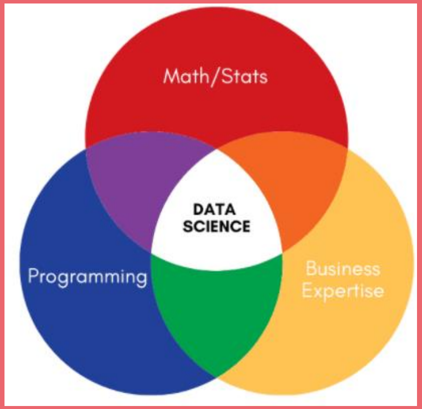
Neural Network

인하대학교
데이터사이언스 학과
김 승 환

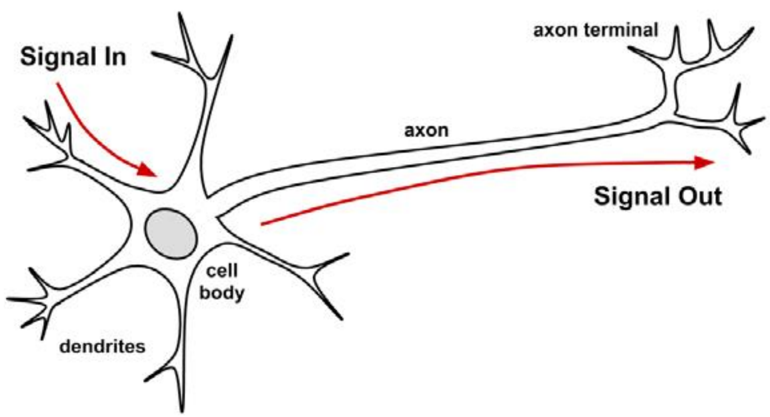
swkim4610@inha.ac.kr

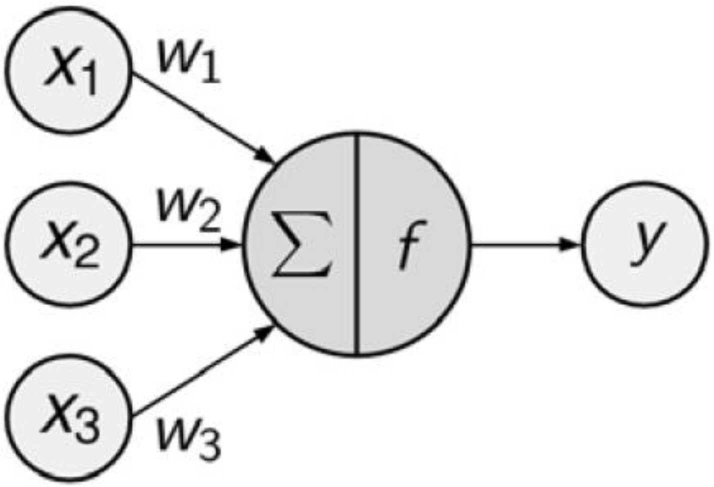


1 Neural Network

신경망 모형(Warren McCulloch and Walter Pitts, 1943)은 인간의 뇌를 수학적 모형으로 표현하여 인간처럼 판단을 수행하고자 하는 아이디어로 출발하였다.

이 아이디어는 $f(\sum w_i x_i + b)$ 의 값이 y 값이 되도록 미지수 w 값을 구하고자 하는 것이다.



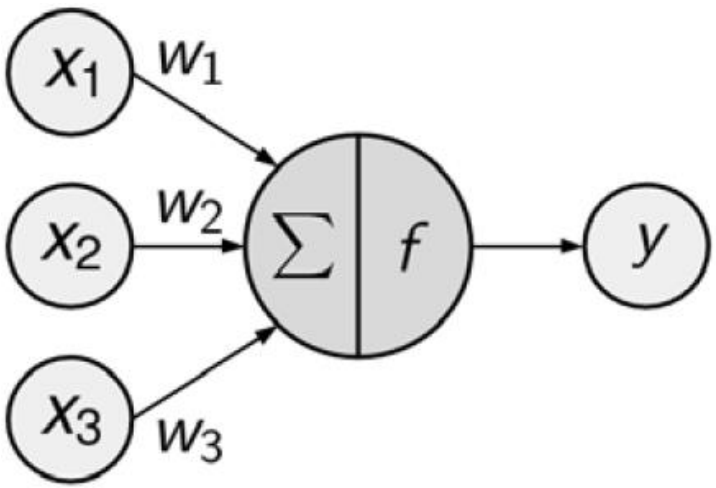


입력 X1, X2, X3 값에 각 가중치를 곱하는데 가중치가 크다는 것은 해당 자극이 신경을 활성화하는데 중요한 자극이라는 의미이다. 즉, 원하는 결과가 어떤 자극에 의해 나타나는지를 가중치로 구하는 것이다.
f는 활성함수(Activation function)이라고 한다.

1 Neural Network

신경망 모형에서 활성 함수를 Sigmoid를 사용할 경우, 신경망 모형은 Logistic Regression(Cox, 1958) 모형이 된다. 서로 접근방법은 달랐지만 McCulloch-Pitts의 신경망 모형과
Cox의 로지스틱 회귀는 결국 같은 모형이다.

y는 종속변수이고, X1, X2, X3는 독립변수이다.
일반적으로 신경망에서는 X1~X3를 Feature, y를 hypothesis, t는 target 이라고 부른다.

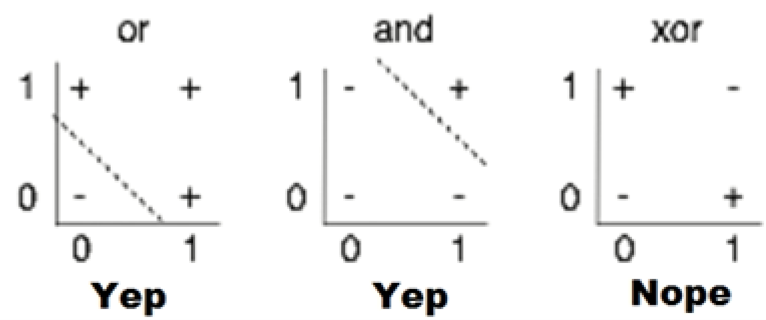


$$Y = S(w_1x_1 + w_2x_2 + w_3x_3 + b)$$

$$= \frac{1}{1+exp(-(w_1x_1+w_2x_2+w_3x_3+b))}$$

1 Neural Network

초기 신경망 모형은 Linear 한 모형으로 or, and 는 풀 수 있으나 xor는 풀 수 없어 많은 사람들이 xor문제에 집중하였다. 이후, 1969년 Minsky는 Hidden Layer를 사용하는 Multiple layer perceptron을 사용해 xor문제를 풀 수 있음을 증명했는데 문제는 MLP의 가중치를 구할 수 없다는 문제에 직면했다.



이후, 1974년 Backpropagation 알고리즘이 나와 MLP의 가중치를 구할 수 있게 되었으나 그 당시 주목받지 못하다가 1986년 Hinton에 다시 동일한 내용의 논문을 발표하면서 부터 다시 주목받기 시작했다.

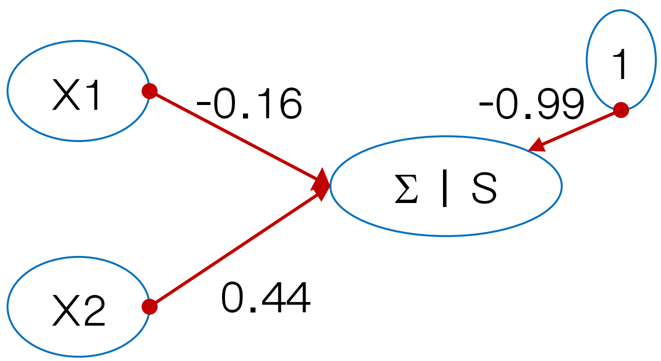
이후, 10년 정도 Neural Network이 유행했으나 많은 단점을 노출하면서 다시 시들해 졌다.

단점으로는 대표적으로 Over-fitting과 layer의 수가 커질수록 학습이 안되는 문제가 있다.

2006, 2007년에 문제에 대한 솔루션이 나오면서 신경망 모형은 다시 Deep Learning이란 이름으로 Re-Branding 하였다. 이후, 빅데이터 등 호재를 만나면서 특히, Image인식 분야에서 본격적으로 딥러닝의 시대를 열었다.

1 Neural Network

먼저, 임의의 W 값으로 아래 표와 같은 연산을 수행하여 error를 구한다.
이후, 이 오차를 줄이는 방향으로 W 값을 update 한다. 아래의 과정을 만족된 해가 나올 때까지 반복한다.



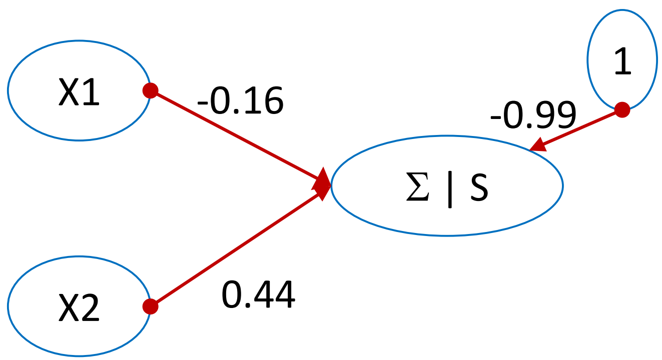
$$\begin{pmatrix} 0.12 \\ 0.73 \\ -0.61 \end{pmatrix} = \begin{pmatrix} -0.16 \\ 0.44 \\ -0.99 \end{pmatrix} - \begin{pmatrix} -0.28 \\ -0.29 \\ -0.38 \end{pmatrix}$$

$$W = W - t(X) \cdot error \cdot S'(Y)$$

x1	x2	sum	Y	T	error
0	0	0*-0.16+0*0.44-0.99=-0.99	1/(1+exp(0.99)) = 0.27	0	0.27
0	1	0*-0.16+1*0.44-0.99=-0.55	1/(1+exp(0.55)) = 0.36	1	-0.64
1	0	1*-0.16+0*0.44-0.99=-1.15	1/(1+exp(1.15)) = 0.24	1	-0.76
1	1	1*-0.16+1*0.44-0.99=-0.71	1/(1+exp(0.71)) = 0.33	1	-0.67

1 Neural Network

전 페이지를 복습하는 의미에서 아래의 표를 완성해보세요~



x1	x2	sum	Y	T	error
0	0				
0	1				
1	0				
1	1				

Neural Network

1943 년에 제안된 초기 신경망 모형은 or, and, xor 문제를 푸는 것이었다.
or, and는 해결하였으나 xor를 만족하는 해를 구하지 못했다.
이후, 1969년 Minsky는 Hidden Layer를 사용해 xor 문제를 풀 수 있음을 증명했는데 그 당시,
가중치를 구할 수 없다는 문제에 직면해 실용화는 어려웠다.

AND

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

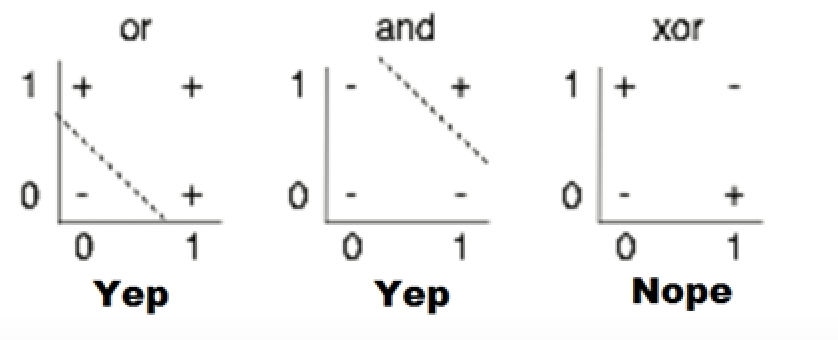
OR

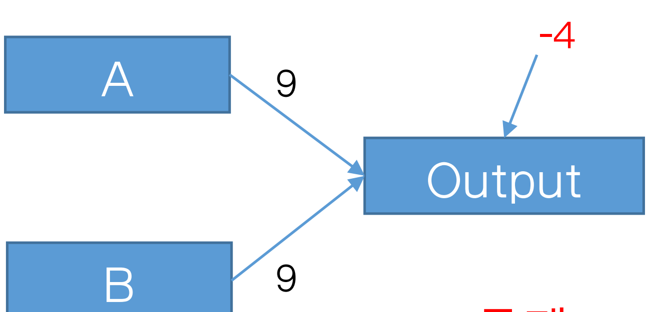
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

XOR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

exclusive-OR





$$Y = S(w_1x_1 + w_2x_2 + b) > 0.5$$

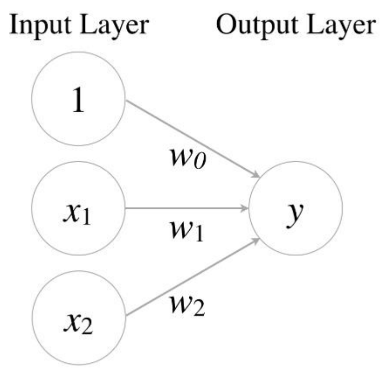
$$(w_1x_1 + w_2x_2 + b) > 0$$

DIY 문제: OR 문제가 해결됨을 보이세요~

1

Neural Network

1958년 Rosenblatt는 AND/OR 문제를 해결하는 Perceptron 기계를 개발하였다.



$$y = \begin{cases} 0 & (w_0 + w_1x_1 + w_2x_2 \leqq 0) \\ 1 & (w_0 + w_1x_1 + w_2x_2 > 0) \end{cases}$$

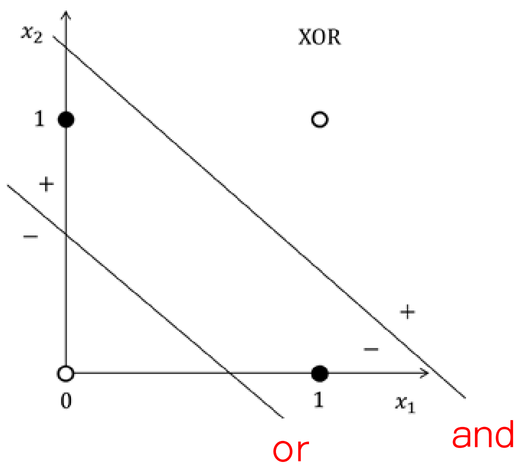
위에서 가중치 w를 구해보자.
아래 식에서 t는 true 값이고 f(net)은 네트워크를 통해 계산된 y값이다. t-f(net) 은 오차다.
w는 아래의 식으로 t-f(net) 의 오차가 양수면 f(net) 이 커져야 하므로 가중치에 일정량을 더하고 음수면 f(net) 이 작아져야 하므로 가중치에 일정량을 뺀다.
여기서, 그리스문자 에타는 상수로 학습율(Learning Rate) 라고 부른다.
학습률은 w값이 목적값으로 가는 속도를 조절하는 상수역할을 하는데 에타가 크면 빨리 해로 가지만 정확한 해를 구하기 어렵고 에타가 작으면 해로 느리게 가지만 정교한 해를 구할 수 있다.

$$w_i = w_i + \eta x_i (t - f(net))$$

DIY 문제: Perceptron을 구현하여 AND, OR 문제를 푸시오

1 Neural Network

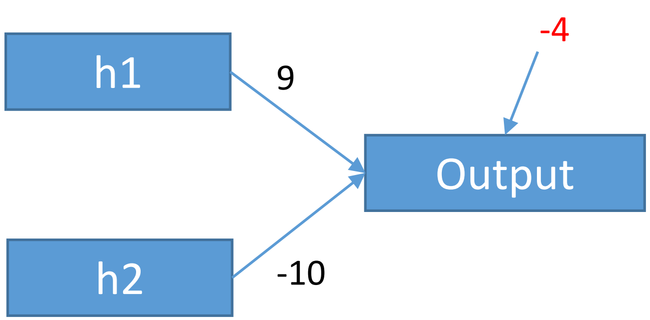
xor 문제는 아래의 그림처럼 h1, h2 두개 선을 사용하여 해결할 수 있다.

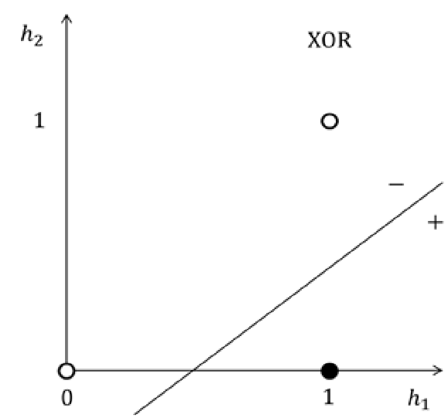


x_1	x_2	h_1	h_2	y
0	0	0(-)	0(-)	0
0	1	1(+)	0(-)	1
1	0	1(+)	0(-)	1
1	1	1(+)	1(+)	0

$$Y = S(w_1 h_1 + w_2 h_2 + b) > 0.5$$

$$(w_1 h_1 + w_2 h_2 + b) > 0$$





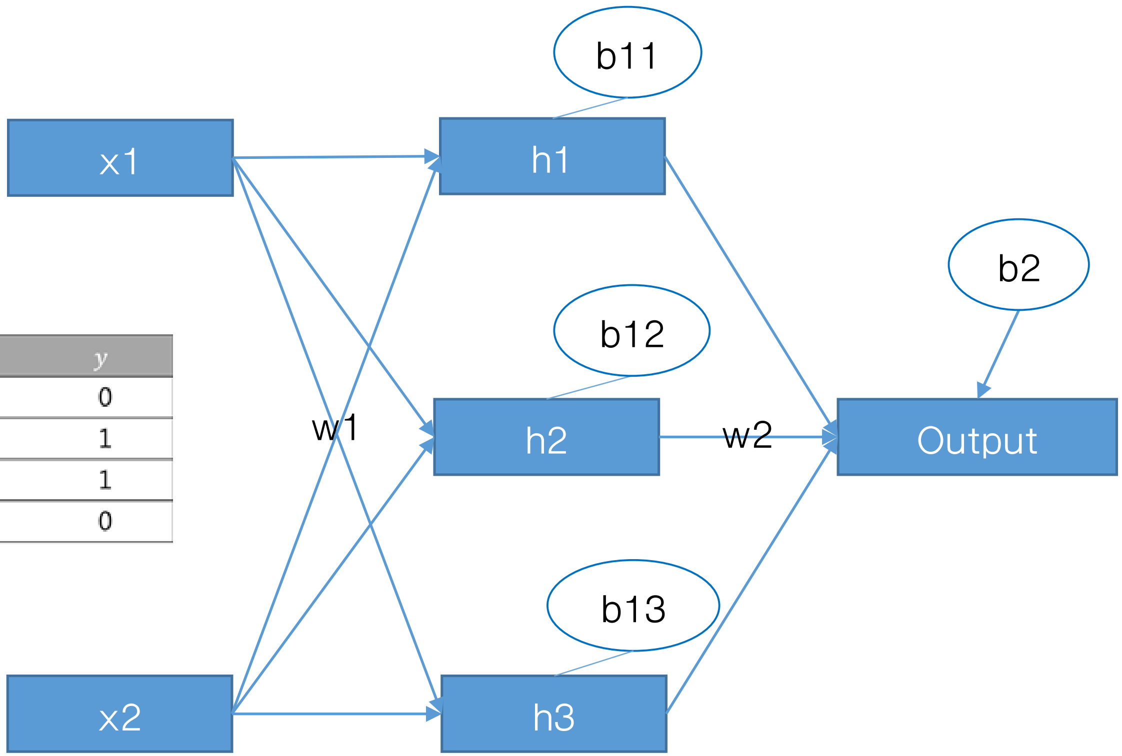
DIY 문제: XOR 문제가 해결됨을 보이세요~

1 Neural Network

신경망 모형 설계

- ✓ xor 문제는 입력이 두개이고 출력이 하나임
- ✓ 히든 레이어는 한 개, 노드는 3개로 가정함(히든 레이어와 노드 수는 자유롭게 결정 가능)
- ✓ w1은 총 6개의 선으로 2 by 3 행렬이고 w2는 3개의 선으로 3 by 1 행렬이다.
- ✓ b1은 1 by 3 행렬이고, b2는 1 by 1 행렬이다.
- ✓ H=x · w1+b1 이고, output=H · w2+b2로 계산한다.

x_1	x_2	h_1	h_2	y
0	0	0(-)	0(-)	0
0	1	1(+)	0(-)	1
1	0	1(+)	0(-)	1
1	1	1(+)	1(+)	0



1 Neural Network

신경망 모형 설계

- ✓ xor 문제는 입력이 두개이고 출력이 하나임
- ✓ 히든 레이어는 한 개, 노드는 3개로 가정함(히든 레이어와 노드 수는 자유롭게 결정 가능)
- ✓ w1은 총 6개의 선으로 2 by 3 행렬이고 w2는 3개의 선으로 3 by 1 행렬이다.
- ✓ b1은 1 by 3 행렬이고, b2는 1 by 1 행렬이다.
- ✓ H=x · w1+b1 이고, output= H · w2+b2로 계산한다.

DIY 문제: 이 신경망 모형을 그려보세요~~~

1 Neural Network

```
# xor.ipynb
import math
import numpy as np

def Sigmoid(x):
   return 1/(1+np.exp(-x))

x=np.array([[0,0], [0,1], [1,0], [1,1]])
w1=np.array([[-2,  5,  4], [ 3,  6,  3]])
b1=np.array([2, -2, -5])
w2=np.array([[-4], [ 8], [-8]])
h=Sigmoid(np.dot(x,w1)+b1)
y=Sigmoid(np.dot(h,w2))

print (y)
```

Hidden Layer가 추가됨으로써 xor문제가 풀림을 알 수 있다.
이제, xor문제에서 주어진 가중치 W, b를 구하는 것이 문제이다.
여기서, np.dot()는 행렬 곱셈을 수행하는 함수이다.

2 Backpropagation

가중 값은 Backpropagation으로 구할 수 있다. 아래와 같은 신경망을 가정하자.

여기서, $h = g(x \cdot w_1)$, $y = g(h \cdot w_2)$, t: Target Value 이고 $g(x) = \frac{1}{1+exp(-x)}$ 이다.

$$x \xrightarrow{w_1} h \xrightarrow{w_2} y, Cost = \frac{1}{2}(y-t)^2$$

우리의 목표는 Cost 함수를 최소화하는 가중값을 구하는 것이다. 가중값은 아래와 같이 Gradient Descent Method 를 반복적으로 사용하여 구한다. 여기서, lambda는 Learning Rate로 수렴속도를 조절하는 값이다.

$$w = w - \lambda \frac{\partial Cost(w)}{\partial w}$$

먼저 출력 값을 결정하는 w2 즉 뒤부터 풀어보자.(Backpropagation)

$$w_2 = w_2 - \lambda \frac{\partial Cost(w_2)}{\partial w_2}$$

$$\frac{\partial}{\partial w_2} Cost(w_2) = (y-t)\frac{\partial}{\partial w_2} y = (y-t)\frac{\partial}{\partial w_2} g(h \cdot w_2)$$

$$= t(h) \cdot (y-t) g(h \cdot w_2)(1 - g(h \cdot w_2)) = t(h) \cdot (y-t) y(1-y) = t(h) \cdot \delta_y$$

여기서, $\delta_y = (y-t)y(1-y)$ 으로 간단히 줄여 쓴 것이다. 최종적으로 가중값은 아래와 같이 업데이트할 수 있다.

$$w_2 = w_2 - \lambda \cdot t(h) \cdot \delta_y$$ t(x): Transpose of x

2

Backpropagation

다음은 w1 차례이다.

$$w_1 = w_1 - \lambda \frac{\partial Cost(w_1)}{\partial w_1}$$

$$\frac{\partial}{\partial w_1} Cost(w_1) = (y-t)\frac{\partial}{\partial w_1} y = (y-t)\frac{\partial}{\partial w_1} g(h \cdot w_2)$$

$$= (y-t) g(h \cdot w_2)(1 - g(h \cdot w_2)) \cdot t(w_2) \cdot \frac{\partial}{\partial w_1} h$$

$$= (y-t) y(1-y) \cdot t(w_2) \cdot \frac{\partial}{\partial w_1} g(x \cdot w_1 + b)$$

$$= t(x) \cdot [(y-t)y(1-y) \cdot t(w_2) \cdot g(x \cdot w_1 + b)(1 - g(x \cdot w_1 + b))]$$

$$= t(x) \cdot [(y-t)y(1-y) \cdot t(w_2) \cdot h(1-h)]$$

$$= t(x) \cdot \delta_h$$

여기서, $\delta_h = (y-t)y(1-y) \cdot t(w_2) \cdot h(1-h)$ 로 간단히 줄여쓰자.

$\delta_h = \delta_y \cdot t(w_2) \cdot h(1-h)$ 로 앞에서 구한 결과를 이용하여 재 표현이 가능하다.

이것이 Backpropagation 이다.

다음은 b에 대한 업데이트 식은 같은 방법으로 $b = b - \lambda \cdot \delta_h$ 이 된다.

2 Backpropagation

즉, Backpropagation은 앞서 구한 delta 값을 이용하여 다음의 delta 값을 계산하는 원리로 출력층에서 입력층으로 역으로 가중값을 보정 전파 할 수 있음을 보였다.

이를 이용하면 여러개의 Hidden Layer가 존재해도 쉽게 가중 값을 구할 수 있다.

아래와 같이 Hidden Layer가 3개 있다고 할 때, 역전파 알고리즘으로 가중값을 구하는 방법은 아래와 같다.

$$x \xrightarrow{w_1} h_1 \xrightarrow{w_2} h_2 \xrightarrow{w_3} h_3 \xrightarrow{w_4} y$$

$$\delta_1 \longleftarrow \delta_2 \longleftarrow \delta_3 \longleftarrow \delta_y$$

Backpropagation

$$\delta_y = (y - \mathbf{t}) \cdot y(1-y)$$
$$\delta_3 = \delta_y \cdot t(w_4) \cdot h_3(1-h_3)$$
$$\delta_2 = \delta_3 \cdot t(w_3) \cdot h_2(1-h_2)$$
$$\delta_1 = \delta_2 \cdot t(w_2) \cdot h_1(1-h_1)$$

$$w_4 = w_4 - t(h_3) \cdot \lambda \cdot \delta_y$$
$$w_3 = w_3 - t(h_2) \cdot \lambda \cdot \delta_3$$
$$w_2 = w_2 - t(h_1) \cdot \lambda \cdot \delta_2$$
$$w_1 = w_1 - t(x) \cdot \lambda \cdot \delta_1$$

2 Backpropagation

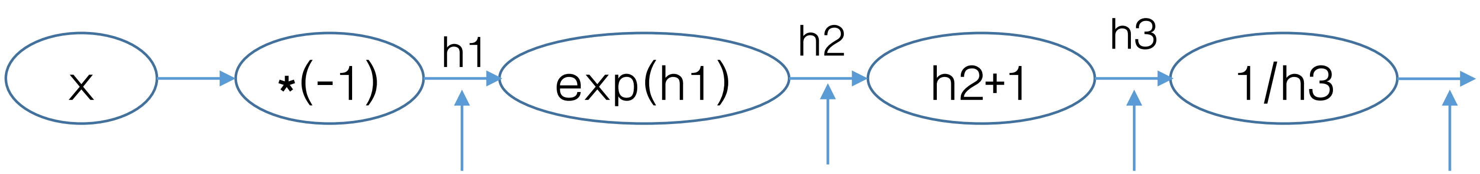
참고: Sigmoid 함수의 미분

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

$$f(x)' = -\frac{(1+exp(-x))'}{(1+exp(-x))^2} = \frac{exp(-x)}{(1+exp(-x))^2} = \frac{1}{1+exp(-x)} \cdot \frac{exp(-x)}{1+exp(-x)}$$

$$= f(x)\frac{1+exp(-x)-1}{1+exp(-x)} = f(x)\left(1 - \frac{1}{1+exp(-x)}\right) = f(x)(1-f(x))$$

Tensorflow 개념을 이용한 미분



$$\frac{\partial h_1}{\partial x} = -1 \qquad \frac{\partial h2}{\partial h_1} = exp(h_1) \qquad \frac{\partial h3}{\partial h_2} = 1 \qquad \frac{\partial f}{\partial h_3} = -\frac{1}{h_3^2}$$

$$\frac{\partial f}{\partial x} = \frac{\partial h_1}{\partial x} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial f}{\partial h_3} = (-1) \cdot exp(h_1) \cdot 1 \cdot (-\frac{1}{h_3^2}) = \frac{exp(-x)}{(1+exp(-x))^2} = f(x)(1-f(x))$$

2 Backpropagation

우측 프로그램은 히든 노드 3개를 가진

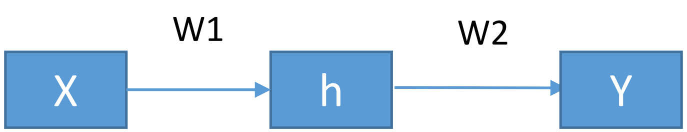
Backpropagation 알고리즘이다.

초기 가중치는 난수로 부여하고, 1,000회

반복하여 결과를 얻을 수 있다.

히든노드는 2개 이상이어야 xor 문제가 풀린다.

np.multiply()는 elementwise 곱을 수행한다.



Y의 추정값

```
0.02710406
0.98069245
0.97965474
0.01673378
```

```python
# xor1.ipynb
import numpy as np
def Sigmoid(x):
   return 1/(1+np.exp(-x))

lamda = 1
x=np.array([[0,0],[0,1],[1,0],[1,1]])
t=np.array([[0],[1],[1],[0]])

w1=2*np.random.rand(2,3)-1
b1=2*np.random.rand(1,3)-1
w2=2*np.random.rand(3,1)-1

for i in range(0,1000):
   h=Sigmoid(np.dot(x,w1)+b1)
   y=Sigmoid(np.dot(h,w2))
   deltaY= np.multiply(y-t,np.multiply(y,(1-y)))
   temp = np.multiply(w2.transpose(),np.multiply(h,(1-h)))
   deltaH = deltaY * temp
   w2=w2-np.dot(h.transpose(),lamda*deltaY)
   w1=w1-np.dot(x.transpose(),lamda*deltaH)
   b1=b1-lamda*deltaH
print (y)
```

2

Backpropagation

XOR 룰을 Tensorflow로 코딩한 결과이다. 학습해보자.

cost 함수는 cross entropy 함수이고, 로지스틱 회귀와 마찬가지로 Gradient Descent Algorithm을 사용하여 진행한다. optimizer='sgd' 라고 하면 learning rate를 지정할 수 없어서

sgd = tf.keras.optimizers.SGD(learning_rate=0.1) 와 같이 sgd 변수를 만들고 진행하자.

```
# XOR_NN_TF2.ipynb
import tensorflow as tf
import numpy as np
x = np.array([[0,0],[0,1],[1,0],[1,1]]).astype('float32')
y = np.array([[0],[1],[1],[0]]).astype('float32')
from tensorflow.keras import layers
model = tf.keras.Sequential()
model.add(layers.Dense(3, activation='sigmoid', input_dim=2))
model.add(layers.Dense(1, activation='sigmoid'))
sgd = tf.keras.optimizers.SGD(learning_rate=0.1)
#model.compile(optimizer='sgd', loss='binary_crossentropy',metrics=['accuracy'])
model.compile(optimizer=sgd, loss='binary_crossentropy',metrics=['accuracy'])
model.fit(x, y, epochs=10000, batch_size=4, verbose=0)
model.evaluate(x, y)
predicted = model.predict(x)
print(predicted)
```

감사합니다

인하대학교
데이터사이언스 학과
김 승 환

swkim4610@inha.ac.kr