

## 01 7 세그먼트 입력 2 진수 출력 인공 신경망

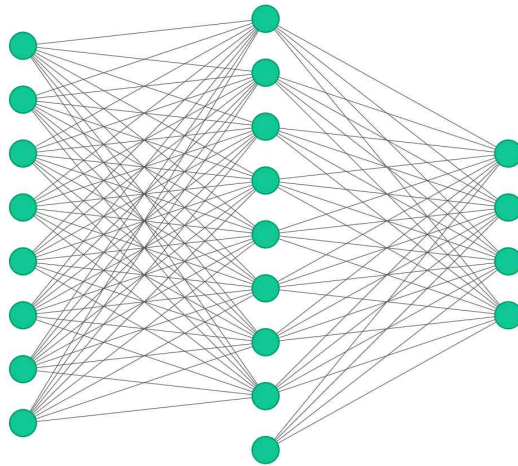
여기서는 7 세그먼트에 숫자 값에 따라 표시되는 LED의 ON, OFF 값을 입력으로 받아 2 진수로 출력하는 인공 신경망을 구성하고 학습시켜 봅니다. 다음은 7 세그먼트 디스플레이 2 진수 연결 진리표입니다.

7 세그먼트 디스플레이  
2 진수 연결 진리표

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

1. 다음과 같이 예제를 작성합니다.

443.py

```
1 import tensorflow as tf
2 import numpy as np
3
4 X=np.array([
5     [ 1, 1, 1, 1, 1, 1, 0 ], # 0
6     [ 0, 1, 1, 0, 0, 0, 0 ], # 1
7     [ 1, 1, 0, 1, 1, 0, 1 ], # 2
8     [ 1, 1, 1, 1, 0, 0, 1 ], # 3
9     [ 0, 1, 1, 0, 0, 1, 1 ], # 4
10    [ 1, 0, 1, 1, 0, 1, 1 ], # 5
11    [ 0, 0, 1, 1, 1, 1, 1 ], # 6
12    [ 1, 1, 1, 0, 0, 0, 0 ], # 7
13    [ 1, 1, 1, 1, 1, 1, 1 ], # 8
14    [ 1, 1, 1, 0, 0, 1, 1 ] # 9
15 ])
16 YT=np.array([
17     [ 0, 0, 0, 0 ],
18     [ 0, 0, 0, 1 ],
19     [ 0, 0, 1, 0 ],
20     [ 0, 0, 1, 1 ],
21     [ 0, 1, 0, 0 ],
22     [ 0, 1, 0, 1 ],
23     [ 0, 1, 1, 0 ],
24     [ 0, 1, 1, 1 ],
25     [ 1, 0, 0, 0 ],
26     [ 1, 0, 0, 1 ]
27 ])
28
29 print(X.shape)
30 print(YT.shape)
31
32 model=tf.keras.Sequential([
33     tf.keras.Input(shape=(7,)),
34     tf.keras.layers.Dense(8, activation='relu'),
35     tf.keras.layers.Dense(4, activation='sigmoid')
36 ])
37
38 model.compile(optimizer='adam', loss='mse')
39
```

```

40 model.fit(X,YT,epochs=10000)
41
42 Y=model.predict(X)
43 print(Y)

```

국소해에 빠지는 경우가 있다.

```

32 model=tf.keras.Sequential([
33     tf.keras.Input(shape=(7,)),
34     tf.keras.layers.Dense(16, activation='relu'),
35     tf.keras.layers.Dense(4, activation='sigmoid')
36 ])

```

Epoch 10000/10000

1/1 [=====] - 0s 2ms/step - loss: 8.0053e-06

1/1 [=====] - 0s 66ms/step

```

[[3.6981618e-03 1.2449448e-03 7.2968280e-04 2.9978044e-03]
 [3.5862092e-07 4.7370908e-03 2.5717681e-03 9.9719149e-01]
 [3.5441283e-03 2.6570888e-07 9.9877763e-01 1.6049853e-03]
 [3.7207388e-05 3.6741528e-03 9.9804980e-01 9.9995095e-01]
 [3.0113156e-03 9.9502647e-01 2.1775323e-03 3.5358509e-03]
 [3.3737638e-03 9.9800378e-01 3.2143770e-03 9.9660629e-01]
 [1.0519896e-03 9.9868309e-01 9.9633408e-01 1.0091607e-06]
 [3.8372610e-07 9.9437213e-01 9.9692625e-01 9.9999881e-01]
 [9.9469268e-01 9.3102808e-06 3.2097907e-03 1.4583615e-04]
 [9.9667680e-01 4.8381379e-03 3.0237666e-04 9.9761075e-01]]

```

```

32 model=tf.keras.Sequential([
33     tf.keras.Input(shape=(7,)),
34     tf.keras.layers.Dense(16, activation='relu'),
35     tf.keras.layers.Dense(4, activation='linear')
36 ])

```

Epoch 10000/10000

1/1 [=====] - 0s 3ms/step - loss: 8.6637e-13

1/1 [=====] - 0s 61ms/step

```

[[-1.9818544e-06 1.2069941e-06 -6.2584877e-07 -1.9371510e-07]
 [-1.1473894e-06 4.9173832e-07 -1.0430813e-07 9.9999982e-01]
 [-2.2202730e-06 1.2069941e-06 9.9999928e-01 -1.3411045e-07]
 [-1.9818544e-06 1.0281801e-06 9.9999905e-01 9.9999946e-01]
 [-4.3213367e-07 1.0000002e+00 -2.0861626e-07 -1.4901161e-07]
 [-1.7434359e-06 1.0000011e+00 -5.3644180e-07 9.9999976e-01]
 [-1.6242266e-06 1.0000008e+00 9.9999970e-01 0.0000000e+00]
 [-1.1473894e-06 1.0000007e+00 9.9999970e-01 9.9999934e-01]
 [ 9.9999827e-01 1.2069941e-06 -7.3015690e-07 2.9802322e-08]
 [ 9.9999923e-01 2.2351742e-07 -2.6822090e-07 9.9999958e-01]]

```

2. 다음과 같이 예제를 수정합니다.

```
01 import tensorflow as tf
02 import numpy as np
03
04 X=np.array([
05     [ 1, 1, 1, 1, 1, 1, 0 ], # 0
06     [ 0, 1, 1, 0, 0, 0, 0 ], # 1
07     [ 1, 1, 0, 1, 1, 0, 1 ], # 2
08     [ 1, 1, 1, 1, 0, 0, 1 ], # 3
09     [ 0, 1, 1, 0, 0, 1, 1 ], # 4
10     [ 1, 0, 1, 1, 0, 1, 1 ], # 5
11     [ 0, 0, 1, 1, 1, 1, 1 ], # 6
12     [ 1, 1, 1, 0, 0, 0, 0 ], # 7
13     [ 1, 1, 1, 1, 1, 1, 1 ], # 8
14     [ 1, 1, 1, 0, 0, 1, 1 ]  # 9
15 ])
16 YT=np.array([
17     0,
18     1,
19     2,
20     3,
21     4,
22     5,
23     6,
24     7,
25     8,
26     9
27 ])
28
29 print(X.shape)
30 print(YT.shape)
31
32 model=tf.keras.Sequential([
33     tf.keras.Input(shape=(7,)),
34     tf.keras.layers.Dense(16, activation='relu'),
35     tf.keras.layers.Dense(1, activation='linear')
36 ])
37
38 model.compile(optimizer='adam', loss='mse')
39
```

```
40 model.fit(X,YT,epochs=10000)
41
42 Y=model.predict(X)
43 print(Y)
```

Epoch 10000/10000

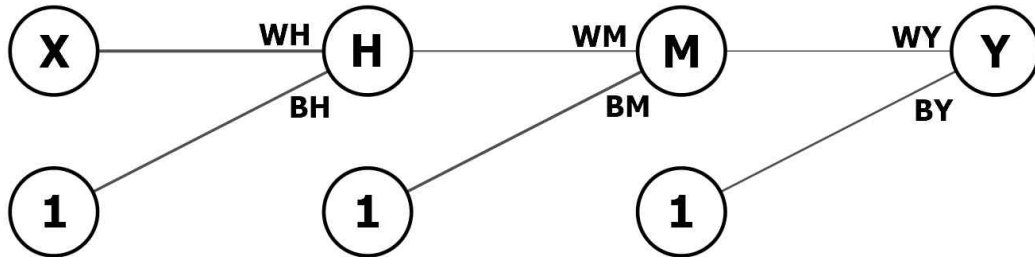
1/1 [=====] - 0s 2ms/step - loss: 1.1666e-13

1/1 [=====] - 0s 64ms/step

```
[ [1.8253922e-07]
  [9.9999988e-01]
  [2.0000000e+00]
  [3.0000000e+00]
  [4.0000000e+00]
  [5.0000005e+00]
  [6.0000005e+00]
  [6.9999995e+00]
  [8.0000010e+00]
  [9.0000000e+00]]
```

## 05 은닉층 추가하기

여기서는 은닉층을 하나 더 추가해 봅니다. 일반적으로 은닉층의 개수가 2개 이상일 때 심층 인공 신경망이라고 합니다. 다음은 은닉층 M이 추가된 X-H-M-Y 심층 인공 신경망입니다.



1. 다음과 같이 이전 예제를 수정합니다.

445.py

```
1 import tensorflow as tf
2 import numpy as np
3
4 X=np.array([
5     [ 1, 1, 1, 1, 1, 1, 0 ], # 0
6     [ 0, 1, 1, 0, 0, 0, 0 ], # 1
7     [ 1, 1, 0, 1, 1, 0, 1 ], # 2
8     [ 1, 1, 1, 1, 0, 0, 1 ], # 3
9     [ 0, 1, 1, 0, 0, 1, 1 ], # 4
10    [ 1, 0, 1, 1, 0, 1, 1 ], # 5
11    [ 0, 0, 1, 1, 1, 1, 1 ], # 6
12    [ 1, 1, 1, 0, 0, 0, 0 ], # 7
13    [ 1, 1, 1, 1, 1, 1, 1 ], # 8
14    [ 1, 1, 1, 0, 0, 1, 1 ] # 9
15 ])
16 YT=np.array([
17     [ 0, 0, 0, 0 ],
18     [ 0, 0, 0, 1 ],
19     [ 0, 0, 1, 0 ],
20     [ 0, 0, 1, 1 ],
21     [ 0, 1, 0, 0 ],
22     [ 0, 1, 0, 1 ],
23     [ 0, 1, 1, 0 ],
24     [ 0, 1, 1, 1 ],
25     [ 1, 0, 0, 0 ],
```

```

26     [ 1, 0, 0, 1 ]
27 ])
28
29 print(X.shape)
30 print(YT.shape)
31
32 model=tf.keras.Sequential([
33     tf.keras.Input(shape=(7,)),
34     tf.keras.layers.Dense(16, activation='relu'),
35     tf.keras.layers.Dense(16, activation='relu'),
36     tf.keras.layers.Dense(4, activation='sigmoid')
37 ])
38
39 model.compile(optimizer='adam', loss='mse')
40
41 model.fit(X,YT,epochs=10000)
42
43 Y=model.predict(X)
44 print(Y)

```

```

Epoch 10000/10000
1/1 [=====] - 0s 2ms/step - loss: 1.5937e-06
1/1 [=====] - 0s 65ms/step
[[2.1450971e-03 2.6790133e-08 1.5646226e-03 2.0306043e-03]
 [1.7417160e-04 1.4992780e-03 1.4501681e-03 9.9823612e-01]
 [1.4564299e-04 3.8473964e-08 9.9927604e-01 1.4604448e-04]
 [8.7747352e-08 1.8280195e-03 9.9805599e-01 9.9946278e-01]
 [1.0885850e-03 9.9859947e-01 1.4493716e-03 1.7422306e-03]
 [7.2394044e-04 9.9832177e-01 9.6621981e-04 9.9852198e-01]
 [1.3732681e-06 9.9998128e-01 9.9779463e-01 6.1089480e-09]
 [6.0231827e-08 9.9813199e-01 9.9901772e-01 9.9999899e-01]
 [9.9803913e-01 8.1328420e-07 2.5301281e-04 6.4430591e-05]
 [9.9821341e-01 1.9305401e-03 1.8042119e-08 9.9852437e-01]]

```

### 소수점 조절하기

```
np.set_printoptions(formatter={'float_kind':lambda x: "{0:0.4f}".format(x)})
```

```
np.set_printoptions(formatter={'float_kind':lambda x: "{0:0.0f}".format(x)})
```

```
np.set_printoptions(precision=4)
```


```
np.set_printoptions(precision=0)
```

## 06 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 = 

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

1. 다음과 같이 이전 예제를 수정합니다.

446.py

```
01 import tensorflow as tf
02 import numpy as np
03
04 X=np.array([
05     [ 0, 0, 0, 0 ],
06     [ 0, 0, 0, 1 ],
07     [ 0, 0, 1, 0 ],
08     [ 0, 0, 1, 1 ],
09     [ 0, 1, 0, 0 ],
10     [ 0, 1, 0, 1 ],
11     [ 0, 1, 1, 0 ],
12     [ 0, 1, 1, 1 ],
13     [ 1, 0, 0, 0 ],
14     [ 1, 0, 0, 1 ]
15 ])
16 YT=np.array([
```



```

17     [ 1, 1, 1, 1, 1, 1, 0 ], # 0
18     [ 0, 1, 1, 0, 0, 0, 0 ], # 1
19     [ 1, 1, 0, 1, 1, 0, 1 ], # 2
20     [ 1, 1, 1, 1, 0, 0, 1 ], # 3
21     [ 0, 1, 1, 0, 0, 1, 1 ], # 4
22     [ 1, 0, 1, 1, 0, 1, 1 ], # 5
23     [ 0, 0, 1, 1, 1, 1, 1 ], # 6
24     [ 1, 1, 1, 0, 0, 0, 0 ], # 7
25     [ 1, 1, 1, 1, 1, 1, 1 ], # 8
26     [ 1, 1, 1, 0, 0, 1, 1 ]  # 9
27 ])
28
29 print(X.shape)
30 print(YT.shape)
31
32 model=tf.keras.Sequential([
33     tf.keras.Input(shape=(4,)),
34     tf.keras.layers.Dense(16, activation='relu'),
35     tf.keras.layers.Dense(16, activation='relu'),
36     tf.keras.layers.Dense(7, activation='sigmoid')
37 ])
38
39 model.compile(optimizer='adam', loss='mse')
40
41 model.fit(X,YT,epochs=10000)
42
43 Y=model.predict(X)
44 print(Y)

```

```

Epoch 10000/10000
1/1 [=====] - 0s 3ms/step - loss: 1.6006e-06
1/1 [=====] - 0s 84ms/step
[[[9.9837542e-01 1.0000000e+00 9.9942923e-01 9.9838495e-01 9.9867290e-01
  9.9805862e-01 2.0758377e-03]
 [2.8269070e-03 9.9999988e-01 9.9999803e-01 5.7722747e-09 5.2259210e-04
  1.9353762e-03 7.6457864e-06]
 [1.0000000e+00 9.9879044e-01 9.2368649e-04 1.0000000e+00 1.0000000e+00
  1.6841360e-03 9.9997163e-01]
 [1.0000000e+00 9.9949187e-01 9.9898601e-01 9.9999142e-01 1.4232044e-03
  3.0082923e-05 9.9773967e-01]
 [1.1908805e-03 9.9775684e-01 1.0000000e+00 1.2415396e-03 2.2758653e-03
  9.9999845e-01 9.9822211e-01]
 [9.9873447e-01 2.5232721e-03 9.9974149e-01 9.9749506e-01 4.7901113e-04
  9.9961621e-01 9.9999905e-01]
 [1.0348437e-03 1.9403934e-03 9.9913245e-01 9.9999565e-01 9.9831843e-01
  9.9849856e-01 1.0000000e+00]
 [9.9735361e-01 9.9994606e-01 9.9989724e-01 1.8674997e-04 6.6310778e-05
  2.6371385e-04 3.3872519e-03]
 [9.9999976e-01 9.9999136e-01 9.998903e-01 1.0000000e+00 9.9917656e-01
  1.0000000e+00 9.9998820e-01]
 [9.9845940e-01 9.9902737e-01 1.0000000e+00 1.8195179e-03 9.3764601e-07
  9.9903655e-01 9.9843305e-01]]]

```

예측 값이 목표 값에 적당히 가까운 것을 볼 수 있습니다.

## 학습 시키고 모델 내보내기

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

1. 다음과 같이 예제를 수정합니다.

```
01 import tensorflow as tf
02 import numpy as np
03
04 np.set_printoptions(formatter={'float_kind':lambda x: "{0:6.4f}".format(x)})
05
06 X=np.array([
07     [ 1, 1, 1, 1, 1, 1, 0 ], # 0
08     [ 0, 1, 1, 0, 0, 0, 0 ], # 1
09     [ 1, 1, 0, 1, 1, 0, 1 ], # 2
10     [ 1, 1, 1, 1, 0, 0, 1 ], # 3
11     [ 0, 1, 1, 0, 0, 1, 1 ], # 4
12     [ 1, 0, 1, 1, 0, 1, 1 ], # 5
13     [ 0, 0, 1, 1, 1, 1, 1 ], # 6
14     [ 1, 1, 1, 0, 0, 0, 0 ], # 7
15     [ 1, 1, 1, 1, 1, 1, 1 ], # 8
16     [ 1, 1, 1, 0, 0, 1, 1 ] # 9
17 ])
18 YT=np.array([
19     [ 0, 0, 0, 0 ],
20     [ 0, 0, 0, 1 ],
21     [ 0, 0, 1, 0 ],
22     [ 0, 0, 1, 1 ],
23     [ 0, 1, 0, 0 ],
24     [ 0, 1, 0, 1 ],
25     [ 0, 1, 1, 0 ],
26     [ 0, 1, 1, 1 ],
27     [ 1, 0, 0, 0 ],
28     [ 1, 0, 0, 1 ]
29 ])
30
31 model=tf.keras.Sequential([
32     tf.keras.Input(shape=(7,)),
33     tf.keras.layers.Dense(16, activation='relu'),
34     tf.keras.layers.Dense(16, activation='relu'),
35     tf.keras.layers.Dense(4, activation='sigmoid')
36 ])
37
38 model.compile(optimizer='adam', loss='mse')
39
```

```
40 model.fit(X,YT,epochs=10000)
41
42 Y=model.predict(X)
43 print(Y)
44
45 model.save('model.h5')
```

```
[[0.0011 0.0000 0.0010 0.0010]
 [0.0004 0.0015 0.0004 0.9988]
 [0.0008 0.0000 0.9999 0.0006]
 [0.0001 0.0009 0.9991 0.9997]
 [0.0007 0.9986 0.0005 0.0011]
 [0.0005 0.9993 0.0010 0.9994]
 [0.0000 1.0000 0.9988 0.0000]
 [0.0000 0.9987 0.9996 1.0000]
 [0.9986 0.0000 0.0002 0.0001]
 [0.9990 0.0009 0.0000 0.9994]]
```

## 모델 불러오기 : load\_model

```
01 from tensorflow import keras
02 import numpy as np
03
04 model = keras.models.load_model("model.h5")
05
06 W1, b1 = model.layers[0].get_weights()
07 W2, b2 = model.layers[1].get_weights()
08 W3, b3 = model.layers[2].get_weights()
09
10 print(W1.shape, b1.shape)
11 print(W2.shape, b2.shape)
12 print(W3.shape, b3.shape)
```

## NumPy로 예측해 보기

다음은 이전 예제에서 학습한 결과를 읽어와 학습된 가중치를 얻어온 후, NumPy를 이용하여 예측을 수행해 봅니다.

1. 다음과 같이 예제를 작성합니다.

```
01 from tensorflow import keras
02 import numpy as np
03
04 model = keras.models.load_model("model.h5")
05
06 W1, b1 = model.layers[0].get_weights()
07 W2, b2 = model.layers[1].get_weights()
08 W3, b3 = model.layers[2].get_weights()
09
10 print(W1.shape, b1.shape)
11 print(W2.shape, b2.shape)
12 print(W3.shape, b3.shape)
13
14 np.set_printoptions(formatter={'float_kind':lambda x: "{0:6.0f}".format(x)})
15
16 X=np.array([
17     [ 1, 1, 1, 1, 1, 1, 0 ], # 0
18     [ 0, 1, 1, 0, 0, 0, 0 ], # 1
19     [ 1, 1, 0, 1, 1, 0, 1 ], # 2
20     [ 1, 1, 1, 1, 0, 0, 1 ], # 3
21     [ 0, 1, 1, 0, 0, 1, 1 ], # 4
22     [ 1, 0, 1, 1, 0, 1, 1 ], # 5
23     [ 0, 0, 1, 1, 1, 1, 1 ], # 6
24     [ 1, 1, 1, 0, 0, 0, 0 ], # 7
25     [ 1, 1, 1, 1, 1, 1, 1 ], # 8
26     [ 1, 1, 1, 0, 0, 1, 1 ] # 9
27 ])
28
29 xs = X
30 ys = []
31 for x in xs:
32     x = np.array([x]) # x should be array
33     h1 = x @ W1 + b1 # dense layer
34     h1 = np.maximum(0, h1) # ReLU
```

```
35     h2 = h1 @ W2 + b2      # dense layer
36     h2 = np.maximum(0, h2) # ReLU
37     h3 = h2 @ W3 + b3      # dense layer
38     h3 = 1/(1+np.exp(-h3))
39     ys.append(h3)
40
41 for y in ys:
42     print(y)
```

```
[[ 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 내보내기 : Numpy Array

여기서는 pico에서 사용할 수 있도록 Numpy Array로 모델을 내 보냅니다.

다음과 같이 예제를 작성합니다.

```
01 from tensorflow import keras
02 import numpy as np
03
04 model = keras.models.load_model("model.h5")
05
06 W1, b1 = model.layers[0].get_weights()
07 W2, b2 = model.layers[1].get_weights()
08 W3, b3 = model.layers[2].get_weights()
09
10 names = ["W1", "b1", "W2", "b2", "W3", "b3"]
11 arrays = [W1, b1, W2, b2, W3, b3]
12
13 # for Python Pico
14 print('import numpy as np')
15 print()
16 for name, array in zip(names, arrays):
17     print('%s = np.array(' %name)
18     print(np.array2string(array, separator=', '))
19     print('\n')
```

18 : numpy.array2string 함수는 numpy 배열을 쉼표(.)와 같이 출력할 때 사용합니다.

다음은 실행 결과입니다.

```
import numpy as np

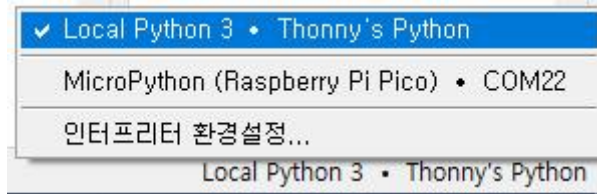
W1 = np.array(
[[ 7.44309425e-02,  1.30884099e+00,  7.66401470e-01,  1.29332745e+00,
   2.56680906e-01, -1.51242584e-01, -5.08458614e-01, -1.07547522e+00,
  -7.73565590e-01, -3.59571218e-01, -9.20085311e-01, -4.87854987e-01,
  -8.85957301e-01,  4.67360020e-04, -9.73209500e-01,  2.58826733e-01],
 [-1.00835532e-01,  9.80220199e-01, -2.88857102e-01,  1.15689278e+00,
...

b3 = np.array(
[-0.16350213, -0.1538213 ,  0.06870118,  0.16322489]
)
```

출력된 내용을 복사하여 model\_data.py 파일로 저장합니다.

## Numpy Array 모델 테스트하기 1

여기서는 Numpy array로 저장된 모델을 PC에서 테스트해 봅니다.



다음과 같이 예제를 작성합니다.

```
01 from model_data import *
02
03 np.set_printoptions(formatter={'float_kind':lambda x: "{0:6.0f}".format(x)})
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
18 xs = X
19 ys = []
20 for x in xs:
21     x = np.array([x]) # x should be array
22     h1 = x @ W1 + b1 # dense layer
23     h1 = np.maximum(0, h1) # ReLU
24     h2 = h1 @ W2 + b2 # dense layer
25     h2 = np.maximum(0, h2) # ReLU
26     y = h2 @ W3 + b3 # dense layer
27     y = 1/(1+np.exp(-y))
28     ys.append(y)
29
30 for y in ys:
31     print(y)
```

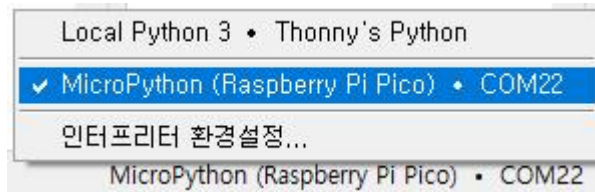


다음은 실행 결과입니다.

```
[[ 0 0 0 0]] [[0.00095002 0.00052321 0.00067394 0.00062433]]
[[ 0 0 0 1]] [[2.49748907e-05 1.00767953e-03 9.13522740e-04 9.99341760e-01]]
[[ 0 0 1 0]] [[3.40098552e-04 3.35165059e-05 9.99976387e-01 1.29097570e-04]]
[[ 0 0 1 1]] [[6.12110404e-04 8.81467395e-04 9.99157189e-01 9.99999071e-01]]
[[ 0 1 0 0]] [[9.46219187e-04 9.98899235e-01 3.75308298e-05 1.08108015e-03]]
[[ 0 1 0 1]] [[2.72192589e-04 9.98810627e-01 9.31879442e-04 9.99187217e-01]]
[[ 0 1 1 1]] [[1.81365955e-07 9.99933744e-01 9.99055561e-01 2.45656387e-07]]
[[ 0 1 1 1]] [[7.67109992e-05 9.98856577e-01 9.99268093e-01 9.99664700e-01]]
[[ 1 0 0 0]] [[9.98807473e-01 1.14704546e-04 5.56537411e-04 4.45173992e-04]]
[[ 1 0 0 0]] [[9.99005777e-01 1.33809514e-03 4.82817202e-04 9.99171303e-01]]
```

## Numpy Array 모델 테스트하기 2

여기서는 Numpy Array 모델을 피코에서 테스트해 봅니다.



1. model\_data.py 파일을 피코의 /에 저장합니다.

2. 다음 부분을 수정합니다.

```
1 from ulab import numpy as np
```

[import numpy as np]를

[from ulab import numpy as np]으로 변경합니다.

3. 다음과 같이 예제를 작성합니다. 이전 예제를 복사한 후 수정합니다. 다음 예제는 피코에서 테스트합니다.

```
01 from model_data import *
02
03 X=np.array([
04     [ 1, 1, 1, 1, 1, 1, 0 ], # 0
05     [ 0, 1, 1, 0, 0, 0, 0 ], # 1
06     [ 1, 1, 0, 1, 1, 0, 1 ], # 2
07     [ 1, 1, 1, 1, 0, 0, 1 ], # 3
08     [ 0, 1, 1, 0, 0, 1, 1 ], # 4
09     [ 1, 0, 1, 1, 0, 1, 1 ], # 5
10     [ 0, 0, 1, 1, 1, 1, 1 ], # 6
11     [ 1, 1, 1, 0, 0, 0, 0 ], # 7
12     [ 1, 1, 1, 1, 1, 1, 1 ], # 8
13     [ 1, 1, 1, 0, 0, 1, 1 ]  # 9
14 ])
15
16 xs = X
17 ys = []
18 for x in xs:
19     x = np.array([x])          # x should be array
20     h1 = np.dot(x,W1) + b1    # dense layer
21     h1 = np.maximum(0, h1)    # ReLU
```

```

22 h2 = np.dot(h1,W2) + b2 # dense layer
23 h2 = np.maximum(0, h2) # ReLU
24 y = np.dot(h2,W3) + b3 # dense layer
25 y = 1/(1+np.exp(-y))
26 ys.append(y)
27
28 for y in ys:
29     print(y)

```

20,22,24 : @ 연산자를 np.dot으로 변경합니다.

결과를 확인합니다.

```

array([[0.0009500202, 0.0005232104, 0.0006739407, 0.0006243248]], dtype=float32)
array([[2.497488e-05, 0.001007679, 0.0009135229, 0.9993418]], dtype=float32)
array([[0.0003400986, 3.351653e-05, 0.9999763, 0.0001290976]], dtype=float32)
array([[0.0006121096, 0.0008814653, 0.9991571, 0.999999]], dtype=float32)
array([[0.0009462198, 0.9988993, 3.753078e-05, 0.001081079]], dtype=float32)
array([[0.0002721922, 0.9988107, 0.0009318792, 0.9991872]], dtype=float32)
array([[1.813658e-07, 0.9999337, 0.9990556, 2.456567e-07]], dtype=float32)
array([[7.671102e-05, 0.9988565, 0.9992682, 0.9996647]], dtype=float32)
array([[0.9988074, 0.0001147048, 0.0005565377, 0.0004451745]], dtype=float32)
array([[0.9990059, 0.001338095, 0.0004828163, 0.9991714]], dtype=float32)

```

피코 파이썬에서는 소수점 이하 자리 조절 함수를 지원하지 않습니다.

## predict 함수 정의하기

여기서는 predict 함수를 정의한 후, 테스트해 봅니다.

다음과 같이 예제를 수정합니다.

```
01 from model_data import *
02
03 X=np.array([
04     [ 1, 1, 1, 1, 1, 1, 0 ], # 0
05     [ 0, 1, 1, 0, 0, 0, 0 ], # 1
06     [ 1, 1, 0, 1, 1, 0, 1 ], # 2
07     [ 1, 1, 1, 1, 0, 0, 1 ], # 3
08     [ 0, 1, 1, 0, 0, 1, 1 ], # 4
09     [ 1, 0, 1, 1, 0, 1, 1 ], # 5
10     [ 0, 0, 1, 1, 1, 1, 1 ], # 6
11     [ 1, 1, 1, 0, 0, 0, 0 ], # 7
12     [ 1, 1, 1, 1, 1, 1, 1 ], # 8
13     [ 1, 1, 1, 0, 0, 1, 1 ]  # 9
14 ])
15
16 def predict(x) :
17     x = np.array([x])      # x should be array
18     h1 = np.dot(x,W1) + b1 # dense layer
19     h1 = np.maximum(0, h1) # ReLU
20     h2 = np.dot(h1,W2) + b2 # dense layer
21     h2 = np.maximum(0, h2) # ReLU
22     y = np.dot(h2,W3) + b3 # dense layer
23     y = 1/(1+np.exp(-y))
24     return y
25
26 xs = X
27 ys = []
28 for x in xs:
29     y = predict(x)
30     ys.append(y)
31
32 for y in ys:
33     print(y)
```

## 예측 시간 측정 1

여기서는 10개 데이터에 대한 예측 시간을 측정해 봅니다.

다음과 같이 예제를 수정합니다.

```
01 from model_data import *
02 import time
03
04 X=np.array([
05     [ 1, 1, 1, 1, 1, 1, 0 ], # 0
06     [ 0, 1, 1, 0, 0, 0, 0 ], # 1
07     [ 1, 1, 0, 1, 1, 0, 1 ], # 2
08     [ 1, 1, 1, 1, 0, 0, 1 ], # 3
09     [ 0, 1, 1, 0, 0, 1, 1 ], # 4
10     [ 1, 0, 1, 1, 0, 1, 1 ], # 5
11     [ 0, 0, 1, 1, 1, 1, 1 ], # 6
12     [ 1, 1, 1, 0, 0, 0, 0 ], # 7
13     [ 1, 1, 1, 1, 1, 1, 1 ], # 8
14     [ 1, 1, 1, 0, 0, 1, 1 ]  # 9
15 ])
16
17 def predict(x) :
18     x = np.array([x])      # x should be array
19     h1 = np.dot(x,W1) + b1 # dense layer
20     h1 = np.maximum(0, h1) # ReLU
21     h2 = np.dot(h1,W2) + b2 # dense layer
22     h2 = np.maximum(0, h2) # ReLU
23     y = np.dot(h2,W3) + b3 # dense layer
24     y = 1/(1+np.exp(-y))
25     return y
26
27 xs = X
28 ys = []
29 start = time.ticks_ms()
30 for x in xs:
31     y = predict(x)
32     ys.append(y)
33 end = time.ticks_ms()
34
35 for y in ys:
```

```
36     print(y)
37
38 print('Prediction time for 10 data :', end-start, 'ms')
```

```
Prediction time for 10 data : 18 ms
```

## 예측 시간 측정 2

여기서는 1개 데이터에 대한 예측 시간을 측정해 봅니다.

다음과 같이 예제를 수정합니다.

```
01 from model_data import *
02 import time
03
04 X=np.array([
05     [ 1, 1, 1, 1, 0, 0, 1 ], # 3
06 ])
07
08 def predict(x) :
09     x = np.array([x]) # x should be array
10     h1 = np.dot(x,W1) + b1 # dense layer
11     h1 = np.maximum(0, h1) # ReLU
12     h2 = np.dot(h1,W2) + b2 # dense layer
13     h2 = np.maximum(0, h2) # ReLU
14     y = np.dot(h2,W3) + b3 # dense layer
15     y = 1/(1+np.exp(-y))
16     return y
17
18 x=X[0]
19
20 start = time.ticks_ms()
21
22 y = predict(x)
23
24 end = time.ticks_ms()
25
26 print(y)
27
28 print('Prediction time for 1 data : ', end-start, 'ms')
```

Prediction time for 1 data : 2 ms