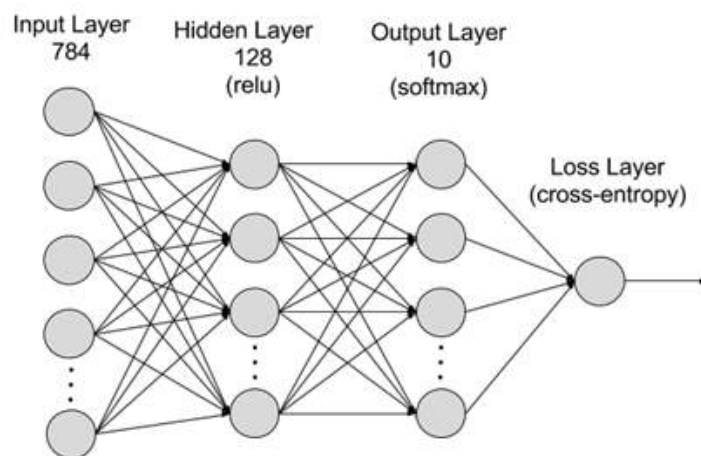
```
(1, 7)
```

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

03 딥러닝 활용 예제 살펴보기



여기서는 MNIST라고 하는 손글씨 숫자 데이터를 입력받아 학습을 수행하는 예제를 살펴봅니다. 다음과 같은 모양의 인공 신경망을 구성하고 학습시켜 봅니다.



1. 다음과 같이 예제를 작성합니다.

_463.py

```
01 import tensorflow as tf
02
03 mnist = tf.keras.datasets.mnist
04
05 (X, Y), (x, y) = 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,)),
```

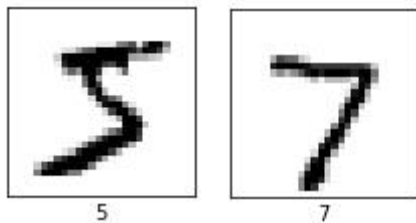
```

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)

```

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.

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 가리킵니다.



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.

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.

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. 다음과 같이 예제를 실행합니다.

```
$ python _463.py
```

다음은 실행 결과 화면입니다.

```
Epoch 1/5  
1875/1875 [=====] - 29s 15ms/step - loss: 0.4259 - accuracy: 0.8800  
Epoch 2/5  
1875/1875 [=====] - 27s 14ms/step - loss: 0.1214 - accuracy: 0.9645  
Epoch 3/5  
1875/1875 [=====] - 27s 14ms/step - loss: 0.0787 - accuracy: 0.9756  
Epoch 4/5  
1875/1875 [=====] - 27s 14ms/step - loss: 0.0574 - accuracy: 0.9834  
Epoch 5/5  
1875/1875 [=====] - 27s 14ms/step - loss: 0.0420 - accuracy: 0.9875  
313/313 [=====] - 3s 9ms/step - loss: 0.0763 - accuracy: 0.9761
```

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

❷ 학습 진행에 따른 정확도가 표시됩니다. 처음에 88.00%에서 시작해서 마지막엔 98.75%의 정확도로 학습이 끝납니다. 즉, 100개의 손글씨가 있다면 98.75개를 맞춘다는 의미입니다.

❸ 학습이 끝난 후에, evaluate 함수로 시험 데이터를 평가한 결과입니다. 손실값이 늘어났고, 정확도가 97.61%로 약간 떨어진 상태입니다.