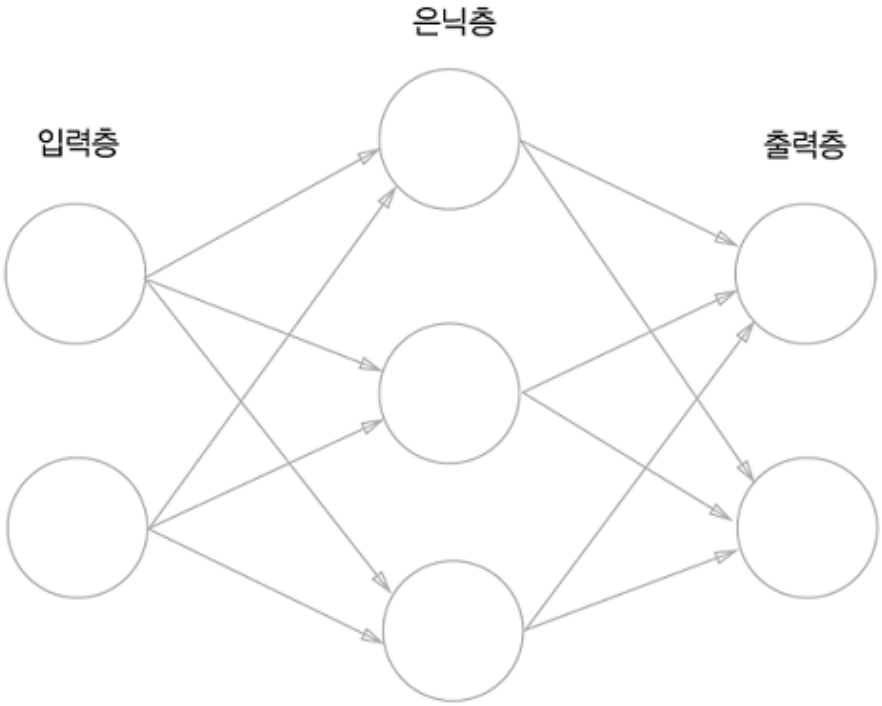
밑바닥부터 시작하는 딥러닝 3장 - 신경망 -

3.1 신경망이란?

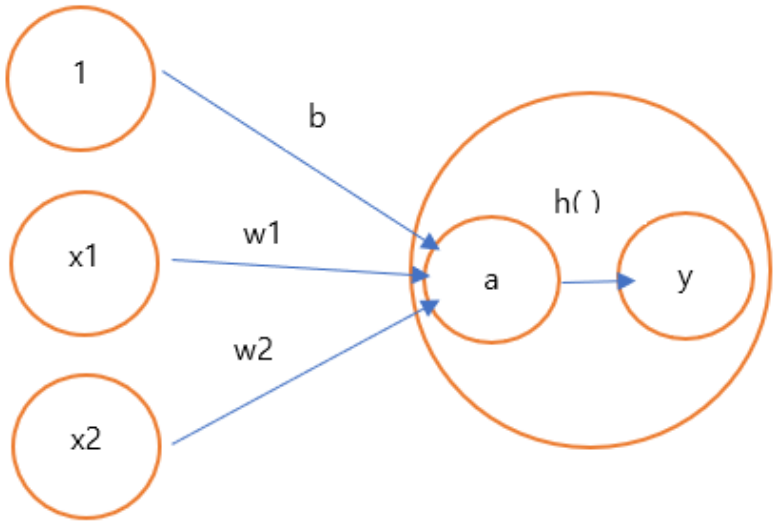


신경망 예시 (히든레이어 1층)

- 입력층: 학습 데이터셋이 입력되는 곳으로 학습 데이터의 features의 차원 수 만큼의 뉴런 개수를 가진다. 입력층은 단 한층만 존재한다.
- 출력층: 출력하고자 하는 데이터의 형태에 따라 노드의 수가 바뀐다. (ex, MNIST 분류 ⇒ 출력층 노드 10개, 시그모이드 사용해서 이진분류 ⇒ 출력층 노드 1개)
- 은닉층: 입력층과 출력층 사이 모든 층이다. 입력층, 출력층에 input, output되는 데이터셋과는 달리 은닉층은 사람 눈에는 보이지 않는다.

- 0층을 입력층, 1층을 은닉층, 2층을 출력층이라 한다. 위의 그림 속 신경망은 3층으로 구성되어있지만 가중치를 갖는 층이 2개이기 때문에 *2층 신경망* 이라고 한다.

> 💡 ***활성화 함수(activateion function,)*** : **입력 신호의 총합**을 **출력 신호로 변환**하는 함수. 변환된 신호를 다음 뉴런에 전달한다. 입력 신호의 총합이 활성화를 일으키는지를 정하는 역할을 한다.

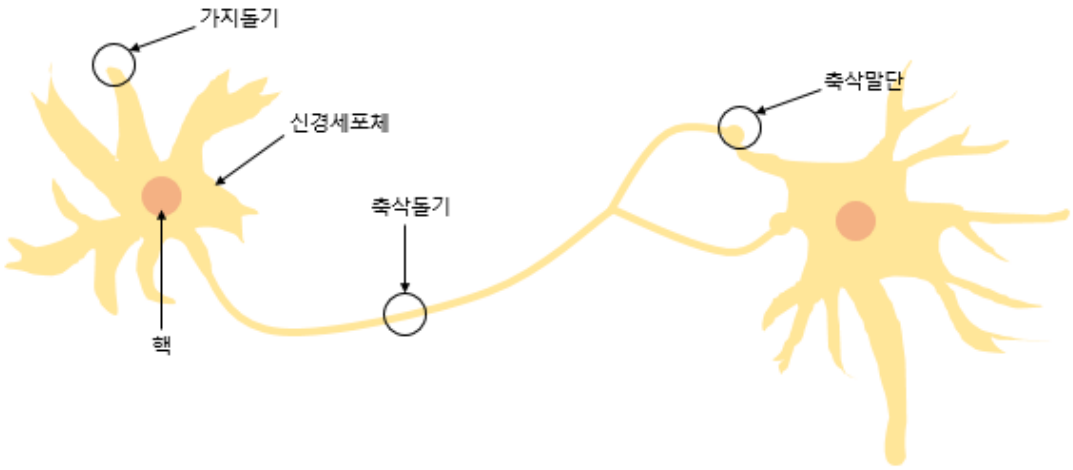


- **a = b (bias) +w1x1+w2x2 #가중치가 달린 입력 신호와 편향의 총합**
- **y=h(a) #a를 활성화함수 h()에 넣어 y를 출력**

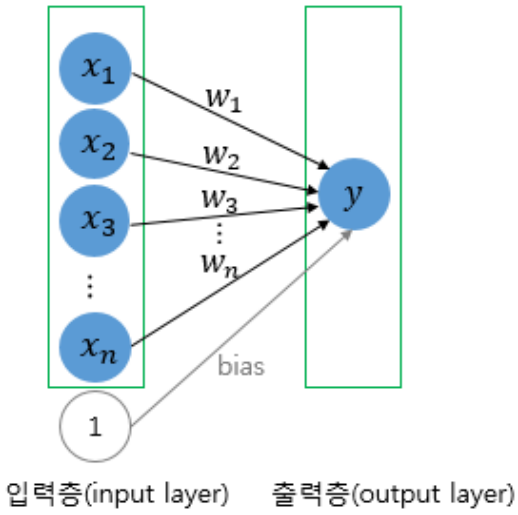
단순 퍼셉트론 : 단층 네트워크에서 ***계단 함수*** **(임계값을 경계로 출력이 바뀌는 함수)**를 활성화 함수로 사용한 모델

다층 퍼셉트론 : *신경망* **(1.여러 층으로 구성되고 2. 시그모이드 함수 등의 매끈한 활성화 함수를 사용하는 네트워크)**

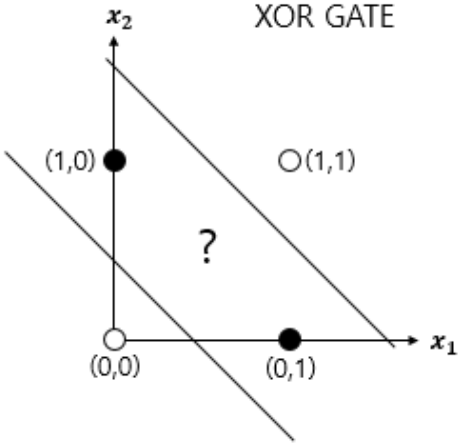
3.1.0 퍼셉트론



- 퍼셉트론(Perceptron, 단층 퍼셉트론 = SLP)은 프랑크 로젠블라트(Frank Rosenblatt)가 1957년에 제안한 초기 형태의 인공 신경망으로 다수의 입력으로부터 하나의 결과를 내보내는 알고리즘
- 퍼셉트론은 실제 뇌를 구성하는 신경 세포 뉴런의 동작과 유사하다. 뉴런은 가지돌기에서 신호를 받아들이고, 이 신호가 일정치 이상의 크기를 가지면 축삭돌기를 통해서 신호를 전달하는 것과 동일하게 동작한다.



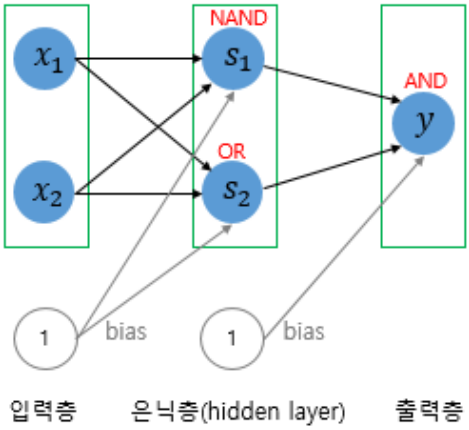
- 즉, 다수의 입력을 받는 퍼셉트론은 신경 세포 뉴런의 입력 신호와 출력 신호가 퍼셉트론에서 각각 입력값과 출력값에 해당된다



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

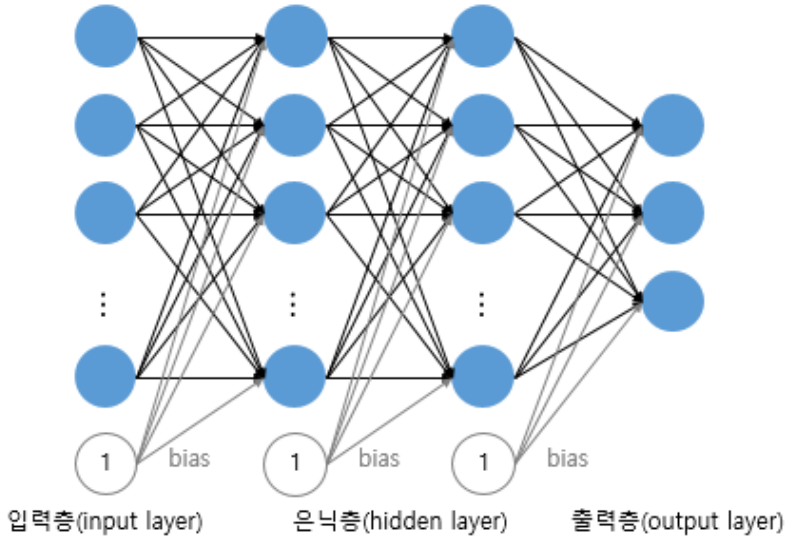
XOR GATE

💡 XOR GATE: x1, x2 중 어느 한 쪽이 1일 때 1을 출력하는 논리적 연산 ⇒ 선형 하나로는 분리가 불가하다.



MLP 구조도

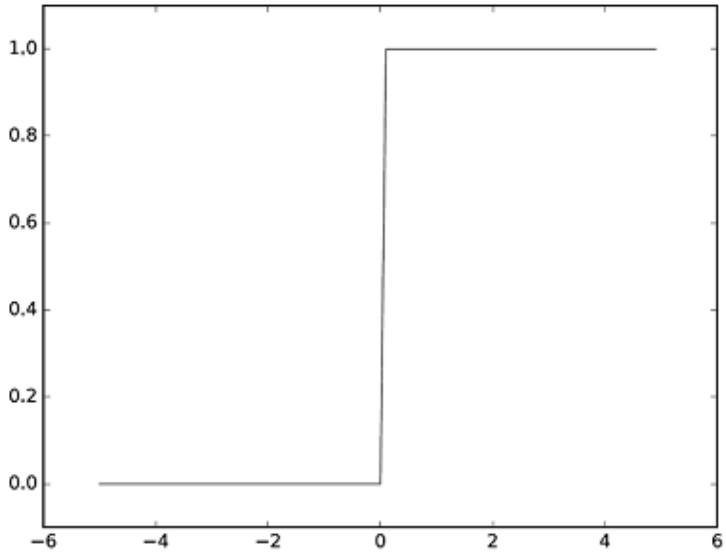
- XOR 게이트를 해결하기 위해 입력층, 출력층 외 한 층을 더 쌓는 다층 퍼셉트론 (MLP) 이 고안되었다.
- 즉, 층을 겹겹히 쌓아나가면서 선형 분류만으로는 풀지 못했던 문제를 해결할 수 있게 된 것이다.



- 위와 같이 은닉층이 2개 이상인 신경망 (즉, 3층 신경망부터) 을 **심층 신경망(Deep Neural Network, DNN)** 이라고 한다. 심층 신경망은 다층 퍼셉트론만 이야기 하는 것이 아니라, 여러 변형된 다양한 신경망들도 은닉층이 2개 이상이 되면 심층 신경망이라고 한다.
- 학습을 시키는 인공 신경망이 심층 신경망일 경우에는 이를 심층 신경망을 학습시킨다고 하여, **딥 러닝(Deep Learning)**이라고 한다.

3.2 활성화 함수

3.2.1 계단 함수



계단함수는 임계값을 기준으로 0과 1 중 하나의 값만 돌려준다. -> 뉴런 사이에 0 or 1이 흐른다.

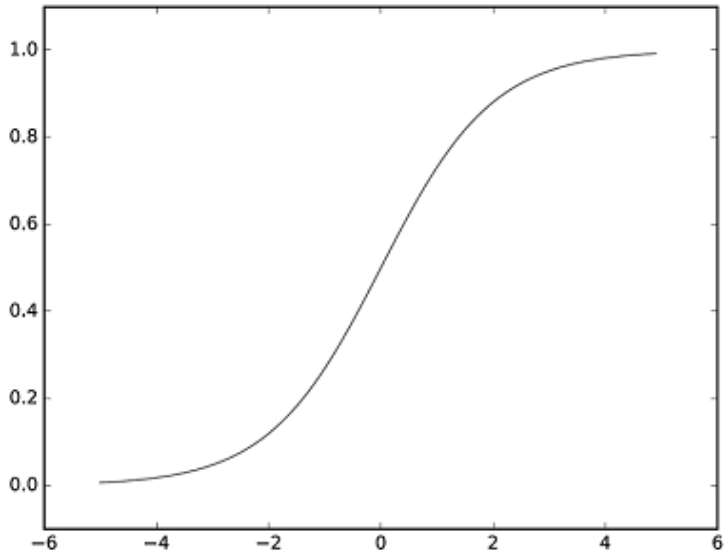
$$y = \begin{cases} 0\ (b + w_1x_1 + w_2x_2 \leq 0) \\ 1\ (b + w_1x_1 + w_2x_2 > 0) \end{cases}$$

시그모이드 함수 : 실수를 돌려준다. -> 연속적인 실수가 흐른다.

3.2.2 시그모이드 함수

$$h(x) = \frac{1}{1 + \exp(-x)}$$

신경망에서는 활성화 함수로 시그모이드 함수 (S자 곡선 형태) 를 이용하여 신호를 변환하고 그 변환된 신호를 다음 뉴런에 전달한다.



- ~~시그모이드가 계단함수(Step function)의 미분가능한 형태이기 때문이라고 하고 또는 작은 자극에는 감각을 거의 느끼지 못하다가 어떤 임계값을 넘어가야 감각을 느끼는 우리의 신경망 세포와 비슷하다는 ssul.. 이 있다.~~

- **계단 함수와 시그모이드 함수의 공통점**
  - 입력이 중요하면 큰 값을 출력하고 입력이 중요하지 않으면 작은 값을 출력한다. (입력이 작을 때의 출력은 0에 가깝고 혹은 0이고, 입력이 커지면 출력이 1에 가까워지는 혹은 1이 되는 구조이다.)
  - (입력이 아무리 작거나 커도) 출력은 0에서 1사이이다.
  - 비선형 함수(직선 1개로는 그릴 수 없는 함수)이다.

> 💡 **활성화 함수로 비선형 함수를 사용해야 하는 이유:** 선형 함수를 사용할 경우 신경망의 층 (= 히든레이어) 를 깊게 하는 이유가 없기 때문이다.

ex) h(x) = cx를 활성화 함수로 사용한 3층 네트워크를 나타내면

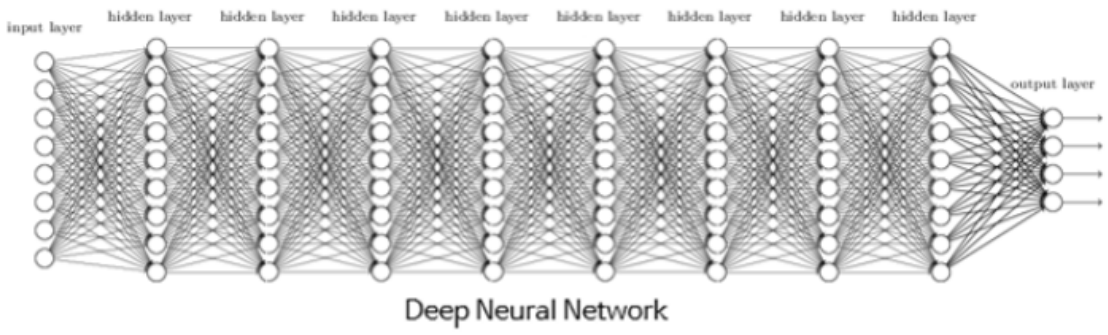
y(x) = h(h(h(x)))가 된다. 하지만 y(x) = ax와 똑같은 식이다. a=c^3일 뿐이다.

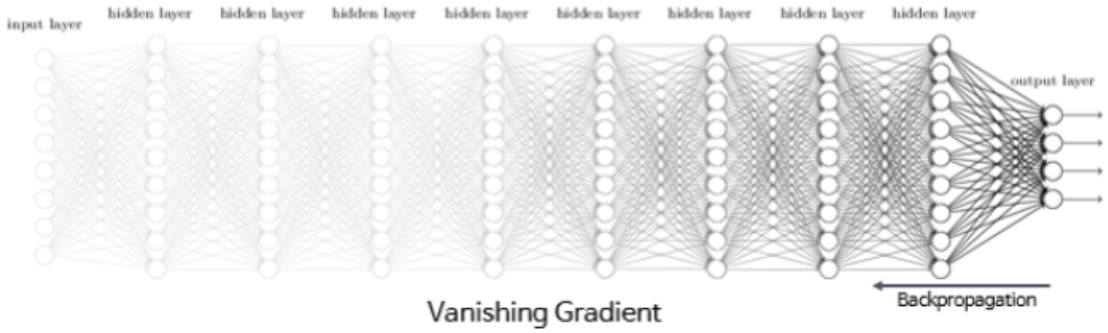
시그모이드의 장점

- 출력 값의 범위가 0~1이며, 매우 매끄러운 곡선을 가지므로 SGD 시행시 기울기가 급격하게 변해서 발산하는 기울기 폭주가 발생하지 않는다.
- 분류는 0과 1로 나뉘므로 출력 값이 어느쪽에 가까운지를 통해 어느 분류에 속하는지 쉽게 알 수 있다.

시그모이드의 단점

- 입력값이 아무리 크더라도 출력되는 값의 범위가 매우 좁기 때문에 SGD 수행 시에 범위가 너무 좁아서, 0에 수렴해버리는 기울기 소실 문제가 발생할 수 있다.
- 시그모이드 함수의 출력값이 모두 양수이기 때무에 경사하강법을 진행할 때 그 기울기 역시 모두 양수이거나 음수가 된다. 이는 기울기 업데이트가 지그재그로 변동하는 결과를 가지고 오고 학습 효율성을 감소시켜 학습에 더 많은 시간이 들어가게 한다.





💡 **기울기 소실 문제:** 미분 함수에 대하여 값이 일정 이상 커지는 경우 **미분값이 소실되는 문제**

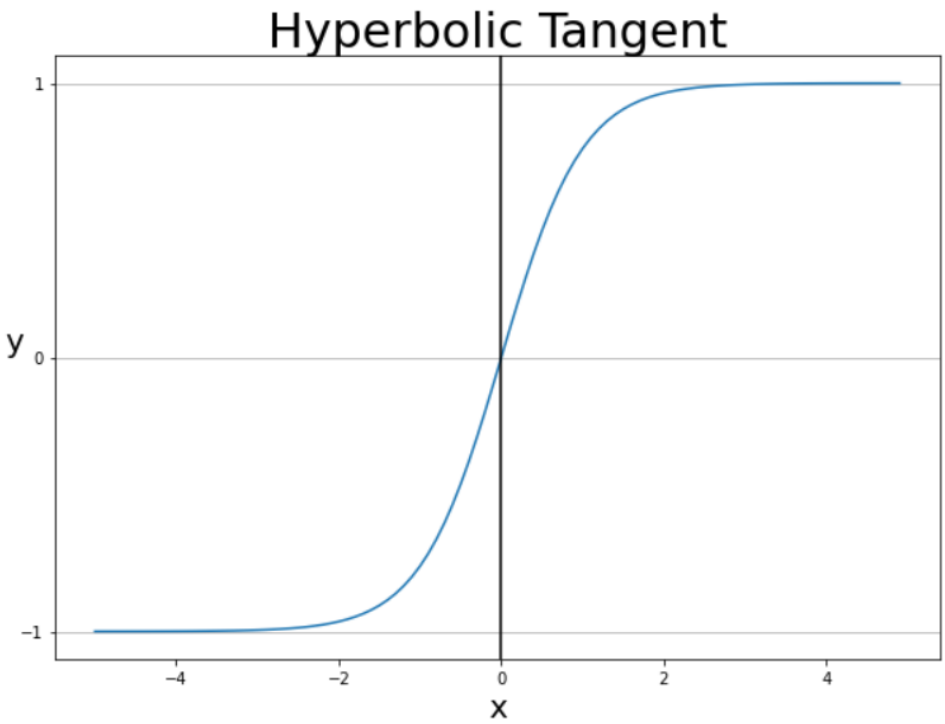
시그모이드 함수에 아무리 큰 값이 들어와도 (input) 0~1사이 값만 반환하므로 값이 일정 비율로 줄어들어 값이 현저하게 줄어든다. 또한 출력값의 중앙값이 0 이 아닌 0.5이며 모두 양수이기 때문에 출력의 **가중치 합이 입력의 가중치 합보다 커지게 된다. 이를 편향 이동 (Bias Gradient) 라고 하고 신호가 각 레이어를 통과할 때 마다 분산이 계속 커져 활성화함수의 출력이 최댓값과 최솟값인 0, 1에 수렴하게 된다.**

시그모이드 함수의 도함수에 들어가는 함수의 값이 0이나 1에 가까울수록 출력되는 값이 0에 가까워진다. 이로 인해 뉴런의 기울기값이 0이 되고 역전파 시 0이 곱해ㅕ서 기울기가 소멸 (kill) 되는 현상이 발생해버린다. 즉, 역전파가 진행될 수록 아래 층에 아무런 신호도 전달되지 않는 것이다.

⇒ 정리하자면, 시그모이드는 1) 출력값이 너무 작아 제대로 학습이 안되고 2) 시간이 다소 오래걸린다. 따라서 출력층에서 이진분류시 시그모이드를 사용하는 것은 상관없지만, 아래로 정보가 계속 흘러가는 은닉층에서는 시그모이드 함수를 활성화 함수로 사용해서는 안된다.

만약 입력층에서 사용하려고 한다면 이의 발전형인 하이퍼볼릭 탄젠트 함수를 사용하는 것이 더 좋다.

3.2.3 하이퍼볼릭 탄젠트 함수

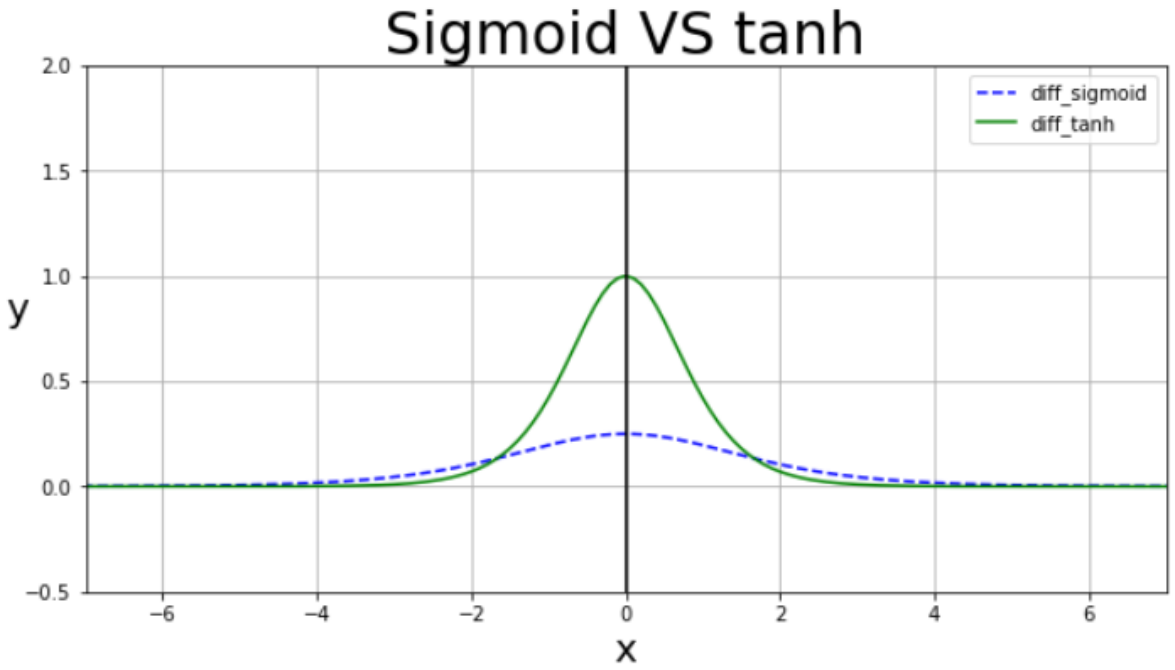


- 하이퍼볼릭 탄젠트 함수는 시그모이드 함수의 일부를 보완한 함수이다.
- 시그모이드 함수와 가장 큰 차이는 출력값의 범위로 하이퍼볼릭 탄젠트 함수는 -1에서 1 사이의 값을 출력하고 중앙값도 0이 된다.

	시그모이드 함수	하이퍼볼릭 탄젠트 함수
범위	0 ~ 1	-1 ~ 1
중앙값	0.5	0
미분 최댓값	0.3	1

- 즉 중앙값이 0이기 때문에 경사하강법 사용시 시그모이드 함수에서 발생하는 편향 이동이 발생하지 않는다.

- 기울기가 양, 음수 모두 나올 수 있어서 시그모이드 함수보다 학습 효율성이 뛰어나다.



시그모이드 vs 하이퍼볼릭 탄젠트 미분함수 비교

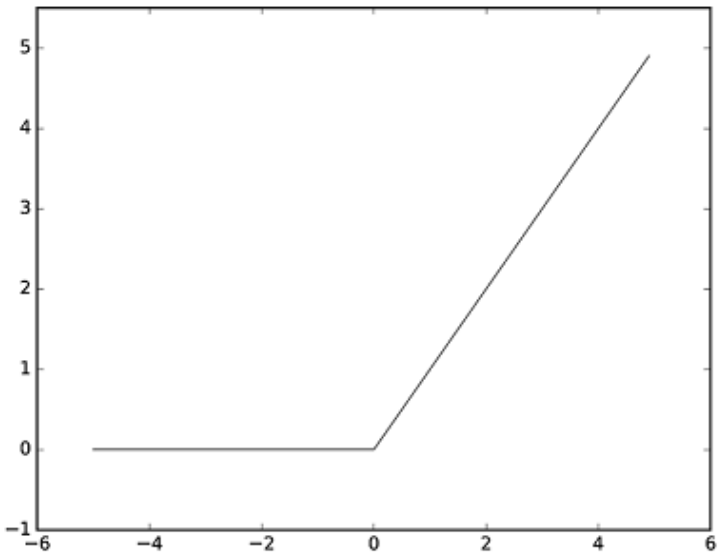
- 또한 시그모이드 함수보다 범위가 넓기 때무에 출력값의 변화폭이 크고 그로 인해 기울기 소실 증상이 더 적은 편이다.
- 따라서, 은닉층에서 시그모이드 함수와 같은 역할을 하는 레이어를 쌓는다면 하이퍼볼릭 탄젠트를 사용하는 것이 더 효과적이다.
- BUT, 시그모이드에 비해 범위가 넓은 것이지 하이퍼볼릭 탄젠트 역시 그 구간이 그렇게 크지 않기에 x가 -5보다 작거나 5보다 큰 경우 기울기가 0으로 작아져 소실되는 기울기 소실 현상 문제가 여전히 존재한다.

3.2.4 ReLU 함수

$$h(x) = \begin{cases} x(x > 0) \\ 0(x \leq 0) \end{cases}$$

- 입력이 0을 넘으면 그 입력을 그대로 출력하고 0이하이면 0을 출력하는 함수 ⇒ 특정 양수 값에 수렴하지 않는다.

- 우리말로 '정류된 선형 함수' 라고 하는데 (난생 처음 들어봄..) 간단하게 말해서 +/-가 반복되는 신호에서 - 흐름을 차단한다는 의미이다.



3.2.0 ReLU를 쓰자..

- ReLU 함수는 딥러닝 역사에 있어 한 획을 그은 활성화 함수로 해당 함수가 등장하기 이전에는 시그모이드 함수를 사용해서 딥러닝을 수행하였다.
- 그러나 ReLU함수는 출력값의 범위가 넓고 양수인 경우 자기 자신을 그대로 반환하기 때문에 시그모이드 함수에서 발생했던 **기울기 소실 문제가 발생하지 않는다.**
- 참고로, 시그모이드의 기울기 소실 문제는 1986~2006년까지 쭉 해결되지 않았으나 ReLU 함수의 등장으로 완전히 해결하게 되었다.
- ~~은닉층에서 상당히 많이 사용되는 활성화 함수로 은닉층에서 어떤 활성화 함수를 써야 할지 모르겠다면 그냥 ReLU를 써라라는 말이 있을 정도이다.~~

- 음수면 0, 양수면 자기 자신을 반환하는 단순한 공식인 만큼 **속도가 매우 빠르다.** (SGD 사용 시 시그모이드 OR 하이퍼볼릭 탄젠트 함수에 비해 6배 이상 빠르다고 한다..)
- ReLU 함수는 부드러운 (smooth) 한 구간에 도달하는 순간 가중치 업데이트 속도가 매우 느려지는 시그모이드 OR 하이퍼볼릭 탄젠트 함수와는 달리 편미분 (기울기) 시 1로 일정하므

로 가중치 업데이트 속도가 빠르다.

ReLU의 한계

- 죽어가는 ReLU 현상

> 💡 **죽어가는 ReLU 현상:** 음수값이 들어오면 모두 0으로 반환하는 문제가 있다보니 입력값이 음수인 경우 기울기도 모조리 0으로 나오게 된다. 즉, 입력값이 음수인 경우 한정하여 기울기가 0이 되므로 가중치 업데이트가 안되는 현상이 발생할 수 있다. ⇒ 즉 , 가중치가 업데이트 되는 과정에서 가중치 합이 음수가 되는 순간 ReLU는 0을 반환하기 때문에 해당 노드는 그 이후로 0만 반환하는 아무것도 변하지 않는 현상이 발생할 수 있는 것이다. 죽은 뉴런을 초래하는 현상이기에 이를 **"죽어가는 ReLU 현상"** 이라고 한다.

  - 또한 ReLU 함수는 기울기 소실 문제를 방지하기 위해 사용하는 활성화 함수이기 때문에 은닉층에서만 사용하는 것이 추천된다.
  - ReLU의 출력값은 0 or 양수이며 기울기 역시 0 or 1이므로 둘 다 양수이다. 따라서, 이로인해 시그모이드 함수처럼 가중치 업데이트 시 지그제그로 최적의 가중치를 찾아가는 **지그재그 현상**이 발새한다.

⇒ 기본적으로 은닉층에서는 ReLU 함수를 사용하지만 때에 따라 ReLU 함수의 단점이 두드러지는 경우가 존재한다. 이를 보완하기 위한 ReLU 함수의 형제 함수들이 존재한다.

ReLU family: 리키 렐루 (Leaky ReLU, LReLU), 파라미터 렐루 (Parameter ReLU, PReLU), E렐루 (ELU), SELU

참고) 많이 쓰는 순서: ReLU 가 성능이 별로면 → LReLU or ELU. 가능한 시그모이드는 잘 쓰지 않는 것이 좋다고 한다. (출처: cs231n)

이 외에도 하이퍼볼릭 탄젠트 함수는 대체하기 위한 softsign, ReLU 를 부드럽게 꺾은 듯한 softplus 등이 최근 발표되었고 높은 성능이 기대되는 Swish, ReLU, LReLU를 일반화한 Maxout 등 다양한 활성화 함수가 새로 만들어졌거나 지금도 추가되고 있다.

Functions

`deserialize(...)` : Returns activation function given a string identifier.

`elu(...)` : Exponential Linear Unit.

`exponential(...)` : Exponential activation function.

`gelu(...)` : Applies the Gaussian error linear unit (GELU) activation function.

`get(...)` : Returns function.

`hard_sigmoid(...)` : Hard sigmoid activation function.

`linear(...)` : Linear activation function (pass-through).

`relu(...)` : Applies the rectified linear unit activation function.

`selu(...)` : Scaled Exponential Linear Unit (SELU).

`serialize(...)` : Returns the string identifier of an activation function.

`sigmoid(...)` : Sigmoid activation function, `sigmoid(x) = 1 / (1 + exp(-x))`.

`softmax(...)` : Softmax converts a vector of values to a probability distribution.

`softplus(...)` : Softplus activation function, `softplus(x) = log(exp(x) + 1)`.

`softsign(...)` : Softsign activation function, `softsign(x) = x / (abs(x) + 1)`.

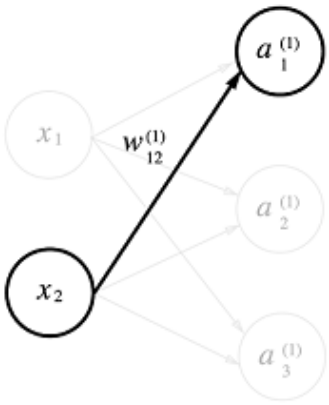
`swish(...)` : Swish activation function, `swish(x) = x * sigmoid(x)`.

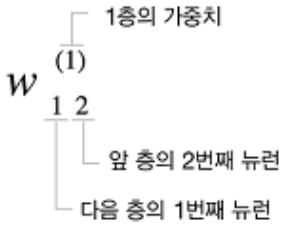
`tanh(...)` : Hyperbolic tangent activation function.

출처) https://www.tensorflow.org/api_docs/python/tf/keras/activations

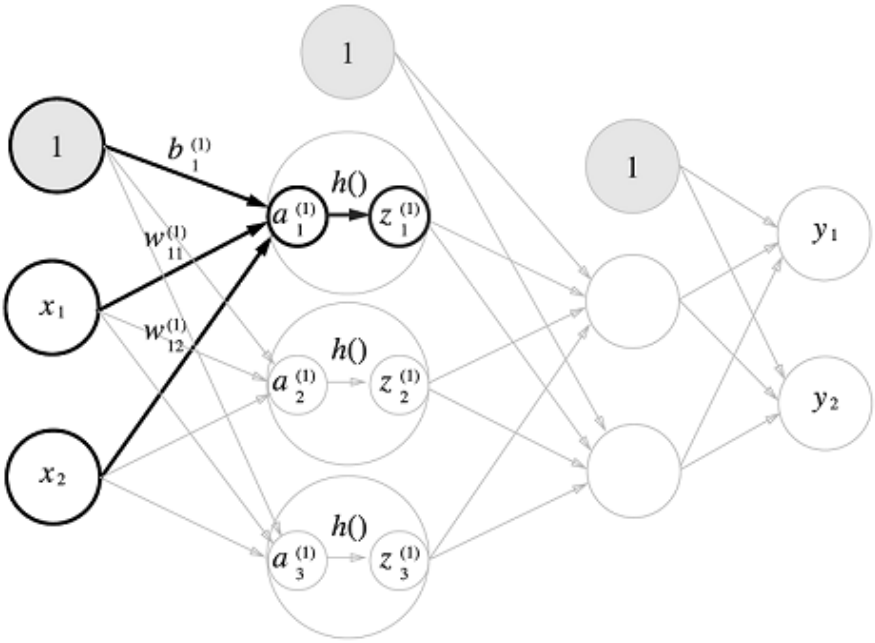
3.3 3층 신경망 구현

3.3.1 신경망 표기법





3.3.2 은닉층의 신호전달 구현



- 은닉층에서의 가중치 합(가중 신호와 편향의 총합)을 a로 표기하고 활성화 함수 h( )로 변환된 신호를 z로 표현한다.

$$a_1^{(1)} = w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + b_1^{(1)}$$

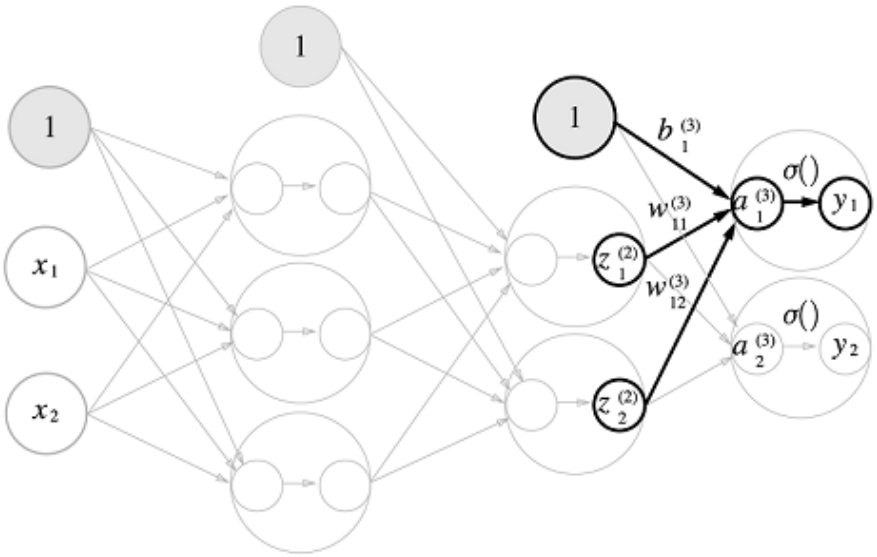
⇒

$$\mathbf{A}^{(1)} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{B}^{(1)}$$

$$A^{(1)} = \begin{pmatrix} a_1^{(1)} & a_2^{(1)} & a_3^{(1)} \end{pmatrix} \quad X = \begin{pmatrix} x_1 & x_2 \end{pmatrix}$$

$$B^{(1)} = \begin{pmatrix} b_1^{(1)} & b_2^{(1)} & b_3^{(1)} \end{pmatrix} \quad W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \end{pmatrix}$$

3.3.3 출력층으로의 신호전달 구현

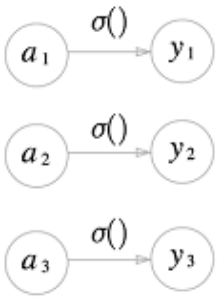


- **일반적으로 *회귀*에서는 출력층의 활성화 함수를 *항등함수*로, 이진*클래스 분류*에서는 *시그모이드 함수나 소프트맥스 함수*로, *다중 클래스 분류*에서는 *소프트맥스 함수*로 사용한다.**

3.4 출력층 설계하기

3.4.1 항등함수

• 항등함수(identity function): 입력을 그대로 출력



3.4.2 소프트맥스 함수

- 주로 이진분류의 출력층에서 자주 사용되는 시그모이드 함수와는 달리 세 개 이상을 분류되는 **다중 클래스 분류 시**의 출력층에서 자주 사용하는 활성화 함수이다.
- 분류될 클래스가 n개 라고 할 때, n차원의 벡터를 입력받아 **각 클래스에 속할 확률을 추정**한다.

  (”k번일 확률 / 전체확률”)

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

- n= 출력층의 뉴런 수 (= 총 클래스의 수), k= k번째 클래스
- 만약 클래스가 3 이라면

$$softmax(z) = [\frac{e^{z1}}{e^{z1}+e^{z2}+e^{z3}}, \quad \frac{e^{z2}}{e^{z1}+e^{z2}+e^{z3}}, \quad \frac{e^{z3}}{e^{z1}+e^{z2}+e^{z3}}] = [p_1, p_2, p_3]$$

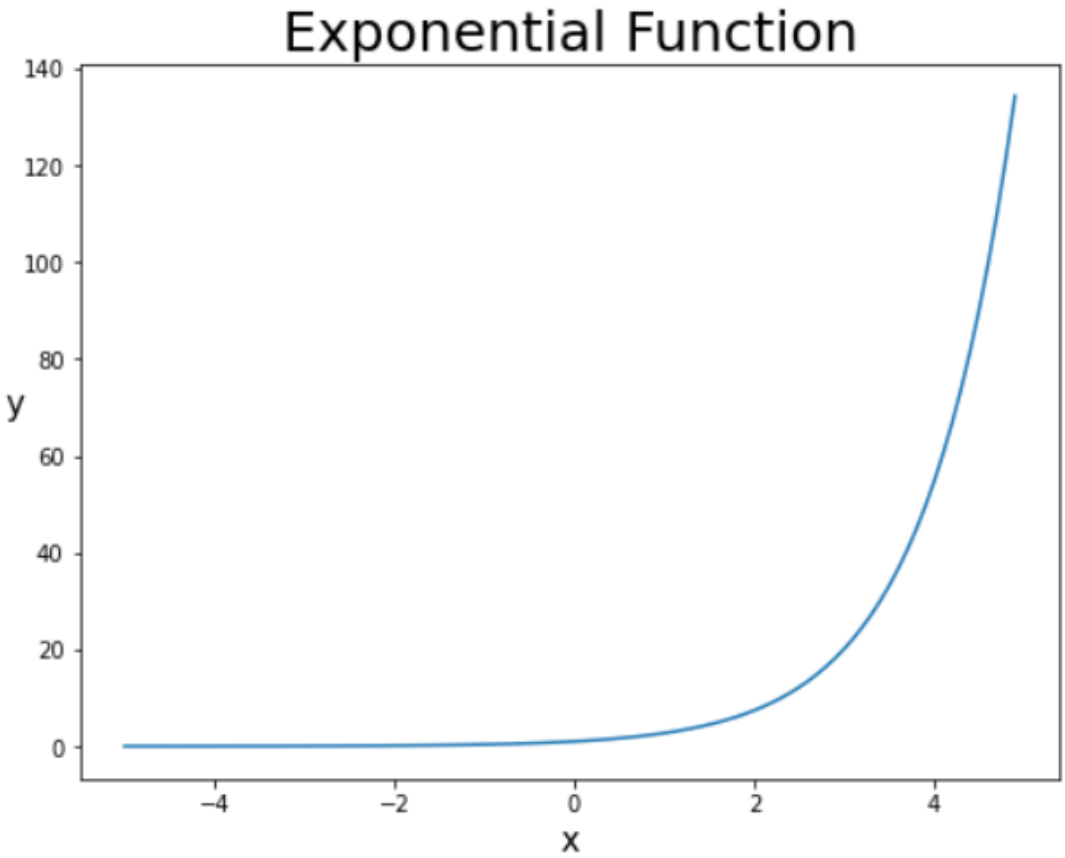
- 소프트맥스 함수의 분자는 입력 신호의 지수함수, 분모는 모든 입력 신호의 지수 함수의 합으로 구성된다.

💡 상수 e에 대한 지수함수를 사용하는 이유: 소프트맥스 함수는 시그모이드 함수로부터 유도된 것이다. (= 로지스틱 회귀)

로짓 (Logit) > 시그모이드 함수 > 소프트맥스 함수 순으로 유도됨

- 지수함수를 적용하면, 아무리 작은 값의 차이라도 확실히 구별될 정도로 커진다.
- exp(x) 의 미분은 원래 값과 동일하기 때문에 미분을 하기 좋다.

$$\frac{d}{dx}e^x = e^x$$



input 원소들의 최댓값 빼기 전

- 그런데 이때, 지수함수를 사용하는 소프트맥스 함수는 '오버플로우'의 문제가 발생해 수치가 '불안정'해질 수 있는 문제점이 있다.

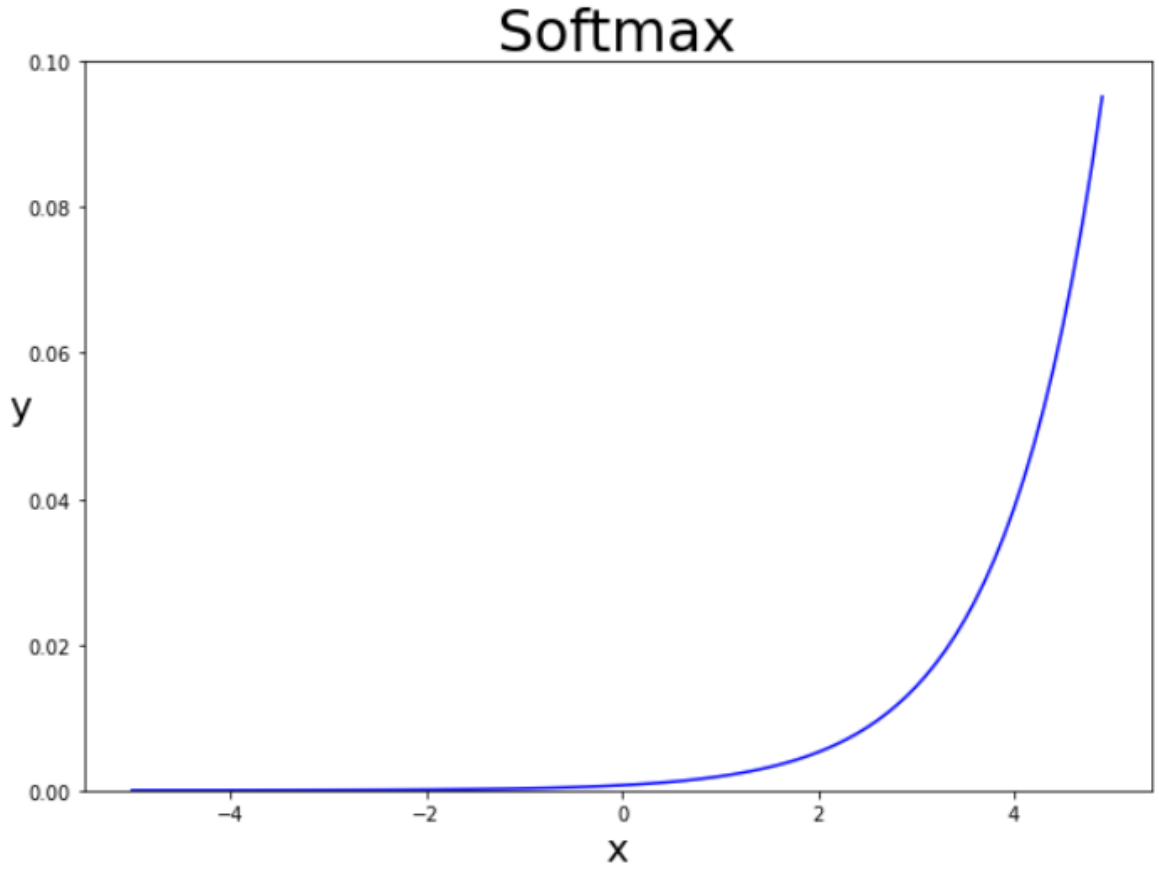
> 💡 • 오버플로우(overflow) : 표현할 수 있는 수의 범위가 한정되어 너무 큰값은 표현할 수 없는 경우를 의미한다.

- 소프트맥스 함수 구현 개선

$$
\begin{aligned}
y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n}\exp(a_i)} &= \frac{C\exp(a_k)}{C\sum_{i=1}^{n}\exp(a_i)} \\
&= \frac{\exp(a_k + \log C)}{\sum_{i=1}^{n}\exp(a_i + \log C)} \\
&= \frac{\exp(a_k + C')}{\sum_{i=1}^{n}\exp(a_i + C')}
\end{aligned}
$$

- 소프트맥스의 지수 함수를 계산할 때 어떤 정수를 더해도(혹은 빼도) 결과는 바뀌지 않는다.
- 따라서, 분자와 분모 양쪽에 C라는 상수를 곱하였으며 함수 exp() 안으로 옮겨져 logC를 만들었다. logC를 C ' 이라는 새로운 기호로 바꿔주었다.
- **C '에 어떤 값을 대입해도 상관없지만 오버플로를 막기 위해 입력 신호 중 최댓값을 이용하는 것이 일반적이다.**

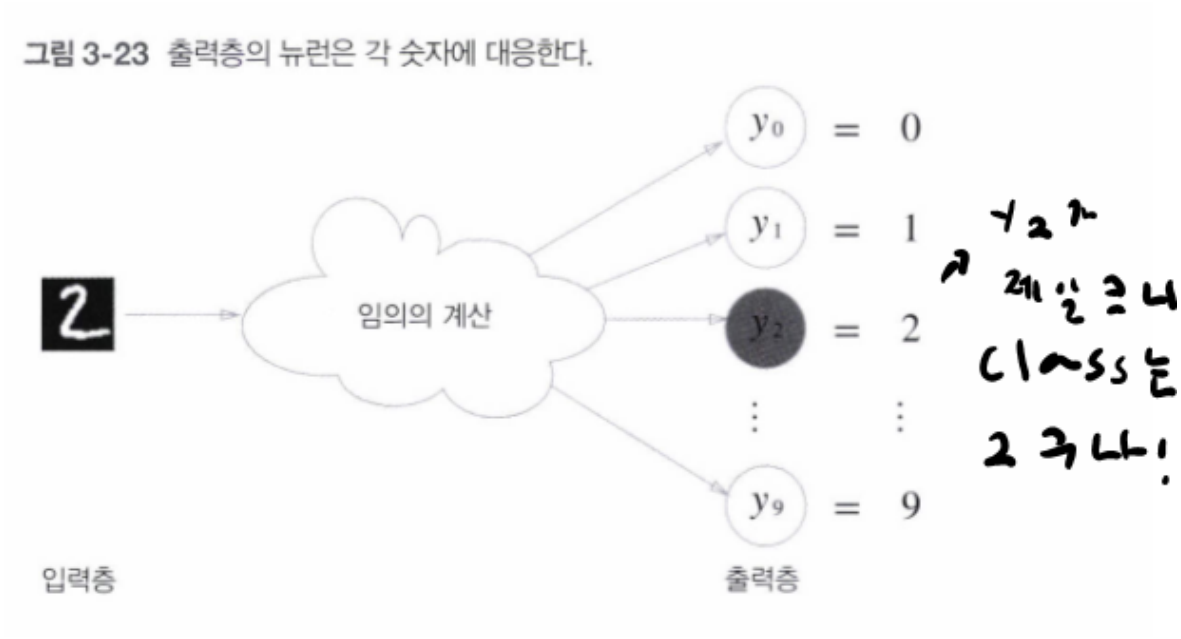
```
>>> a = np.array([1010, 1000, 990])
>>> np.exp(a) / np.sum(np.exp(a)) # 소프트맥스 함수의 계산
array([ nan,  nan,  nan])         # 제대로 계산되지 않는다.
>>>
>>> c = np.max(a)                 # c = 1010 (최댓값)
>>> a - c
array([  0, -10, -20])
>>>
>>> np.exp(a - c) / np.sum(np.exp(a - c))
array([  9.99954600e-01,   4.53978686e-05,   2.06106005e-09])
```



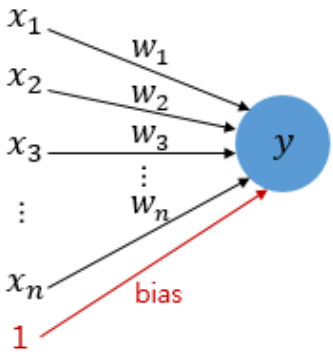
오버플로우 개선

- **소프트맥스 함수의 특징**
  - 출력값은 0에서 1.0 사이의 실수이다.
  - 출력의 총합은 1이다. -> 즉, **확률**로 해석가능하다.
  - 지수함수가 단조 증가 함수 (a>b 이면 f(a) > f(b)인 함수를 의미한다.) 이기 때문에 소프트맥스 함수를 적용해도 **각 원소의 대소 관계는 변하지 않는다.** 결과적으로 신경망으로 분류할 때는 출력층의 소프트맥스 함수를 생략해도 된다.

3.4.4 출력층의 뉴런 수 정하기



- 출력층의 뉴런 수는 풀려는 문제에 맞게 적절히 정해야 한다.
- *분류* 에서는 분류하고 싶은 클래스 수로 설정하는 것이 일반적이다.



3.5 추가

3.5.1 순전파 VS 역전파

- 신경망에서 데이터 흐름은 입력층 > 출력층의 방향으로 전달되었는데 이를 순전파라고 한다.
- BUT, 인공 신경망이 스스로 가장 적합한 가중치를 찾아나가는 것을 뜻하는 “학습” 은 역전파 과정을 통해 이루어진다.

⇒ 정리 : 먼저 신경망의 틀을 만들고, 그 신경망에서 가장 적합한 가중치를 찾는 것 ⇒ 학습 = 모델을 만든다고 하며 역전파 과정을 통해 진행됨 > 모델 완성 후 이 모델에 데이터를 집어넣고 그 결과를 확인 하는 것은 입 → 출력 순으로 진행되니 순전파 되어 실행된다고 할 수 있음