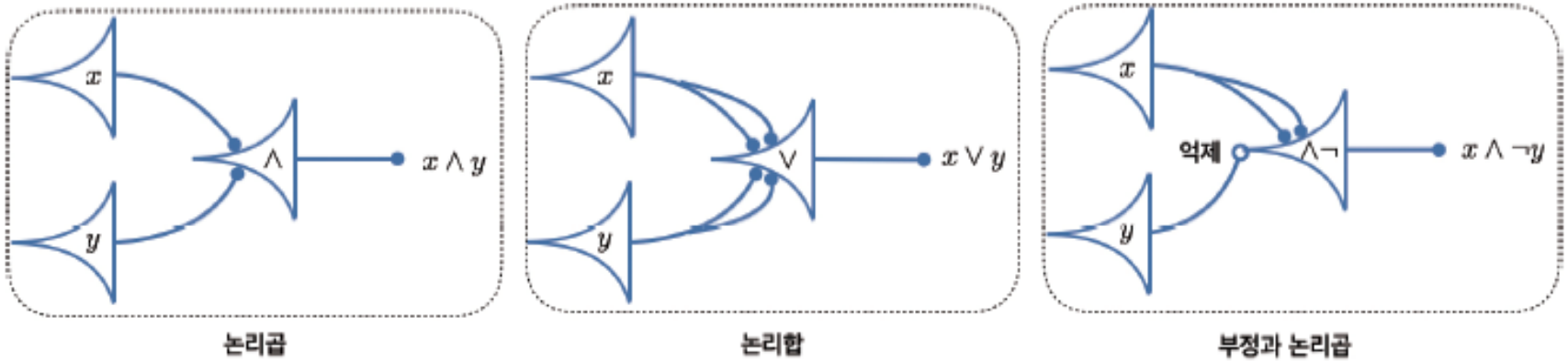
인공 신경망 기초
– 퍼셉트론의 등장과 한계

이장에서 배울 것들

- 연결주의가 무엇이고 어떤 목표를 갖고 있는지 살펴 보자.
- 각광받던 신경망 모델들이 관심 밖으로 밀려난 이유는 무엇일까.
- 신경망을 복잡하게 구성하는 일은 다시 어떤 벽을 만나게 되었을까.
- 오차를 이용하여 신경망의 가중치를 개선하기 위한 획기적인 방법은 무엇일까.

뇌의 동작을 흉내 내자 – 연결주의자의 목표

- 뇌는 뉴런neuron이라는 수없이 많은 신경세포들의 연결을 갖고 있음
- 인공 신경망은 이러한 신경세포의 동작을 흉내 내는 장치나 소프트웨어를 만들어 뇌가 수행하는 인지나 사고 능력을 갖춘 기계를 만들려는 노력
  - 이러한 방식의 연구를 통해 인공지능을 구현하려는 방식을 연결주의connectionism
- 이와 관련된 가장 선구적인 연구는 1940년대에 워렌 맥컬록Warren McCulloch과 월터 피츠Walter Pitts가 발표한 논문
- 두 저자는 아래와 같은 모양의 인공 신경세포 연결로 논리 연산이 가능하다는 것을 보임
- 삼각형은 신경세포neuron이며, 이들이 활성화되면 선으로 표현된 연결의 끝 지점에 있는 시냅스synapse가 이 신호를 다른 신경세포로 전달 이때 파란색을 가진 끝점은 1의 신호가 전달되는 것을 의미
- 흰색 원으로 표시된 끝점은 억제inhibitory 시냅스로서 연결된 신경세포가 활성화되지 못하게 막는 역할



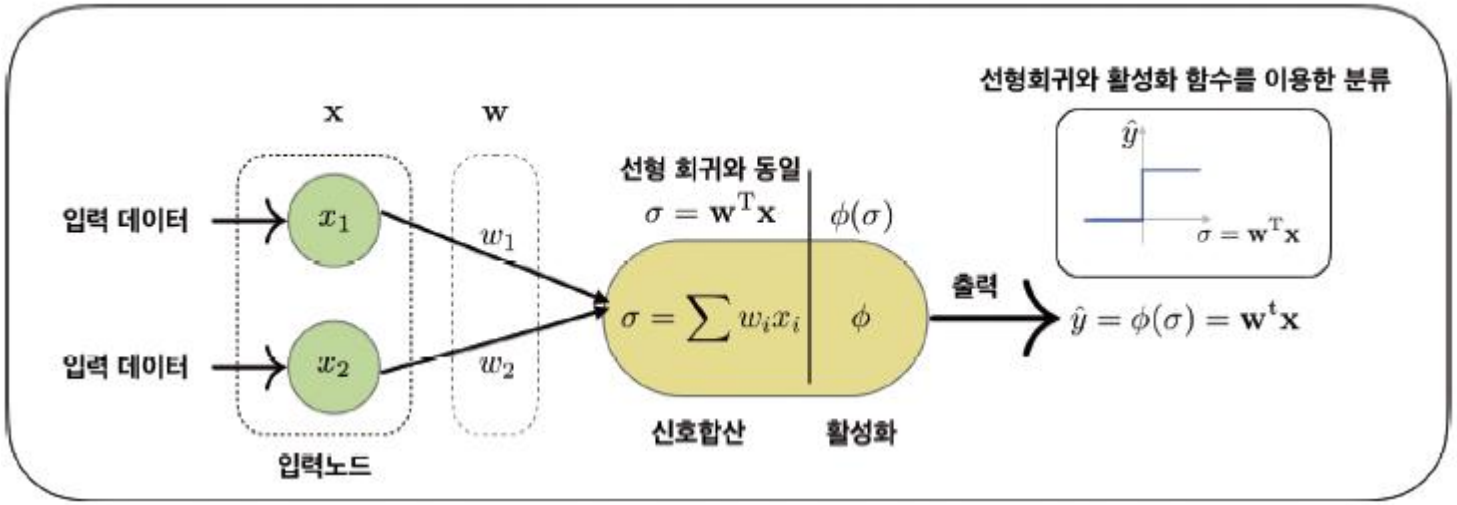
논리곱 논리합 부정과 논리곱

맥컬록 피츠 모델의 의미와 한계

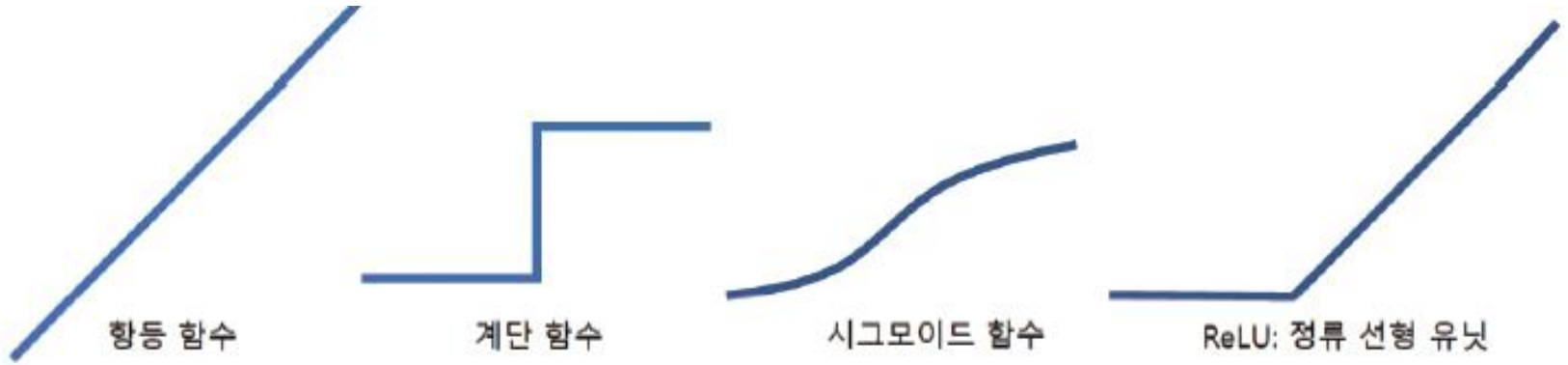
- 신호의 전달이 논리 연산이라면, 그것이 바로 계산
  - 맥컬록과 피츠의 선구적 연구는 신경세포의 신호 전달 방식을 모방함으로써 계산을 수행할 수 있음을 보여준 것
- 맥컬록과 피츠의 신경세포 모델은 "학습"이 불가능
  - 이 모델의 신경세포는 설계된 동작만을 수행할 수 있음

학습할 수 있는 신경 모델 – 퍼셉트론

- 학습이 가능하게 만든 것이 **퍼셉트론**perceptron
- 1958년에 **프랭크 로젠블랫**Frank Rosenblatt이 만든 퍼셉트론은 미국 해군에 의해 최초로 공개되어, 당시 인공지능의 새로운 장을 연 기술로 각광
- 가장 중요한 변화는 그림 왼쪽의 입력을 받아들이는 신경세포 x_i들이 출력을 수행하는 노드로 연결될 때 **연결강도**weight w_i에 의해 조정되어 전달된다는 것
- 출력 노드는 두 가지 일을 하는데, 우선 전달되어 오는 신호를 모두 합한다.
- 합산된 신호에 따라 출력을 결정하는 **활성화**activation 함수가 최종 출력을 발생



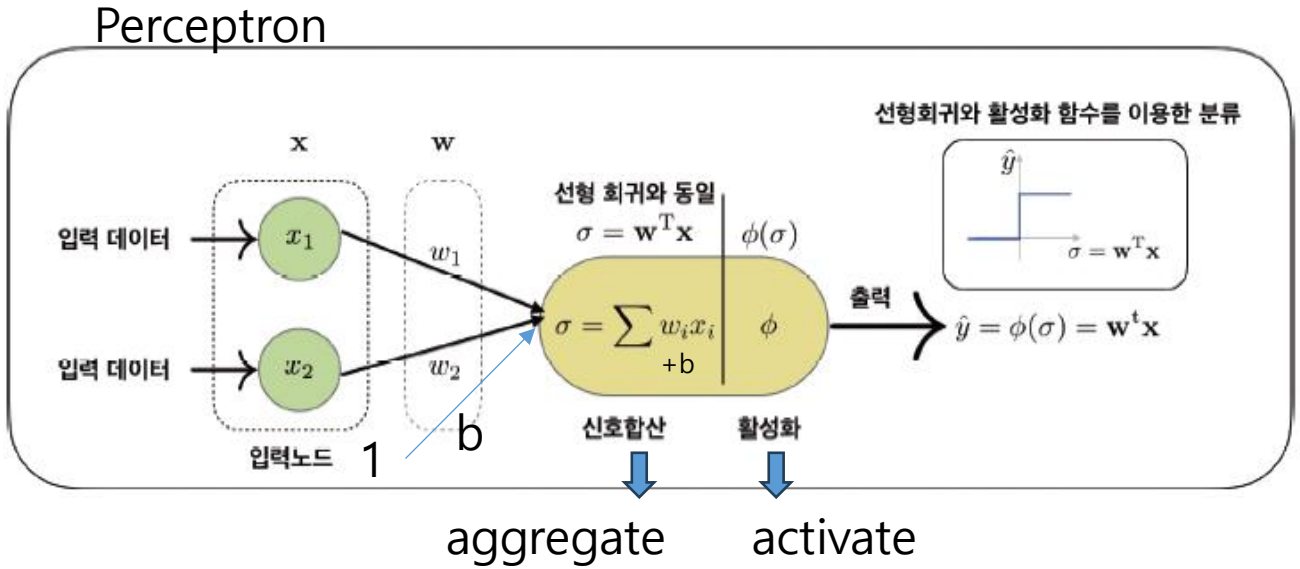
- 학습은 경험을 통해 기능이나, 성능이 바뀌는 것
  - 이를 위해서는 학습 과정을 통해 바뀌는 부분이 모델 내에 존재해야 함
  - 퍼셉트론은 학습을 통해 연결강도를 바꾸어 나감
  - 출력의 결과와 정답을 비교하면 오차를 알 수 있으며 이를 줄이도록 연결강도를 바꾸는 과정 = 학습
- 활성화 함수는 출력 노드에 모아진 신호를 다음으로 내어 보낼때 얼마나 강하게 보낼지를 결정하는 함수
  - 계단 함수는 최초의 퍼셉트론이 채택한 활성화 함수 미분이 되는 구간에서 미분치가 언제나 0이기 때문에 경사 하강법을 통한 최적화가 불가능
  - 이러한 이유로 신경망 분야에서 새롭게 도입되어 오랫동안 사용되던 활성화 함수가 시그모이드sigmoid



- 신경망이 동작을 변경하는 방법은 연결강도를 조정하는 것
- 인공 신경망의 학습이라는 것은 출력을 목표치와 비교하여 오차를 계산하고 이 오차를 줄이는 방향으로 연결강도를 변경하는 일을 의미
  - 신경망 모델에서는 이 연결강도가 바로 모델의 동작을 결정하는 **파라미터**parameter
  - 오차를 계산하는 방법은 다양하며, 이 오차를 줄이는 방향으로 연결강도를 조정하는 일이 신경망의 **최적화**optimization, 즉 학습
  - 학습과정을 조절하는 **하이퍼파라미터**hyperparameter로는 오차에 따른 연결강도 조정 정도, 최적화 방법, 학습 반복 횟수 등이 있음

- **퍼셉트론 만들어 보기**
  - **학습 기능 없이 구현해 보기**

```
# parameters
W, b = np.array([0.5, 0.5]), -0.7


def perceptron(x1, x2):
    summed = aggregate(x1, x2)
    return activate(summed)

def aggregate(x1, x2):
    X = np.array([x1, x2])
    return X.dot(W)+b

def activate(val) :
    if val >0 : return 1
    else : return -1
```



- **만들어 본 퍼셉트론 테스트해 보기**

```
for x in [[-1, -1], [-1, 1], [1, -1], [1, 1]] :
    print(perceptron(x[0], x[1]))
```

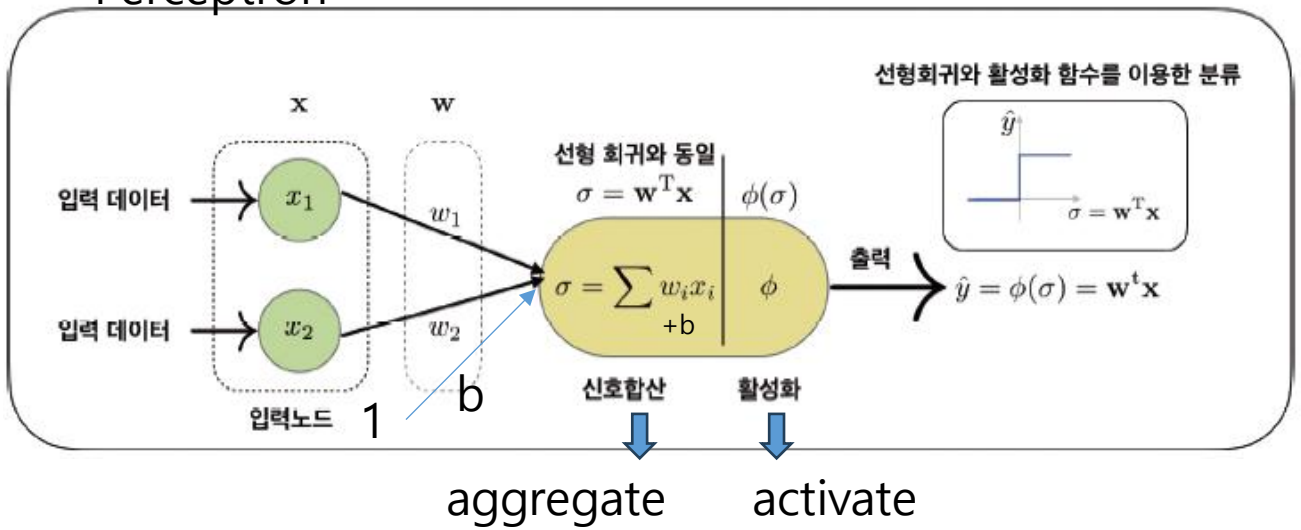
```
-1
-1
-1
 1
```

X1	X2	result
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

AND 게이트의 동작을 수행하는 퍼셉트론

```
# parameters
W, b = np.array([0.5, 0.5]), -0.7
```

Perceptron



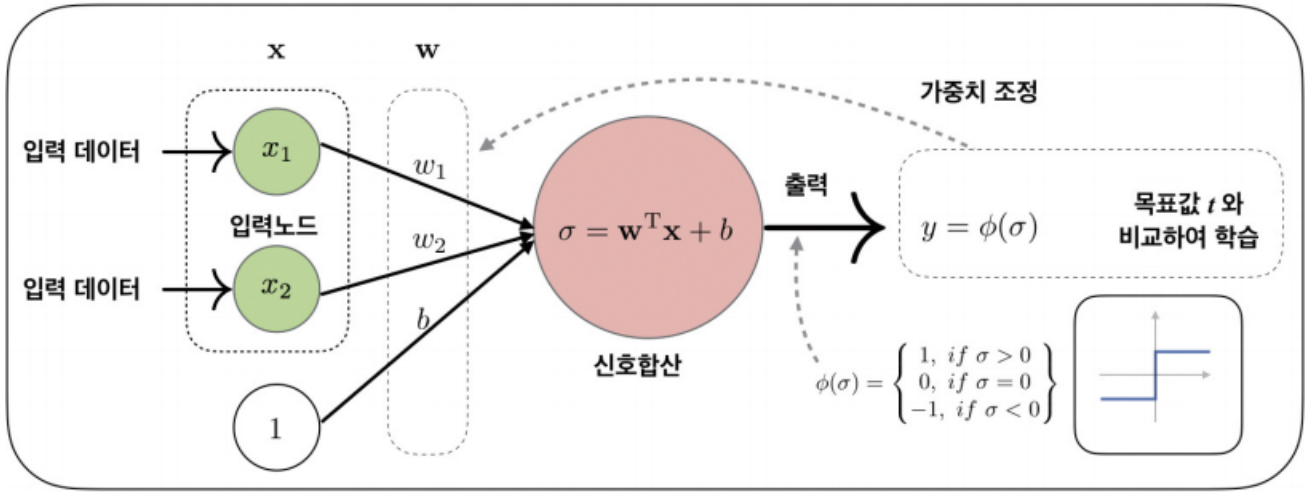
LAB 7-1 AND / OR 연산을 수행하는 퍼셉트론

실습 목표

퍼셉트론 모델을 이용하여 참(1)과 거짓(-1)의 값을 갖는 두 입력에 대해 논리곱 AND 연산과, 논리합 OR 연산을 수행할 수 있는 퍼셉트론을 만들어 보자.

힌트

퍼셉트론 모델은 아래와 같이 두 개의 입력에 대해 $\mathbf{w}$ 벡터가 곱해지고 편향 b가 더한 결과를 활성화 함수를 통과하게 만들면 된다. 거짓과 참을 표현할 때 0,1을 사용하는 방법도 있고, -1, 1을 사용하는 방법도 있는데, 신호를 크게 분리하기 위해 우선은 -1과 1을 사용하자. 활성화 함수 $\phi(\ )$는 입력이 0보다 크면 1, 그렇지 않으면 -1을 반환한다.



OR – perceptron

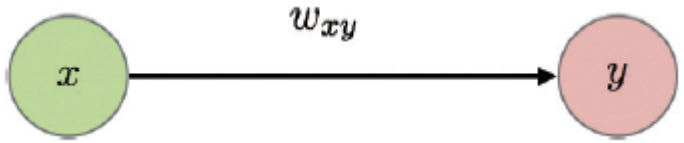
```
# parameters
W, b = np.array([0.1, 0.1]), 0.1

for x in [[-1, -1], [-1, 1], [1, -1], [1, 1]] :
    print(perceptron(x[0], x[1]))
```

```
-1
 1
 1
 1
```

X1	X2	result
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

학습의 원리 – 연결강도의 변경

- 중요한 것은 어떻게 연결강도를 변경할 것인지를 아는 것이며 이와 관련해 가장 중요한 토대를 만든 이가 도널드 헵Donald Hebb
  - 헵은 "신경세포 A의 축색돌기가 다른 신경세포 B를 자극하기에 충분할 정도로 가까이 있으면서 이 세포를 활성화시키는 일에 반복적이거나 지속적으로 참여한다면, 두 세포 가운데 하나나 둘 모두에서 성장이나 대사 변화가 발생하여 B를 활성화시키는 세포로서 A가 가진 효능이 증가한다"라고 함
  - 이것을 압축한 것이 바로 그 유명한 "함께 활성화되는 세포는 함께 연결된다"라는 표현
- 헵의 발견은 두뇌 신경망의 기능이 고정되어 있지 않고 바뀔 수 있는 가소성plasticity을 가지고 있음을 설명하는 것이며, 이러한 변화가 바로 뇌가 인지 능력을 향상시키는 학습 과정



- 두 신경세포가 함께 활성화는 정도는 두 신경세포가 내는 신호의 곱이고, 함께 연결된다는 것은 연결의 강도 w_{xy}가 커진다는 것이다. 이것은 결국 다음과 같은 수식으로 표현

$$w_{xy} = xy$$

Fire together, wire together!

- 두 신경세포가 가진 값을 m번 관찰했고, k번째 관측값이 $x^{(k)}$와 $y^{(k)}$라고 하면, 매번 관측에서 얻어진 가중치를 평균하면 될 것

$$w_{xy} = \frac{1}{m}\sum_{k=1}^{m} x^{(k)} y^{(k)}$$

- 연결의 가중치, 즉 연결강도를 변경할 수 있다는 것은 신경세포의 연결망을 학습시킬 수 있다는 것
- 헵의 학습 법칙Hebbian learning rule은 신경세포에 가해지는 입력과 출력에 따라 새로운 가중치를 다음과 같이 바꾸어 나간다는 것

$$w_{xy}^{(k+1)} = w_{xy}^{(k)} + x^{(k+1)} y^{(k+1)}$$

생물학적 모델: 입력과 출력이 같이 활성화하면 둘의 연결을 강화한다
Fire together, wire together!

비지도 학습

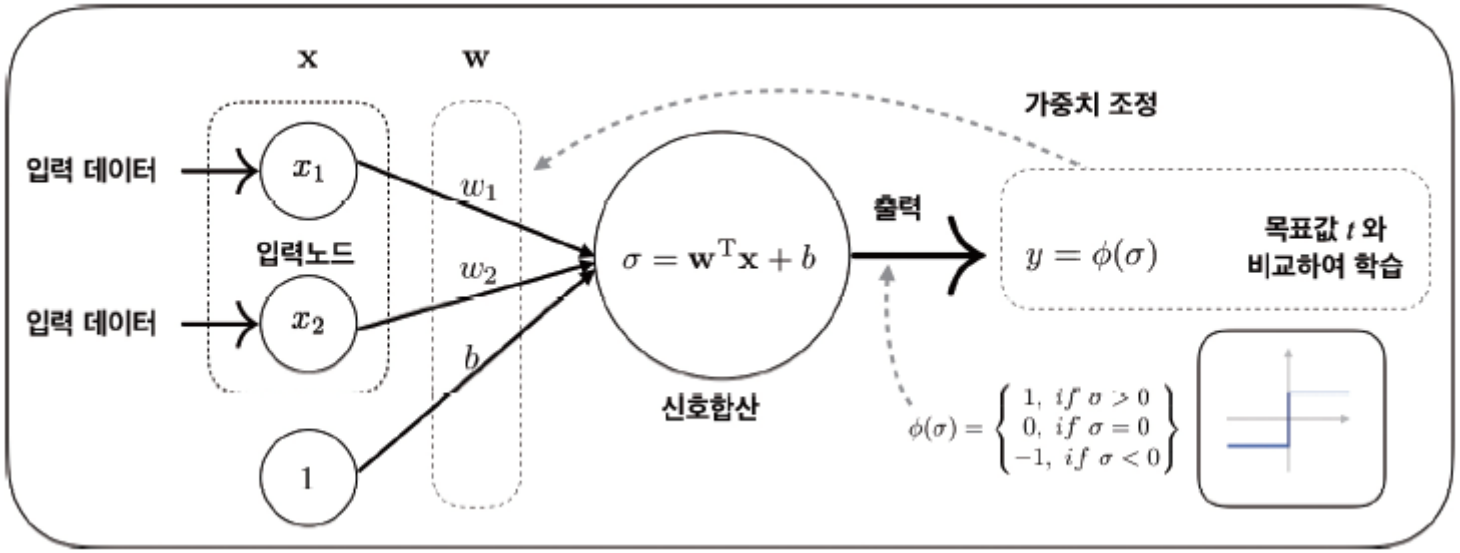
- 퍼셉트론의 학습은 이 헵의 학습 법칙에 근거를 두고 있음
  - 두 신경세포의 활성화가 아니라, 목표치를 제시하고 가중치가 목표치를 발생시키도록 만들어 가는 것
  - 헵의 학습규칙에서 출력에 해당하는 부분을 목표값 t로 변경
  - 가중치의 갱신도 학습률learning rate η를 통해 조금씩 이루어지게 만듦

$$w_{xy}^{(k+1)} = w_{xy}^{(k)} + \eta x^{(k+1)} t^{(k+1)}$$

퍼셉트론 학습: 입력에 대해 출력이 같은 방향으로 반응해야 하면 둘의 관계를 강화한다.
If they have to fire together, wire them tighter!

지도 학습

- 이항 불리언 연산binary boolearn operation이 가능하도록 하는 퍼셉트론은 아래 그림처럼 만들 수 있음
- 참 또는 거짓을 갖는 입력은 x_1과 x_2에 제공
- 이 신호는 가중치가 곱해져 출력 노드로 전달
- 이때 신호의 편향bias이 이루어지도록 하는 가중치 b가 추가
- 편향은 신호값을 양의 방향이나 음의 방향으로 이동시키는 역할



논리합을 수행하는 퍼셉트론 만들기

실습 목표

퍼셉트론 모델을 이용하여 참(1)과 거짓(–1)의 값을 갖는 두 입력에 대해 논리합을 수행할 수 있도록 모델을 학습시켜 보라.

힌트 퍼셉트론 모델은 입력 노드 둘을 담은 벡터 X와 이 입력에 곱해질 가중치 벡터 W를 준비하고, 편향 b를 더해주면 된다. 출력은 W.dot(X) + b의 값을 활성화 함수에 넣어 구한다.

논리합 – 퍼셉트론 구현

```
import numpy as np

W = np.array([0, 0, 0])
learning_rate = 0.01

def activate(val) :
    if val > 0 : return 1
    elif val < 0 : return  -1
    else: return 0

def perceptron(x1, x2) :
    X = np.array([x1, x2, 1])
    return activate(W.dot(X))
```

임의의 파라미터로 퍼셉트론 만들어 보기

먹통 퍼셉트론

X1	X2	result
-1	-1	0
-1	1	0
1	-1	0
1	1	0

동작의 결과 `perceptron(-1, -1), perceptron(-1, 1), perceptron(1, -1), perceptron(1,1)`

(0, 0, 0, 0)

논리합 – 학습 모델

```
def train(x1, x2, target) :
    # 변경하는 값 : W
    global W
    X = np.array([x1, x2, 1])
    y = perceptron(x1, x2)

    # 정답을 맞추었으면, 이 입력에 대한 학습을 수행하지 않음
    if y == target : return False
    # 정답이 아닌 경우, 학습 실시
    W = W + learning_rate * (X * target)
    return True
```

X1	X2	정답
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

```
train(-1, -1, -1)
train(-1,  1,  1)
train( 1, -1,  1)
train( 1,  1,  1)
```

← 지도학습 교재

학습 수행 – 확인

```
adjusted = False
for i in range(100):
    print('training iteration: ', i)
    if train(-1, -1, -1): adjusted = True
    if train(-1,  1,  1): adjusted = True
    if train( 1, -1,  1): adjusted = True
    if train( 1,  1,  1): adjusted = True
    if not adjusted: break;
    adjusted = False
```

```
training iteration: 0
training iteration: 1
```

공부 끝!

X1	X2	result
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

동작의 결과 `perceptron(-1, -1), perceptron(-1, 1), perceptron(1, -1), perceptron(1,1)`

(-1, 1, 1, 1)

```
W
array([0.01, 0.01, 0.01])
```

다양한 논리 연산이 가능하게 퍼셉트론 훈련하기

실습 목표

LAB[7-2]를 이용하여 다양한 논리 연산이 가능하도록 퍼셉트론을 학습시켜 보자.

힌트 학습을 실시하는 train 함수를 이용하여 필요한 논리 연산의 입력과 출력을 제공하고 훈련을 실시하면 다양한 논리 연산을 구현할 수 있다.

NAND – 학습 모델

```
def train(x1, x2, target) :
    # 변경하는 값 : W
    global W
    X = np.array([x1, x2, 1])
    y = perceptron(x1, x2)

    # 정답을 맞추었으면, 이 입력에 대한 학습을 수행하지 않음
    if y == target : return False
    # 정답이 아닌 경우, 학습 실시
    W = W + learning_rate * (X * target)
    return True
```

X1	X2	정답
-1	-1	1
-1	1	1
1	-1	1
1	1	-1

```
train(-1, -1,  1)
train(-1,  1,  1)
train( 1, -1,  1)
train( 1,  1, -1)
```

← 지도학습 교재

NAND 학습 수행 – 확인

```
adjusted = False
for i in range(100):
    print('training iteration: ', i)
    if train(-1, -1,  1): adjusted = True
    if train(-1,  1,  1): adjusted = True
    if train( 1, -1,  1): adjusted = True
    if train( 1,  1, -1): adjusted = True
    if not adjusted: break;
    adjusted = False
```

```
training iteration: 0
training iteration: 1
```

공부 끝!

X1	X2	result
-1	-1	1
-1	1	1
1	-1	1
1	1	-1

동작의 결과 `perceptron(-1, -1), perceptron(-1, 1), perceptron(1, -1), perceptron(1,1)`

(1, 1, 1, -1)

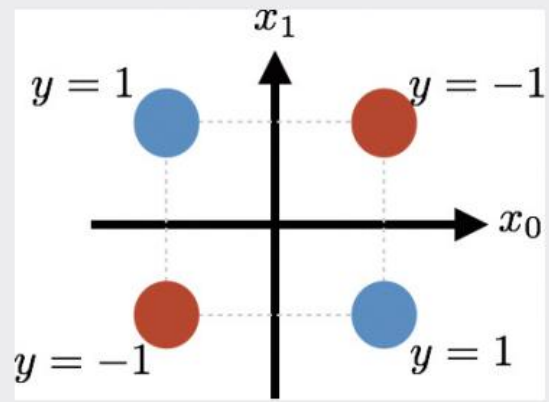
```
[67] W

array([-0.01, -0.01,  0.01])
```

도전문제 7.1 XOR의 진리표를 이용하여 퍼셉트론을 학습시켜 보자

XOR는 아래와 같은 기호로 표시된다. 진리표가 그 아래 나타나 있다. 이런 논리 연산을 수행하는 퍼셉트론을 만들 수 있을까? 만들어지지 않는다면 그 이유를 생각해 보라.

$$y = (x_0 \wedge \neg x_1) \vee (\neg x_0 \wedge x_1)$$

x_0	x_1	y
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



퍼셉트론, 연결주의가 누린 첫 영예와 긴 좌절

- 지금은 인공지능 이야기를 할 때 퍼셉트론과 같은 연결주의적 모델이 주인공이지만, 당시에는 "인공지능"이라는 용어 자체가 기호주의자들이 만들어낸 것
- 기호주의자 중의 핵심적 인물이었던 마빈 민스키는 로젠블랫과 종종 학회에서 맹렬한 논쟁을 벌였는데, 토어스튼 조아킴즈Thorsten Joachims는 이들의 입장을 이렇게 요약함

> *"로젠블랫은 볼 수 있고, 언어를 이해하는 컴퓨터를 만들 수 있다는 비전을 가지고 있었는데, 마빈 민스키는 **기능이 너무 단순해서 그런 일은 벌어지지 않을 것**이라 지적했다."*

XOR 학습

```
adjusted = False
for i in range(100):
    print('training iteration: ', i)
    if train(-1, -1, -1): adjusted = True
    if train(-1,  1,  1): adjusted = True
    if train( 1, -1,  1): adjusted = True
    if train( 1,  1, -1): adjusted = True
    if not adjusted: break;
    adjusted = False
```

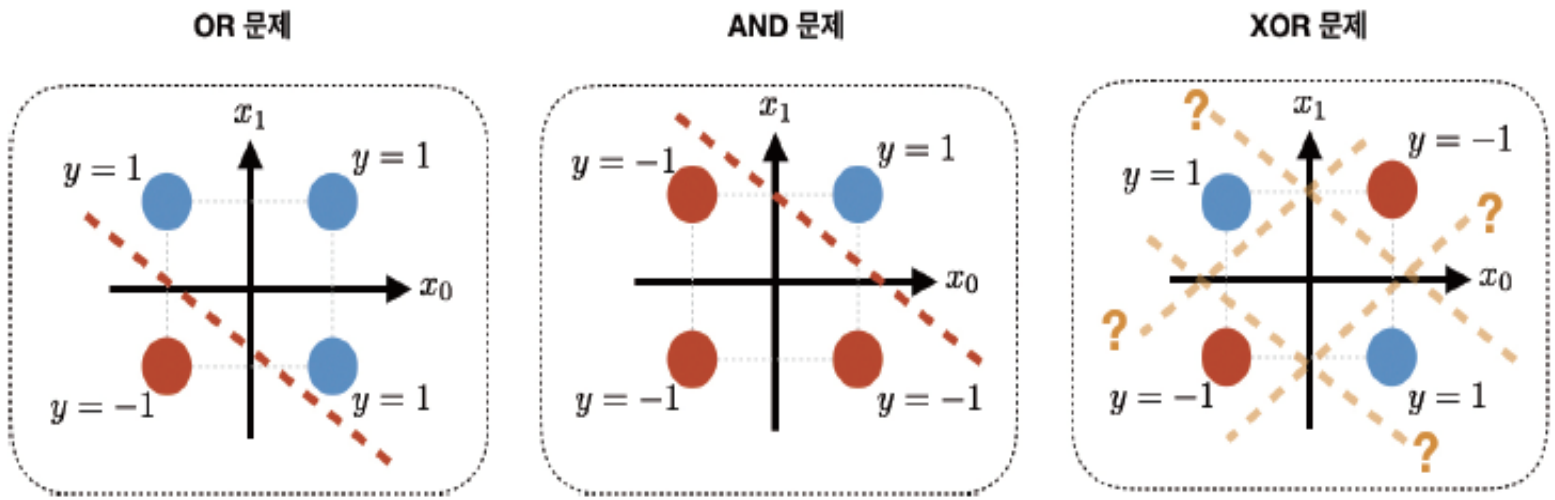
```
training iteration:  42
training iteration:  43
training iteration:  44
training iteration:  45
training iteration:  46
training iteration:  47
training iteration:  48
training iteration:  49
training iteration:  50
training iteration:  51
training iteration:  52
training iteration:  53
training iteration:  54
training iteration:  55
training iteration:  56
training iteration:  57
training iteration:  58
training iteration:  59
training iteration:  60
training iteration:  61
training iteration:  62
training iteration:  63
training iteration:  64
training iteration:  65
training iteration:  66
training iteration:  67
training iteration:  68
training iteration:  69
training iteration:  70
training iteration:  71
training iteration:  72
training iteration:  73
training iteration:  74
training iteration:  75
training iteration:  76
training iteration:  77
training iteration:  78
training iteration:  79
training iteration:  80
training iteration:  81
training iteration:  82
training iteration:  83
training iteration:  84
training iteration:  85
training iteration:  86
training iteration:  87
training iteration:  88
training iteration:  89
training iteration:  90
training iteration:  91
training iteration:  92
training iteration:  93
training iteration:  94
training iteration:  95
training iteration:  96
training iteration:  97
training iteration:  98
training iteration:  99
```

```
for x in [[-1, -1], [-1, 1], [1, -1], [1, 1]]:
    y = perceptron(x[0], x[1])
    print(x[0], x[1], y)
```

```
-1 -1 1
-1 1 1
1 -1 1
1 1 -1
```

X1	X2	result	정답
-1	-1	1	-1
-1	1	1	1
1	-1	1	1
1	1	-1	-1

- 기호주의자들은 로젠블랫이 과학적 기준을 따르지 않고, 언론을 편파적으로 활용해 "과장된 주장overclaim"을 한다고 봄
- 마빈 민스키와 시모어 패퍼트가 쓴 "퍼셉트론"이라는 저서는 당시 엄청난 기대를 받던 신경망의 한계를 밝혀내기 위해 온 힘을 기울여 쓴 책
    - 가장 잘 알려진문제는 퍼셉트론이 XOR 연산을 구현하지 못 한다는 것

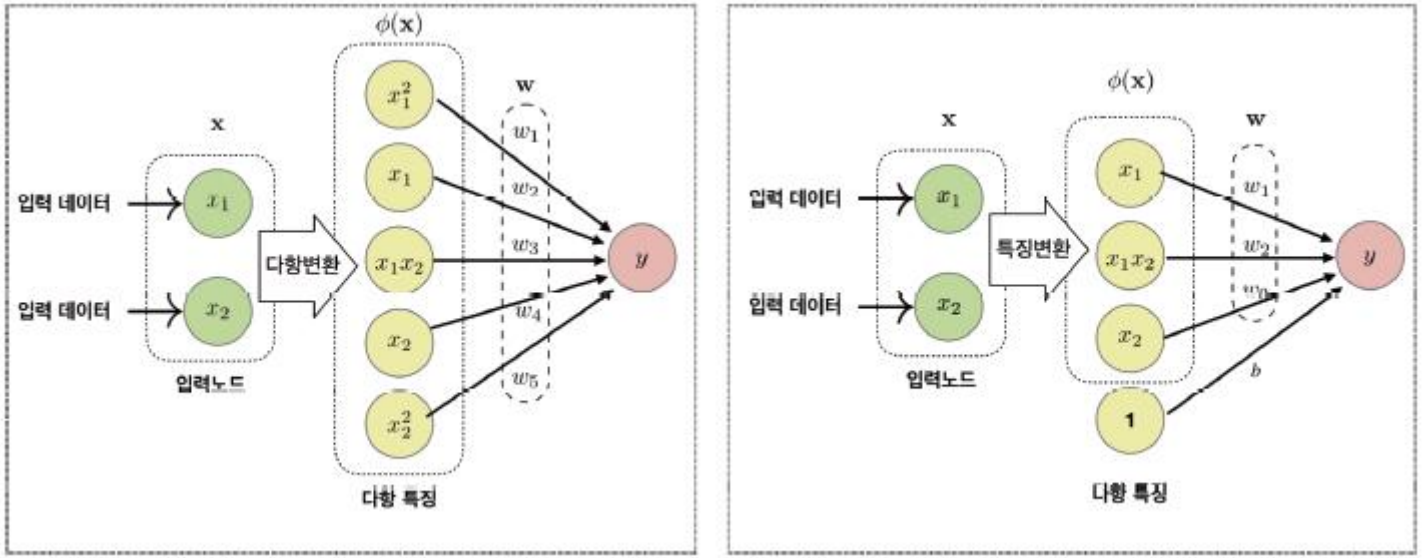


퍼셉트론이 XOR 문제를 풀 수 있게 만드는 방법

- 입력 노드가 출력으로 바로 연결되어 있는 단순한 퍼셉트론은 선형 분리만 가능하지만, 이미 맥컬록과 피츠의 신경 모델에서도 신호는 여러 층을 거쳐 전달되는 모델이 사용됨
- 퍼셉트론의 구조를 복잡하게 만들어 다양한 일을 하게 할 수 있지 않을까? 선형 함수로서 동작하는 모델을 비선형 함수로 바꾸는 방법 중에 우리가 이미 살펴본 것은 특징 벡터를 다항화하는 방법
- 단순한 퍼셉트론 모델에서는 x_1 와 x_2 의 입력을 가지므로, 이를 2차 다항화하면 다음과 같은 특징 벡터를 얻을 수 있을 것

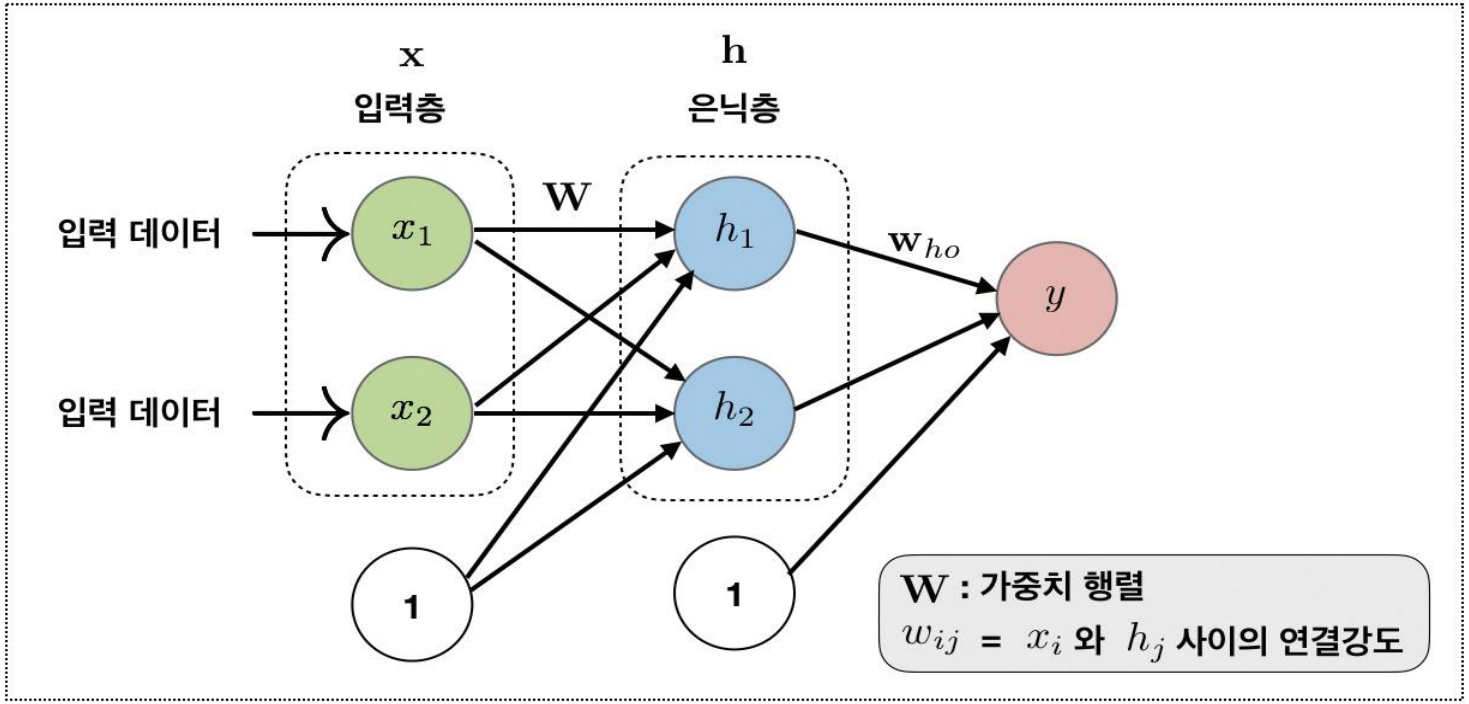
$$\left(x_1^2, x_1, x_1 x_2, x_2^2\right)$$

- 두 개의 입력을 이렇게 다항화하고 이를 퍼셉트론으로 인지하는 모델은 아래의 왼쪽과 같은 퍼셉트론으로 설명

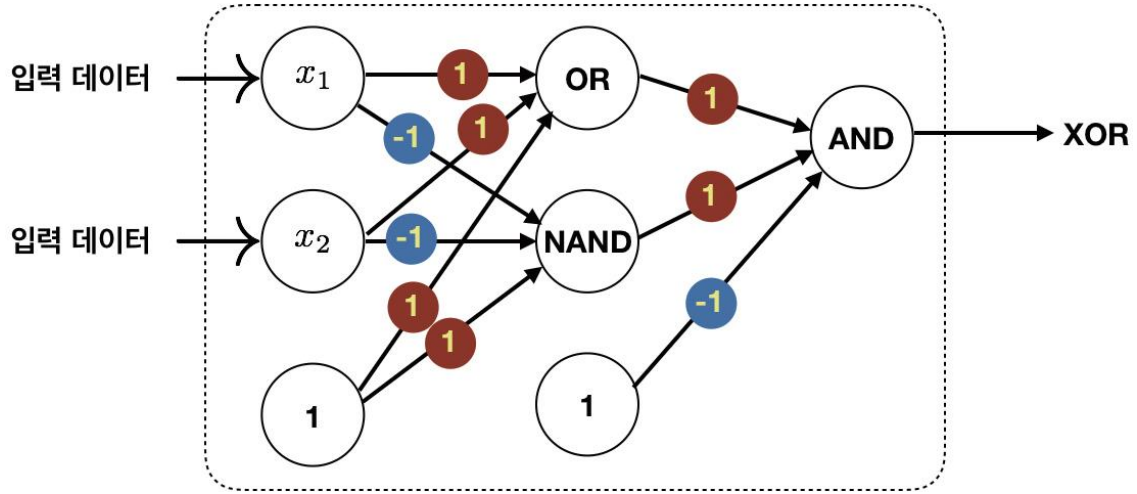


- 원래의 입력 x_1, x_2를 다항 변환하여 새로운 입력 $\phi(\mathrm{x})$로 만드는 과정은 신호를 전달하는 것이 아니라 직접적인 계산이 이루어짐
- 입력과 출력을 바로 연결하지 않고 중간에 신호를 중계하는 신경세포들을 두는 것
  - 입력과 출력 사이에 존재하는 이런 중간 노드들의 계층을 은닉층hidden layer

- 중간에 층을 두고 연결하면 다음과 같은 구조의 **다층 퍼셉트론**multi-layer perceptron을 만들 수 있을 것
- 가중치는 입력층에 있는 n개의 노드가 발생시키는 신호를 은닉층에 있는 m개의 노드로 연결하는 가중치 행렬 W에 의해 전달
- 은닉층의 노드들이 발생시키는 신호는 또 다른 연결강도 벡터 w_{ho}에 의해 출력으로 연결



- 이런 구조의 퍼셉트론은 더 많은 수의 연결이 더 많은 단계로 연결되어 복잡한 동작을 할 수 있음
- 적절한 연결강도를 부여하면 단순한 모델이 할 수 없었던 XOR 문제 풀이도 가능



- 이 방법은 다층 퍼셉트론에서는 최종 은닉층과 출력의 연결에만 적용
- 층을 쌓아 만든 퍼셉트론이 분명 다양한 기능을 수행할 수 있지만, 오류를 줄이고 원하는 기능을 학습할 수 있도록 하는 방법이 없다면 이 모델을 훈련시킬 수 없음

입력 다항화로 XOR를 해결해 보기

실습 목표

참 또는 거짓의 값을 갖는 두 개의 입력에 대해 XOR 연산을 수행할 수 있도록, 입력에 대한 다항 변환을 실시하고, 이를 통해 얻은 다항 특징에 대해 퍼셉트론 학습을 통해 XOR 연산을 수행하는 모델을 구현하자.

힌트

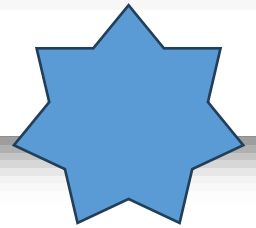
두 입력 x_1와 x_2에 다항 변환을 적용하면 $(x_1^2, x_2^2, x_1x_2, x_1, x_2)$의 다항 특징을 얻는다. 이 가운데 x_1^2과 x_2^2은 무조건 1이므로 $(x_1, x_2, x_1x_2, 1)$에 대해 퍼셉트론 학습을 적용하자.

퍼셉트론 모델

```
W_poly = np.array([1, -1, 1, 1])
learning_rate = 0.8

def activate(val) :
    if val > 0 : return 1
    elif val < 0 : return  -1
    else: return 0

def perceptron_poly(x1, x2) :
    X_poly = np.array([x1, x2, x1*x2, 1])
    return activate(W_poly.dot(X_poly))

def train(x1, x2, target) :
    global W_poly
    X_poly = np.array([x1, x2, x1*x2, 1])
    y = perceptron_poly(x1, x2)

    if target == y : return False

    #### 학습 실시
    W_poly = W_poly + learning_rate * (X_poly * target)
    return True
```

```
adjusted = False
for i in range(2000):
    print('training iteration: ', i)
    if train(-1, -1, -1): adjusted = True
    if train(-1,  1,  1): adjusted = True
    if train( 1, -1,  1): adjusted = True
    if train( 1,  1, -1): adjusted = True
    if not adjusted: break;
    adjusted = False;
```

```
training iteration:  0
training iteration:  1
```

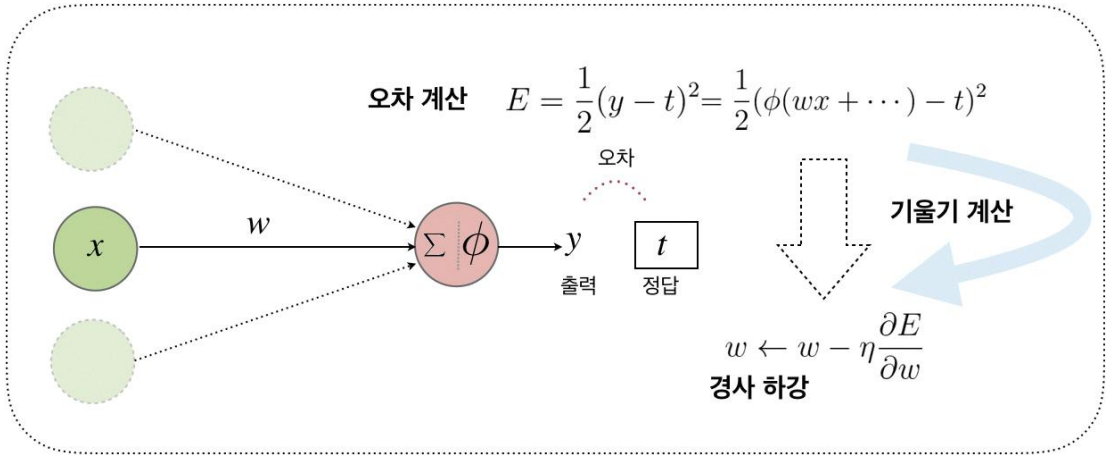


```
W_poly
```

```
array([ 0.2, -0.2, -1.4,  0.2])
```

새로운 퍼셉트론의 학습 – 오차를 줄이도록 연결강도를 고치자

- 인공 신경망에서 연결강도를 조정해 오차를 줄여 보자
- 오차 곡면의 기울기를 연결강도에 대해 구한 뒤에 이 기울기를 따라 내려가는 **경사 하강법**gradient descent을 사용하여 구현할 수 있음
- x가 연결강도 w로 출력 노드에 연결되어 있다고 하자.
- 출력 노드는 x를 비롯한 여러 노드에서 신호를 수신하여 합산하는 부분과 이 값을 활성화함수에 넣어 최종 출력을 결정하는 부분으로 나뉨
- 출력과 정답(목표값)의 차이를 제곱하여 오차를 구하는 일반적인 방법을 사용 가능
- 출력 노드의 출력값 y가 $\phi(wx+\cdots)$이므로, 오차E는 $1/2\,(\phi(wx+\cdots)-t)^2$



- 오차를 연결강도 w에 대해 편미분

$$\frac{\partial E}{\partial w} = \frac{1}{2}\frac{\partial}{\partial w}(\phi(wx+\cdots)-t)^2$$

$$= (\phi(wx+\cdots)-t)\cdot \frac{\partial}{\partial w}\phi(wx+\cdots)$$

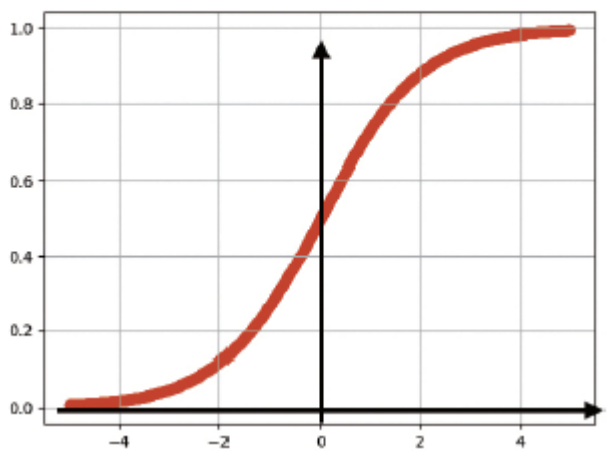
$$= (\phi(wx+\cdots)-t)\cdot \phi'(wx+\cdots)\cdot \frac{\partial}{\partial w}(wx+\cdots)$$

$$= (y-t)\cdot \boxed{\phi'(wx+\cdots)}\cdot x$$

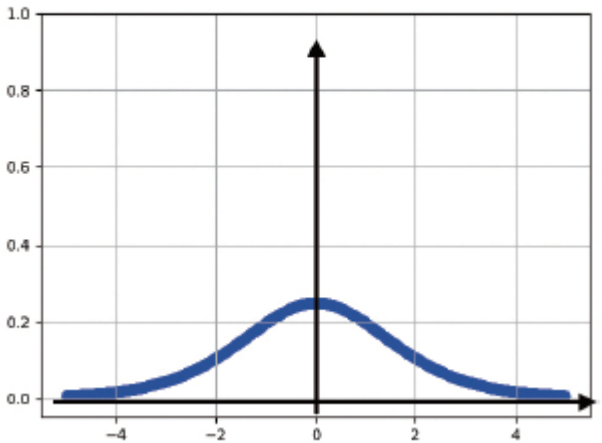
계단 함수는 미분이 불가능

- 기울기를 이용하여 연결강도를 수정
- 활성화 함수 $\phi(\,)$의 미분 $\phi'(\,)$ 만 안다면 입력 x, 출력 y, 목표값 t를 이용하여 간단히 계산할 수 있음
- 계단함수는 미분이 불가능한 지점이 있고, 미분이 되는 곳에서도 미분치가 언제나 0
  - 이 방법을 위해서는 미분이 가능한 새로운 활성화 함수가 필요
  - 신경망에서는 다음과 같은 **시그모이드**sigmoid 혹은 **로지스틱**logistic 함수를 활성화 함수로 도입

$$\phi(x) = \frac{1}{1+e^{-x}}$$



$$\phi'(x) = \frac{1}{1+e^{-x}}\left(1-\frac{1}{1+e^{-x}}\right)$$



- 이 함수는 미분이 가능하다는 장점과 함께, 이 함수의 미분은 다음과 같이 원래의 함수를 이용하여 표현할 수 있는 장점이 있음

$$\phi'(x) = \phi(x)(1-\phi(x))$$

- 조금 전 구했던 오차의 기울기에서 활성화 함수의 미분으로 표시된 부분을 출력값 y로 표현할 수 있음

$$\begin{aligned}\frac{\partial E}{\partial w} &= (y-t)\cdot\phi'(wx+\cdots)\cdot x \\ &= (y-t)\cdot\phi(wx+\cdots)\cdot(1-\phi(wx+\cdots))\cdot x \\ &= (y-t)\cdot y\cdot(1-y)\cdot x\end{aligned}$$

- 출력값과 목표값만 알면 출력의 오차를 줄이는 연결강도 변경을 아래의 수식을 사용하여 수행할 수 있음

$$w \leftarrow w - \eta \cdot \partial E / \partial w$$

- 여기서 출력값 y와 목표값 t만 알면 연결강도 수정에 필요한 $\delta = (y - t)\phi'(wx + \cdots)$ 가 $\delta = (y - t) \cdot y \cdot (1 - y)$를 통해 쉽게 계산되며 δ를 이용하여 연결강도를 더 좋은 상태로 바꿀 수 있음

$$w \leftarrow w - \eta \cdot \delta \cdot x$$

- 신경망의 각 연결강도에 적용하여 오차를 줄일 수 있다면, 복잡한 신경망도 학습이 가능하지 않을까???
  - 이것이 퍼셉트론의 한계를 극복하는 돌파구가 될 수 있다
    - 오류 역전파 알고리즘의 등장

다층 퍼셉트론의 학습 – 오차를 아래로 전파하자

- 복잡한 문제를 해결하기 위해서는 일반적으로 신경망 연결을 여러 층으로 만들어야 하며 아래 그림과 같은 신경망이 연결된 구조에서 신호가 x를 거쳐 y로 전달되는데 이 방향을 **순전파**forward propagation라고 함
- 앞 절에서 오차를 줄이는 방향으로 연결강도를 변경하는 방법을 아래와 같이 출력 부분에서 확인할 수 있는 목표 t와출력 y의 차이, 그리고 출력의 미분 y'가 연결망을 거꾸로 타고 전달되어 가중치를 조정하는 것으로 봄