

파이썬 인공지능 3

02 활성화 함수 추가하기

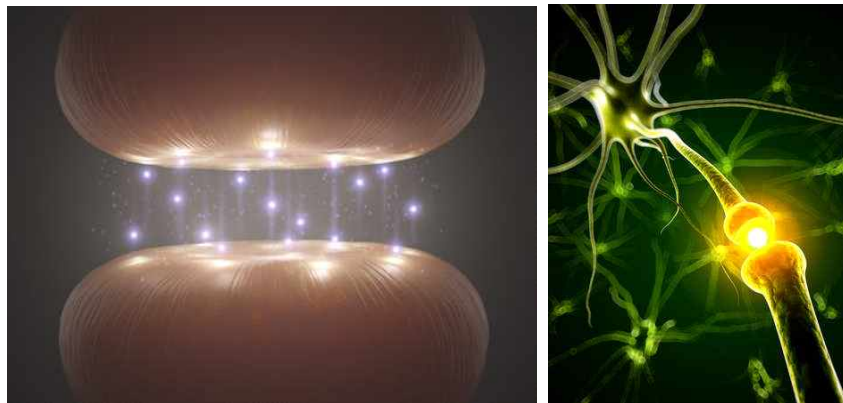
일반적으로 인공 신경의 출력 단에는 활성화 함수가 추가됩니다. 활성화 함수를 추가하면 입력된 데이터에 대해 복잡한 패턴의 학습이 가능해집니다. 또 인공 신경의 출력 값을 어떤 범위로 제한할 수도 있습니다. 여기서는 주로 사용되는 몇 가지 활성화 함수를 살펴보고, 활성화 함수가 필요한 이유에 대해서도 살펴봅니다. 그리고 활성화 함수를 추가한 인공 신경망을 구현해 봅니다.

01 활성화 함수의 필요성

여기서는 활성화 함수가 무엇인지, 활성화 함수는 왜 필요한지, 어떤 활성화 함수가 있는지 살펴봅니다.

활성화 함수는 무엇인가요?

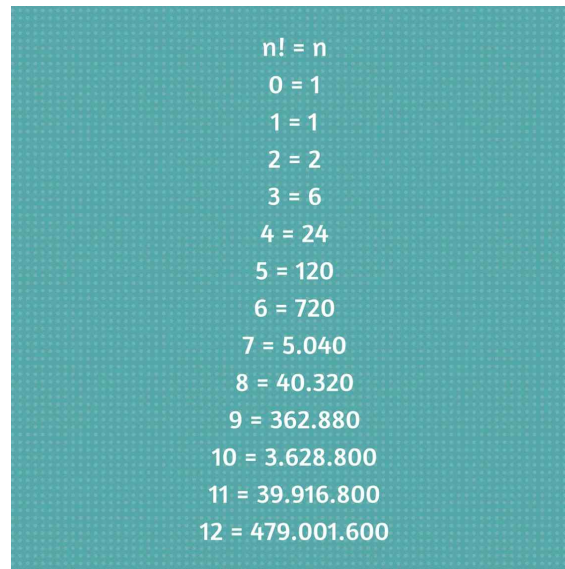
활성화 함수는 인공 신경망에 더해져 복잡한 패턴을 학습하게 해줍니다. 즉, 다양한 형태의 입력값에 대해 신경망을 거쳐 나온 출력값을 우리가 원하는 목표값에 가깝게 해 주기가 더 쉬워집니다. 우리 두뇌에 있는 생체 신경과 비교할 때, 활성화 함수는 신경 말단에서 다음 신경으로 전달될 신호를 결정하는 시냅스와 같은 역할을 합니다. 시냅스는 이전 신경 세포가 내보내는 출력 신호를 받아 다음 신경 세포가 받아들일 수 있는 입력 신호로 형태를 변경합니다. 마찬가지로 활성화 함수는 이전 인공 신경이 내보내는 출력 신호를 받아 다음 인공 신경이 받아들일 수 있는 입력 신호로 형태를 변경해 주는 역할을 합니다.



[시냅스]

활성화 함수는 왜 필요한가요?

앞에서 언급했던 생물학적 유사성과는 별도로 인공 신경의 출력값을 우리가 원하는 어떤 범위로 제한해 줍니다. 이것은 활성화 함수로의 입력이 $(x*w+b)$ 이기 때문입니다. 여기서 x 는 입력, w 는 인공 신경의 가중치, b 는 그것에 더해지는 편향입니다. $(x*w+b)$ 값은 어떤 범위로 제한되지 않으면 신경망을 거치며 순식간에 아주 커지게 됩니다. 특히 수백만 개의 매개변수(가중치와 편향)로 구성된 아주 깊은 신경망의 경우에는 더욱 그렇습니다. 인공 신경을 거치며 반복적으로 계산되는 $(x*w+b)$ 는 factorial 연산과 같은 효과를 내며 이것은 순식간에 컴퓨터의 실수 계산 범위를 넘어서게 됩니다. 인공 신경망을 학습시키다보면 Nan이라고 표시되는 경우가 있는데 이 경우가 그런 경우에 해당합니다. 다음은 factorial 연산의 예를 보여줍니다.

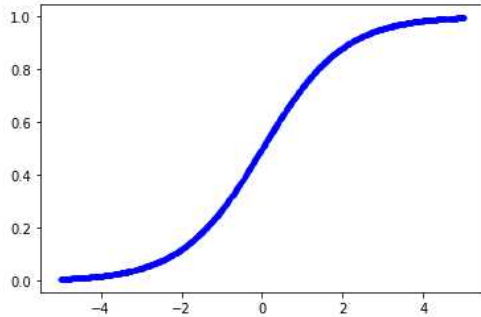


$n! = n$
$0 = 1$
$1 = 1$
$2 = 2$
$3 = 6$
$4 = 24$
$5 = 120$
$6 = 720$
$7 = 5.040$
$8 = 40.320$
$9 = 362.880$
$10 = 3.628.800$
$11 = 39.916.800$
$12 = 479.001.600$

어떤 활성화 함수가 있나요?

❶ 시그모이드

다음은 sigmoid 함수에 대한 그래프와 수식입니다.

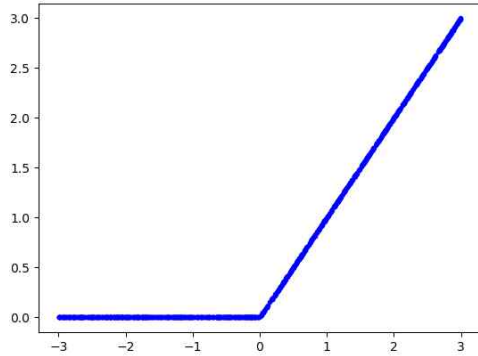


$$y = \frac{1}{1 + e^{-x}} \quad (-5 \leq x \leq 5)$$

시그모이드 활성화 함수는 단지 역사적인 이유로 여기에 소개되며 일반적으로 많이 사용되지 않습니다. 시그모이드 함수는 3층 정도로 구성된 인공 신경망에 적용될 때는 신경망 학습이 잘 되지만 깊은 신경망에 적용될 때는 신경망 학습이 잘 되지 않습니다. 시그모이드 함수는 계산에 시간이 걸리고, 입력값이 아무리 크더라도 출력값의 범위가 0에서 1사이로 매우 작아 신경망을 거칠수록 출력값은 점점더 작아져 0에 수렴하게 됩니다. 이것은 신경을 거치면서 신호가 점점 작아져 출력에 도달하는 신호가 아주 작거나 없어지는 것과 같습니다. 출력에 미치는 신호가 아주 작거나 없다는 것은 역으로 전달될 신호도 아주 작거나 없다는 것을 의미합니다. 시그모이드 함수는 일반적으로 0이나 1로 분류하는 이진 분류 문제에 사용됩니다. 심층 신경망에서 시그모이드 함수를 사용해야 할 경우엔 출력층에서만 사용하도록 합니다.

② ReLU

다음은 relu 함수에 대한 그래프와 수식입니다.

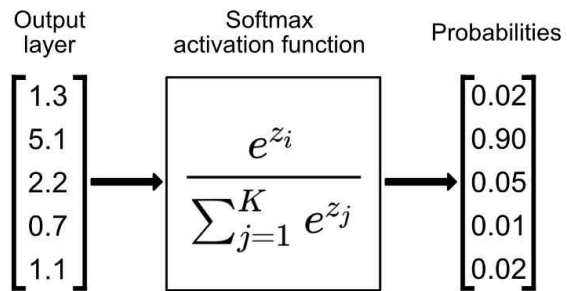


$$y = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \quad (-3 \leq x \leq 3)$$

ReLU 함수는 딥러닝에서 가장 인기 있는 활성화 함수입니다. 특히 합성곱 신경망(CNN)에서 많이 사용됩니다. ReLU 함수는 계산이 빠르고 심층 신경망에서도 신호 전달이 잘 됩니다. ReLU 함수의 경우 입력값이 음수가 될 경우 출력이 0이 되기 때문에 이런 경우에는 어떤 노드를 완전히 죽게 하여 어떤 것도 학습하지 않게 합니다. 이러한 노드가 많으면 많을수록 신경망 전체적으로 학습이 되지 않는 단점이 있습니다. ReLU의 다른 문제는 활성화 값의 극대화입니다. 왜냐하면 ReLU의 상한값은 무한이기 때문입니다. 이것은 가끔 사용할 수 없는 노드를 만들어 학습을 방해하게 됩니다. 이러한 문제들은 초기 가중치 값을 0에 아주 가까운 값으로 고르게 할당하여 해결할 수 있습니다. 일반적으로 은닉층에는 ReLU 함수를 적용하고, 출력층은 시그모이드 함수나 소프트맥스 함수를 적용합니다.

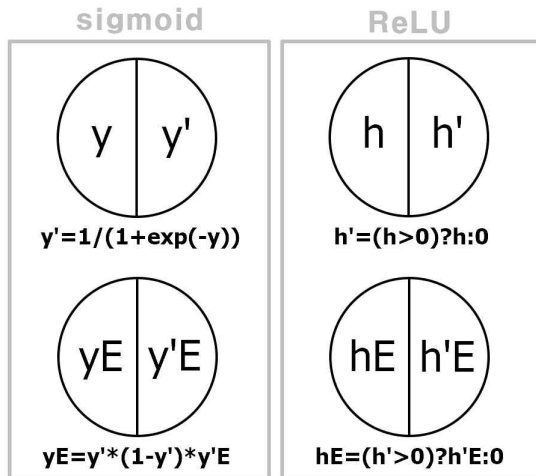
③ 소프트맥스

소프트맥스 활성화 함수는 시그모이드 활성화 함수가 더욱 일반화된 형태입니다. 이것은 다중 클래스 분류(classification) 문제에 사용됩니다. 시그모이드 함수와 비슷하게 이것은 0에서 1사이의 값들을 생성합니다. 소프트맥스 함수는 은닉층에서는 사용하지 않으며, 다중 분류(classification) 모델에서 출력층에서만 사용합니다. 소프트맥스 함수는 뒤에서 자세히 살펴봅니다.



02 활성화 함수의 순전파와 역전파

다음 그림은 sigmoid, ReLU 활성화 함수의 순전파와 역전파 수식을 나타냅니다.



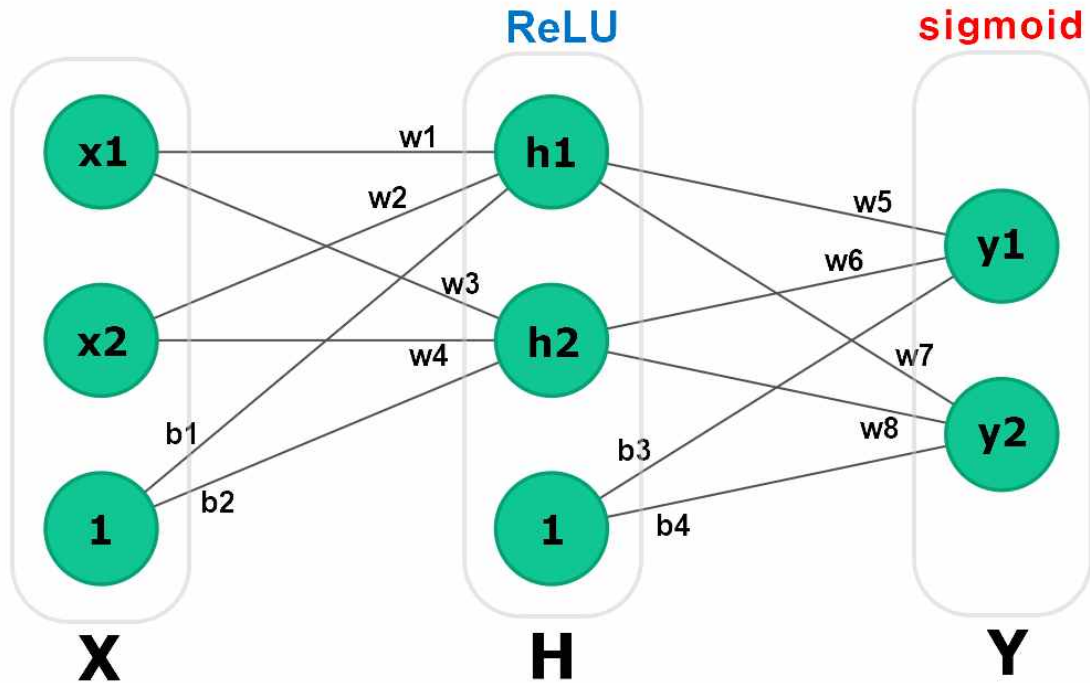
*** 역전파 수식 계산 시 순전파의 x' 값도 사용됩니다. 역전파 수식은 편미분을 이용하며 유도하며 유도 과정은 생략합니다.

- sigmoid 함수의 경우 순전파 출력 x' 값이 0이나 1에 가까울수록 역전파 $x'E$ 값은 0에 가까워집니다. 순전파 출력 x' 값이 0이나 1에 가깝다는 것은 순전파 입력값 x 의 크기가 양 또는 음의 방향으로 아주 크다는 의미입니다. 입력값 x 에 비해 출력값 x' 가 아주 작다는 것은 입력값 x 가 출력값 x' 에 주는 영향이 아주 작다는 것을 의미합니다. 따라서 $x'E$ 값이 역전파를 통해 xE 값에 전달되는 값도 아주 작아야 합니다.

- ReLU 함수의 경우 순전파 입력값 x 값이 0보다 크면 x 값이 x' 로 전달되며, 0보다 작거나 같으면 0값이 x' 로 전달됩니다. 역전파의 경우 순전파 출력값 x' 가 0보다 크면 $x'E$ 값이 xE 로 전달되며, 출력값 x' 가 0보다 작거나 같으면 xE 로 0이 전달됩니다. 이 경우 xE 에서 전 단계의 모든 노드로 전달되는 역전파 값은 0이 됩니다. 입력값 x 가 0보다 클 경우엔 출력값 x' 는 x 가 됩니다. 이 경우 x 는 1만큼 x' 로 전달된 것과 같습니다. 따라서 $x'E$ 값은 역전파를 통해 1만큼 xE 로 전달됩니다. 즉, $xE = x'E$ 가 됩니다. 입력값 x 가 0보다 작거나 같은 경우엔 출력값 x' 는 0이 됩니다. 이 경우 x 는 0만큼 x' 로 전달된 것과 같습니다. 따라서 $x'E$ 값은 역전파를 통해 0만큼 xE 로 전달됩니다. 즉, $xE = 0 * x'E = 0$ 이 됩니다.

활성화 함수 적용해 보기

지금까지 정리한 sigmoid 활성화 함수를 다음과 같이 인공 신경망에 적용해봅니다.



1. 다음과 같이 예제를 작성합니다.

422_5.py

```
01 from math import exp
02
03 x1, x2 = 0.05, 0.10
04
05 w1, w2 = 0.15, 0.20
06 w3, w4 = 0.25, 0.30
07 b1, b2 = 0.35, 0.35
08
09 w5, w6 = 0.40, 0.45
10 w7, w8 = 0.50, 0.55
11 b3, b4 = 0.60, 0.60
12
13 y1T, y2T = 0.01, 0.99
14
15 lr = 0.01
16
17 EPOCH = 1000
18
```



```

19 for epoch in range(EPOCH):
20
21     h1 = x1*w1 + x2*w2 + 1*b1
22     h2 = x1*w3 + x2*w4 + 1*b2
23     # RELU feed forward
24     h1=h1 if h1>0 else 0
25     h2=h2 if h2>0 else 0
26
27     y1 = h1*w5 + h2*w6 + 1*b3
28     y2 = h1*w7 + h2*w8 + 1*b4
29     # sigmoid feed forward
30     y1=1/(1+exp(-y1))
31     y2=1/(1+exp(-y2))
32
33     E = (y1-y1T)**2/2 + (y2-y2T)**2/2
34
35     y1E = y1 - y1T
36     y2E = y2 - y2T
37     # sigmoid back propagation
38     y1E=y1*(1-y1)*y1E
39     y2E=y2*(1-y2)*y2E
40
41     w5E = y1E*h1
42     w6E = y1E*h2
43     w7E = y2E*h1
44     w8E = y2E*h2
45     b3E = y1E*1
46     b4E = y2E*1
47
48     h1E = y1E*w5 + y2E*w7
49     h2E = y1E*w6 + y2E*w8
50     # RELU back propagation
51     h1E=h1E if h1>0 else 0
52     h2E=h2E if h2>0 else 0
53
54     w1E = h1E*x1
55     w2E = h1E*x2
56     w3E = h2E*x1
57     w4E = h2E*x2
58     b1E = h1E*1

```

```

59     b2E = h2E*1
60
61     w5 -= lr*w5E
62     w6 -= lr*w6E
63     w7 -= lr*w7E
64     w8 -= lr*w8E
65     b3 -= lr*b3E
66     b4 -= lr*b4E
67
68     w1 -= lr*w1E
69     w2 -= lr*w2E
70     w3 -= lr*w3E
71     w4 -= lr*w4E
72     b1 -= lr*b1E
73     b2 -= lr*b2E
74
75     if epoch % 100 == 99:
76         print(f'epoch = {epoch}')
77         print(f' y1 : {y1:.6f}')
78         print(f' y2 : {y2:.6f}')
79
80     if E<0.0000001:
81         break
82
83 print(f'w1,w3 = {w1:.6f},{w3:.6f}')
84 print(f'w2,w4 = {w2:.6f},{w4:.6f}')
85 print(f'b1,b2 = {b1:.6f},{b2:.6f}')
86 print(f'w5,w7 = {w5:.6f},{w7:.6f}')
87 print(f'w6,w8 = {w6:.6f},{w8:.6f}')
88 print(f'b3,b4 = {b3:.6f},{b4:.6f}')

```

6. 출력결과를 확인합니다.

```

epoch = 999
y1 : 0.359368
y2 : 0.809741
w1,w3 = 0.149033,0.247115
w2,w4 = 0.198065,0.294230
b1,b2 = 0.330653,0.292297
w5,w7 = 0.009232,0.629159
w6,w8 = 0.063927,0.677595
b3,b4 = -0.603755,0.997324

```

학습 회수에 따라 o1, o2값이 바뀌는 것을 확인합니다. o1, o2값이 각각 0.01, 0.99에 가까워지는 것을 확인합니다. 입력값 0.05, 0.10에 대해 목표값은 0.01, 0.99입니다.

17 EPOCH = 100000

```
epoch = 99999
  y1 : 0.013750
  y2 : 0.987571
w1,w3 = 0.198118,0.291117
w2,w4 = 0.296234,0.382345
b1,b2 = 1.312289,1.172902
w5,w7 = -0.956824,1.076556
w6,w8 = -0.813884,1.083658
b3,b4 = -1.981831,1.591643
```

06 활성화 함수 적용하기

② Tensorflow에 적용하기

다음은 Tensorflow 예제에 적용해 봅니다.

1. 이전 예제 415_2.py를 복사합니다.


2. 다음과 같이 예제를 수정합니다.

416_2.py

```
01~10 # 이전 예제와 같습니다.  
11 model=tf.keras.Sequential([  
12     tf.keras.Input(shape=(2,)),  
13     tf.keras.layers.Dense(2, activation='relu'),  
14     tf.keras.layers.Dense(2, activation='sigmoid')  
15 ]) # 신경망 모양 결정(W, B 내부적 준비)  
16~끝 # 이전 예제와 같습니다.
```

17 : 은닉 층에 활성화 함수 sigmoid를 적용합니다.

18 : 출력 층에 활성화 함수 sigmoid를 적용합니다.

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

```
W= [[0.14903286 0.24711493]  
    [0.19806509 0.29422987]]  
B= [0.33065325 0.29229653]  
W2= [[0.00923172 0.6291586 ]  
    [0.06392691 0.6775945 ]]  
B2= [-0.6037554  0.99732363]  
1/1 [=====] - 0s 18ms/step  
[[0.3591409 0.8098251]]
```

```
26 model.fit(X,YT,epochs=100000)
```

```
W= [[0.19811757 0.29111752]  
    [0.29623425 0.3823446 ]]  
B= [1.3122895 1.1729027]  
W2= [[-0.95682436 1.0765555 ]  
    [-0.8138841  1.083658 ]]  
B2= [-1.9818305 1.5916435]  
1/1 [=====] - 0s 18ms/step  
[[0.01375021 0.9875706 ]]
```

03 출력층에 linear 함수 적용해 보기

```
29 # sigmoid feed forward
30 # y1=1/(1+exp(-y1))
31 # y2=1/(1+exp(-y2))
32
33 E = (y1-y1T)**2/2 + (y2-y2T)**2/2
34
35 y1E = y1 - y1T
36 y2E = y2 - y2T
37 # sigmoid back propagation
38 # y1E=y1*(1-y1)*y1E
39 # y2E=y2*(1-y2)*y2E

epoch = 599
y1 : 0.010858
y2 : 0.989648
w1,w3 = 0.143157,0.241801
w2,w4 = 0.186315,0.283601
b1,b2 = 0.213147,0.186007
w5,w7 = 0.202730,0.533461
w6,w8 = 0.252676,0.582771
b3,b4 = -0.095253,0.730397
```

1. 이전 예제 415_2.py를 복사합니다.

2. 다음과 같이 예제를 수정합니다.

416_2.py

```
01~10 # 이전 예제와 같습니다.
11 model=tf.keras.Sequential([
12     tf.keras.Input(shape=(2,)),
13     tf.keras.layers.Dense(2, activation='relu'),
14     tf.keras.layers.Dense(2, activation='linear')
15 ]) # 신경망 모양 결정(W, B 내부적 준비)
16~끝 # 이전 예제와 같습니다.
```

17 : 은닉 층에 활성화 함수 sigmoid를 적용합니다.

18 : 출력 층에 활성화 함수 sigmoid를 적용합니다.

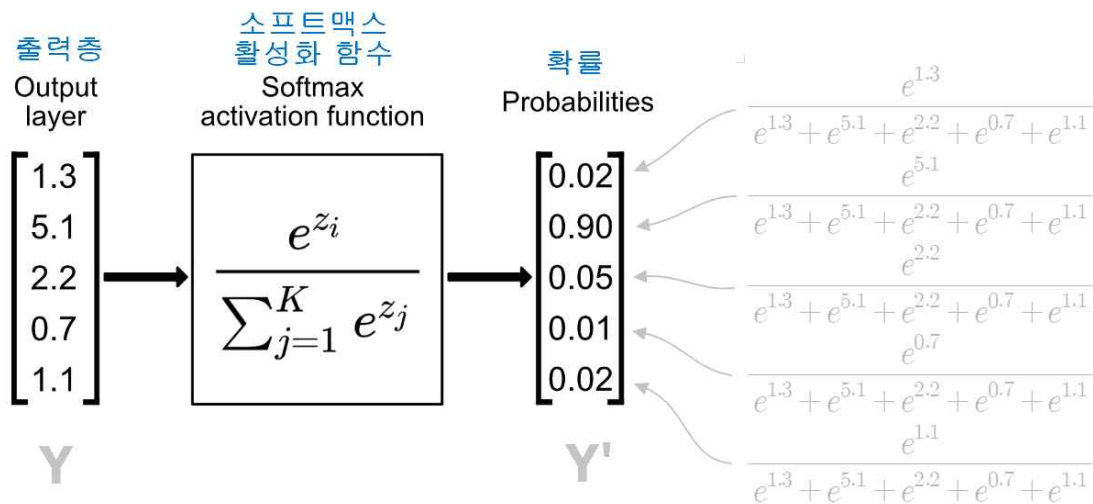
```
W= [[0.14315726 0.24180055]
 [0.18631403 0.2836011 ]]
B= [0.213141 0.18601276]
W2= [[0.20282482 0.5334216 ]
 [0.2527661 0.58273435]]
B2= [-0.09485418 0.7302336 ]
1/1 [=====] - 0s 17ms/step
[[0.01084895 0.98965186]]
```

03 출력층에 softmax 함수 적용해 보기

이전 단원에서 우리는 은닉 신경과 출력 신경에 sigmoid, relu 활성화 함수를 차례대로 적용해 보았습니다. 이 단원에서는 출력 신경의 활성화 함수를 소프트맥스(softmax)로 변경해 봅니다. softmax 활성화 함수는 크로스 엔트로피 오차(cross entropy error) 함수와 같이 사용되며, 분류(classification)에 사용됩니다.

softmax와 cross entropy

다음은 출력층에서 활성화 함수로 사용되는 소프트맥스(softmax) 함수를 나타냅니다.



소프트맥스 함수는 출력층에서 사용되는 활성화함수로 다중 분류(classification)를 위해 주로 사용됩니다. 소프트맥스 함수는 확률의 총합이 1이 되도록 만든 함수이며 아래에 나타낸 크로스 엔트로피 오차 함수와 같이 사용됩니다. 크로스 엔트로피 오차 함수의 경우 목표값(ykT) 중 하나만 1, 나머지는 0 값을 갖도록 목표값을 설정해야 합니다.

$$E = - \sum_k (y_{kT}^* \log(y_k))$$

우리는 앞에서 다음과 같은 평균 제곱 오차 함수를 살펴보았습니다.

$$E = \sum_k \frac{1}{2} (y_k - y_{kT})^2$$

평균 제곱 오차 함수의 경우 역전파 시 전파되는 오차가 다음과 같이 예측값과 목표값의 차인 것을 우리는 이미 앞에서 살펴보았습니다.

$$y_{kE} = y_k - y_{kT}$$

소프트맥스 함수는 크로스 엔트로피 함수와 같이 사용될 때만 역전파 시 소프트맥스 함수를 역으로 거쳐 전파되는 오차가 다음과 같이 예측값과 목표값의 차가 됩니다.

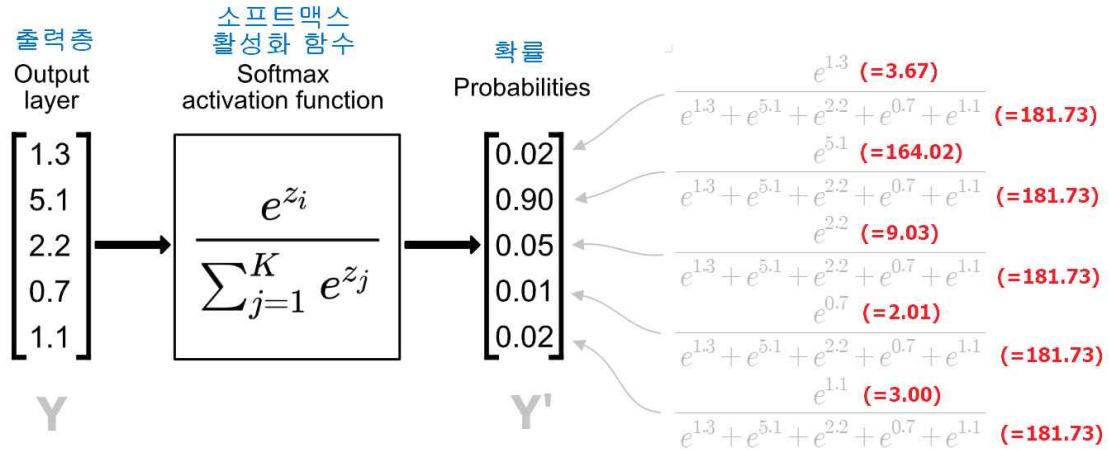
$$y_{kE} = y_k - y_{kT}$$

*** 이 수식은 크로스 엔트로피 함수와 소프트맥스 함수에 대해 차례대로 편미분을 적용하여 얻어진 수식이며 이 책에서는 수식에 대한 유도 과정은 생략합니다.

그래서 일반적으로 소프트맥스 함수를 활성화 함수로 사용할 경우 오차 함수는 크로스 엔트로피 오차 함수가 됩니다.

softmax 함수 구현해 보기

여기서는 다음 그림에 대해 softmax 함수를 구현해 봅니다.

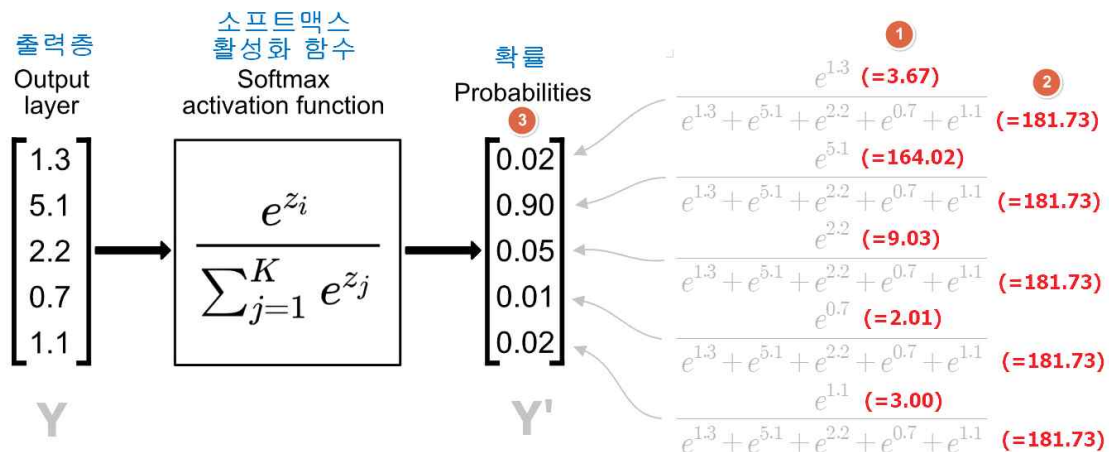


출력층에서 Y의 각 항목은 소프트맥스 활성화 함수를 거쳐 Y'의 각 항목으로 변환됩니다. Y'의 모든 항목의 합은 1이 됩니다.

다음과 같이 파이썬 셸에서 테스트해 봅니다.

```
>>> from math import exp as e
>>> (e(1.3), e(5.1), e(2.2), e(0.7), e(1.1))
(3.669296, 164.0219, 9.025014, 2.013753, 3.004166)
>>> sumY=e(1.3)+e(5.1)+e(2.2)+e(0.7)+e(1.1)
>>> sumY
181.7341
>>> (e(1.3)/sumY, e(5.1)/sumY, e(2.2)/sumY, e(0.7)/sumY, e(1.1)/sumY)
(0.02019047, 0.9025377, 0.04966054, 0.01108076, 0.01653056)
```

다음 그림의 ❶, ❷, ❸과 출력 결과를 비교해 봅니다.



❶ 첫 번째 결과는 Y의 각 항목에 대해 exp 함수를 적용한 결과입니다.

❷ 두 번째 결과는 첫 번째 결과의 모든 항목을 더한 값입니다.

❸ 세 번째 결과는 첫 번째 결과의 값들을 두 번째 결과의 값으로 나눈 결과입니다.

softmax 함수의 분모 크기 줄이기

앞에서 우리는 softmax 함수의 분모를 다음과 같이 계산하여 181.73 값을 얻었습니다.

$$e^{1.3} + e^{5.1} + e^{2.2} + e^{0.7} + e^{1.1} = 181.73$$

이 수식에서 Y의 항목 값이 하나라도 지나치게 클 경우의 계산 결과는 아주 큰 숫자가 되어 계산할 수 있는 실수의 범위를 벗어날 수 있습니다. 이런 경우 그 위험을 줄이기 위해 Y 행렬의 각 항목을 다음과 같이 바꿀 수 있습니다.

$$\begin{bmatrix} 1.3 - 5.1 \\ 5.1 - 5.1 \\ 2.2 - 5.1 \\ 0.7 - 5.1 \\ 1.1 - 5.1 \end{bmatrix} = \begin{bmatrix} -3.8 \\ 0.0 \\ -2.9 \\ -4.4 \\ -4.0 \end{bmatrix}$$

$$Y - 5.1 \rightarrow Y_c$$

위 그림에서 왼쪽의 Y 행렬에서 가장 큰 항목(5.1)을 찾아 행렬의 각 항목에 대해 빼주어 오른쪽의 Yc 행렬로 변환한 후, softmax 함수를 적용하여도 결과는 같게 됩니다. 다음은 이 과정에 대한 구체적인 예를 보여줍니다.

$$\begin{aligned} & \frac{e^{1.3}}{e^{1.3} + e^{5.1} + e^{2.2} + e^{0.7} + e^{1.1}} \quad \textcircled{1} \\ &= \frac{e^{1.3}/e^{5.1}}{(e^{1.3} + e^{5.1} + e^{2.2} + e^{0.7} + e^{1.1})/e^{5.1}} \quad \textcircled{2} \quad \textcircled{3} \quad \textcircled{4} \\ &= \frac{e^{1.3}/e^{5.1}}{e^{1.3}/e^{5.1} + e^{5.1}/e^{5.1} + e^{2.2}/e^{5.1} + e^{0.7}/e^{5.1} + e^{1.1}/e^{5.1}} \quad \textcircled{5} \quad \textcircled{6} \\ &= \frac{e^{(1.3-5.1)}}{e^{(1.3-5.1)} + e^{(5.1-5.1)} + e^{(2.2-5.1)} + e^{(0.7-5.1)} + e^{(1.1-5.1)}} \quad \textcircled{7} \quad \textcircled{8} \\ &= \frac{e^{-3.8}}{e^{-3.8} + e^{0.0} + e^{-2.9} + e^{-4.4} + e^{-4.0}} \quad \textcircled{9} \end{aligned}$$

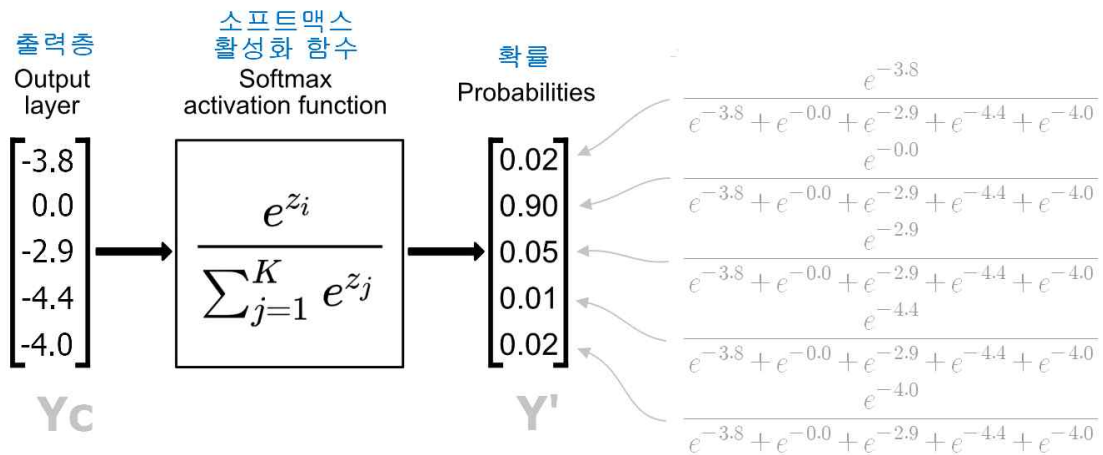
위에서 ① 수식은 최초의 Y 행렬의 첫 번째 항목에 대해 softmax 함수를 적용한 수식입니다. 여기서 ② 가장 큰 항목을 찾아 ③, ④와 같이 분자와 분모를 각각 나누어 줍니다. 그러면 ⑤, ⑥과 같이 분자와 분모를 바꿀 수 있습니다. ⑤, ⑥은 ⑦, ⑧과 같이 계산할 수 있으며, 결과적으로 ⑨와 같이 됩니다. 결과적으로 ①에서 최초의 Y의 첫 번째 항목 1.3에 대해 softmax 함수를 적용한 결과는 ⑨에서 변환된 Yc의 첫 번째 항목 -3.8에 softmax 함수를 적용한 결과와 같게 됩니다.

⑨에서 분모의 합은 다음과 같이 계산하여 1.11이 됩니다.

$$e^{-3.8} + e^{-0.0} + e^{-2.9} + e^{-4.4} + e^{-4.0} = 1.11$$

이렇게 하면 softmax 함수의 분모가 지나치게 커지는 것을 막을 수 있습니다.

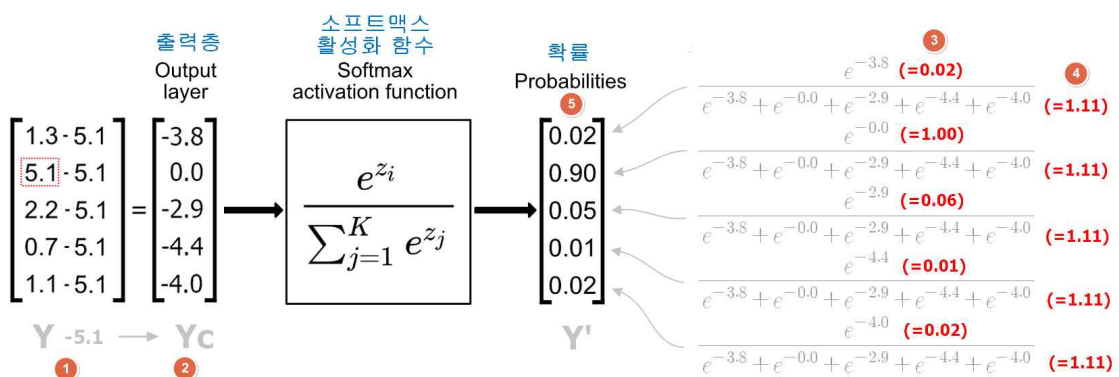
다음 그림은 변환된 Yc 행렬에 대해 softmax 함수를 적용하는 과정을 나타내는 그림입니다.



다음과 같이 파이썬 셸에서 테스트해 봅니다.

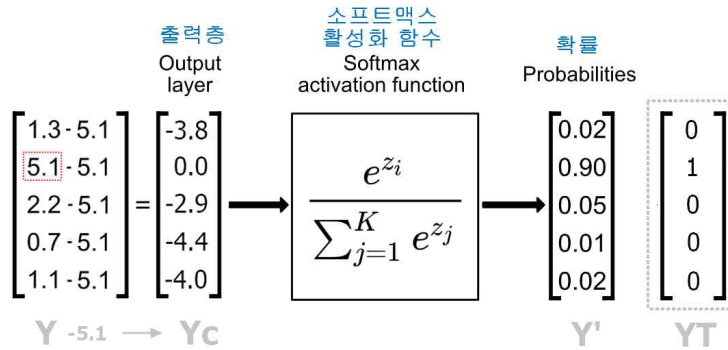
```
>>> (e(1.3-5.1),e(5.1-5.1),e(2.2-5.1),e(0.7-5.1),e(1.1-5.1))
(0.02237077, 1.0, 0.05502323, 0.01227734, 0.01831564)
>>> sumY=e(1.3-5.1)+e(5.1-5.1)+e(2.2-5.1)+e(0.7-5.1)+e(1.1-5.1)
>>> sumY
1.107987
>>> (e(1.3-5.1)/sumY, e(5.1-5.1)/sumY, e(2.2-5.1)/sumY, e(0.7-5.1)/sumY, e(1.1-5.1)/sumY)
(0.02019046, 0.9025376, 0.04966053, 0.01108076, 0.01653055)
```

다음 그림의 ①, ②, ③, ④, ⑤와 출력 결과를 비교해 봅니다.



cross entropy 오차 구현해 보기

softmax 함수를 사용할 경우 다음과 같이 하나의 목표값만 1이고 나머지는 0이 됩니다.



softmax 함수는 분류(classification)를 위해 사용하며 결과값 중 하나의 항목만 1에 가깝고 나머지는 0에 가까워지도록 학습하게 됩니다. 소프트맥스 함수는 확률의 총합이 1이 되도록 만든 함수이며 아래에 나타낸 크로스 엔트로피 오차 함수와 같이 사용합니다.

$$E = - \sum_k (y_{kT} * \log(y_k))$$

위 그림의 경우 오차는 다음과 같이 계산됩니다.

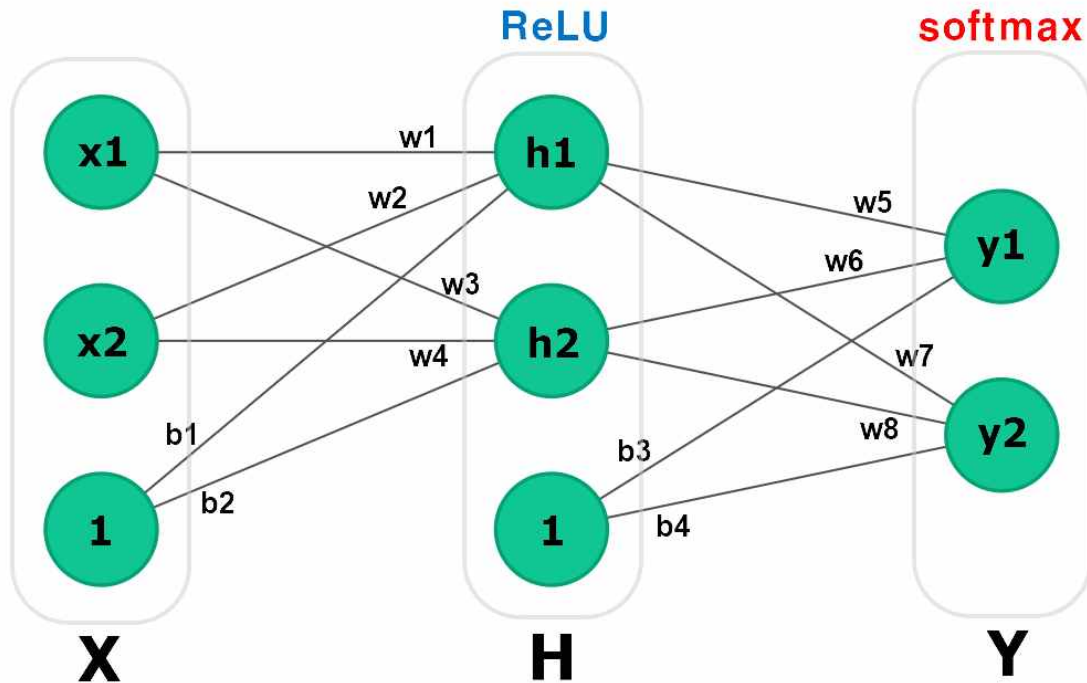
$$\begin{aligned} E &= -(0 * \log(0.02) + 1 * \log(0.90) + 0 * \log(0.05) + 0 * \log(0.01) + 0 * \log(0.02)) \\ &= -(1 * \log(0.90)) = 0.11 \end{aligned}$$

다음과 같이 파이썬 셸에서 테스트해 봅니다.

```
>>> from math import log as ln
>>> -(1*ln(0.90))
0.1053606
```

ReLU와 softmax

여기서는 은닉층 활성화 함수를 ReLU로 출력층 활성화 함수를 softmax로 적용해 봅니다. 다음 그림을 살펴봅시다.



1. 다음과 같이 예제를 작성합니다.

423_5.py

```
01 from math import exp, log
02
03 x1, x2 = 0.05, 0.10
04
05 w1, w2 = 0.15, 0.20
06 w3, w4 = 0.25, 0.30
07 b1, b2 = 0.35, 0.35
08
09 w5, w6 = 0.40, 0.45
10 w7, w8 = 0.50, 0.55
11 b3, b4 = 0.60, 0.60
12
13 y1T, y2T = 0., 1. # 크로스 엔트로피는 하나의 목표 값만 1, 나머지는 0
14
15 lr = 0.01
16
17 EPOCH = 100000
```

```

18
19 for epoch in range(EPOCH):
20
21     h1 = x1*w1 + x2*w2 + 1*b1
22     h2 = x1*w3 + x2*w4 + 1*b2
23     # RELU feed forward
24     h1=h1 if h1>0 else 0
25     h2=h2 if h2>0 else 0
26
27     y1 = h1*w5 + h2*w6 + 1*b3
28     y2 = h1*w7 + h2*w8 + 1*b4
29     # softmax feed forward
30     yMax=y1 if y1>y2 else y2
31     y1-=yMax
32     y2-=yMax
33     sumY=exp(y1)+exp(y2)
34     y1=exp(y1)/sumY
35     y2=exp(y2)/sumY
36
37     E = -(y1T*log(y1)+y2T*log(y2)) # cross entropy error
38
39     y1E = y1 - y1T
40     y2E = y2 - y2T
41     # softmax back propagation
42     # do nothing
43
44     w5E = y1E*h1
45     w6E = y1E*h2
46     w7E = y2E*h1
47     w8E = y2E*h2
48     b3E = y1E*1
49     b4E = y2E*1
50
51     h1E = y1E*w5 + y2E*w7
52     h2E = y1E*w6 + y2E*w8
53     # RELU back propagation
54     h1E=h1E if h1>0 else 0
55     h2E=h2E if h2>0 else 0
56
57     w1E = h1E*x1

```

```

58     w2E = h1E*x2
59     w3E = h2E*x1
60     w4E = h2E*x2
61     b1E = h1E*1
62     b2E = h2E*1
63
64     w5 -= lr*w5E
65     w6 -= lr*w6E
66     w7 -= lr*w7E
67     w8 -= lr*w8E
68     b3 -= lr*b3E
69     b4 -= lr*b4E
70
71     w1 -= lr*w1E
72     w2 -= lr*w2E
73     w3 -= lr*w3E
74     w4 -= lr*w4E
75     b1 -= lr*b1E
76     b2 -= lr*b2E
77
78     if epoch % 100 == 99:
79         print(f'epoch = {epoch}')
80         print(f' y1 : {y1:.6f}')
81         print(f' y2 : {y2:.6f}')
82
83     if E<0.0000001:
84         break
85
86     print(f'w1,w3 = {w1:.6f},{w3:.6f}')
87     print(f'w2,w4 = {w2:.6f},{w4:.6f}')
88     print(f'b1,b2 = {b1:.6f},{b2:.6f}')
89     print(f'w5,w7 = {w5:.6f},{w7:.6f}')
90     print(f'w6,w8 = {w6:.6f},{w8:.6f}')
91     print(f'b3,b4 = {b3:.6f},{b4:.6f}')

```

46 : 목표값을 각각 0과 1로 변경합니다.

54 : 출력층의 활성화 함수를 소프트맥스로 설정합니다.

56 : 오차 계산 함수를 크로스 엔트로피 오차 함수로 설정합니다.

3. 출력 결과를 확인합니다.

```
epoch = 99999  
y1 : 0.000052  
y2 : 0.999948  
w1,w3 = 0.209796,0.311644  
w2,w4 = 0.319592,0.423376  
b1,b2 = 1.546079,1.583957  
w5,w7 = -0.635543,1.535484  
w6,w8 = -0.621608,1.621555  
b3,b4 = -0.767205,1.967242
```

이상에서 출력층의 활성화 함수는 소프트맥스, 오차 계산 함수는 크로스 엔트로피 오차 함수인 인공 신경망을 구현해 보았습니다.

텐서플로우에 적용하기

```
01 import tensorflow as tf
02 import numpy as np
03
04 X=np.array([[.05,.10]]) # 입력데이터 <=
05 YT=np.array([[0.,1.]]) # 크로스 엔트로피는 하나의 목표 값만 1, 나머지는 0
06 W=np.array([[.15,.25],[.20,.30]]) # 가중치 <=
07 B=np.array([.35,.35]) # 편향
08 W2=np.array([[.40,.50],[.45,.55]]) # 가중치 <=
09 B2=np.array([.60,.60]) # 편향
10
11 model=tf.keras.Sequential([
12     tf.keras.Input(shape=(2,)),
13     tf.keras.layers.Dense(2, activation='relu'),
14     tf.keras.layers.Dense(2, activation='softmax')
15 ]) # 신경망 모양 결정(W, B 내부적 준비)
16
17 model.layers[0].set_weights([W,B])
18 model.layers[1].set_weights([W2,B2])
19
20 model.compile(optimizer='sgd',# 7공식, 학습
21               loss='categorical_crossentropy') # 2공식, 오차계산
22
23 Y=model.predict(X)
24 print(Y)
25
26 model.fit(X,YT,epochs=100000)
27
28 print('W=',model.layers[0].get_weights()[0])
29 print('B=',model.layers[0].get_weights()[1])
30 print('W2=',model.layers[1].get_weights()[0])
31 print('B2=',model.layers[1].get_weights()[1])
32
33 Y=model.predict(X)
34 print(Y)
```

```
W= [[0.20979609 0.31164446]
 [0.31959233 0.42337626]]
B= [1.5460793 1.5839566]
W2= [[-0.6355431  1.5354836 ]
 [-0.62160754  1.6215547 ]]
B2= [-0.7672052  1.9672418]
1/1 [=====] - 0s 17ms/step
[[5.1897896e-05 9.9994814e-01]]
```

손실 함수의 종류

Problem type	Last-layer activation	Loss function	Example
Binary classification	sigmoid	binary_crossentropy	Dog vs cat, Sentiment analysis(pos/neg)
Multi-class, single-label classification	softmax	categorical_crossentropy	MNIST has 10 classes single label (one prediction is one digit)
Multi-class, multi-label classification	sigmoid	binary_crossentropy	News tags classification, one blog can have multiple tags
Regression to arbitrary values	None	mse	Predict house price(an integer/float point)
Regression to values between 0 and 1	sigmoid	mse or binary_crossentropy	Engine health assessment where 0 is broken, 1 is new