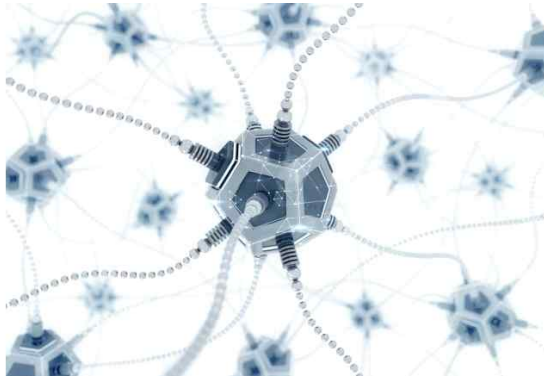


파이썬 인공지능 1

01 딥러닝의 이해

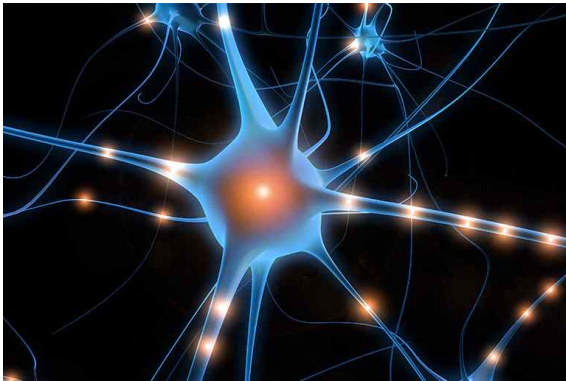


<https://www.discovermagazine.com/technology/we-almost-gave-up-on-building-artificial-brains>

인공 신경망은 딥러닝의 약진에 의해 최근 몇 년 동안 주목을 받아왔습니다. 그러면 인공 신경망은 무엇이고 어떻게 만들어졌을까요? 여기서는 인공 신경망의 바탕이 되는 실제 생체 신경의 구조와 구성 요소를 살펴보고 그것들이 어떻게 인공 신경의 구조와 구성요소에 대응이 되는지 살펴봅니다.

01 인공 신경망이란?

독자 여러분은 지금까지 왜 사람에게에는 아주 간단하지만 컴퓨터에게는 상상할 수 없을 정도로 어려운 일들이 있는지 궁금해 한 적이 있나요? 인공 신경망(ANN's : Artificial neural networks)은 인간의 중앙신경계로부터 영감을 얻어 만들어졌습니다. 생체 신경망과 같이 인공 신경망은 커다란 망으로 함께 연결되어 있는 인공 신경을 기반으로 구성됩니다. 개개의 인공 신경은 생체 신경과 같이 간단한 신호 처리를 할 수 있도록 구현되어 있습니다.



<https://www.allerin.com/blog/4-benefits-of-using-artificial-neural-nets>

인공 신경망으로 할 수 있는 일들

그러면 우리는 인공 신경망으로 무엇을 할 수 있을까요? 인공 신경망은 많은 문제 영역에 성공적으로 적용되어 왔습니다. 예를 들어 다음과 같은 문제들에 적용되었습니다.

- 패턴 인식에 의한 데이터 분류
그림에 있는 이것은 나무인가?
- 입력 데이터가 일반적인 패턴과 맞지 않았을 때의 이상 감지
트럭 운전사가 잠들 위험이 있는가?
이 지진은 일반적인 땅의 움직임인가 아니면 커다란 지진인가?
- 신호 처리
신호 거르기
신호 분리하기
신호 압축하기
- 예측과 예보에 유용한 목표 함수 접근
이 폭풍은 태풍으로 변할 것인가?

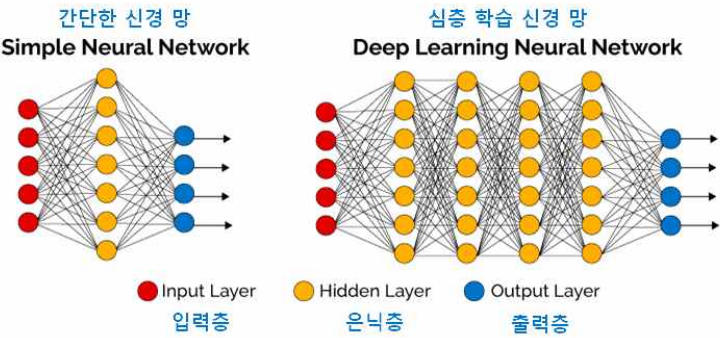
이런 문제들은 조금은 추상적으로 들립니다. 그래서 몇 가지 실제로 적용된 응용 예들을 보도록 합니다. 인공 신경망은 다음과 같은 것들을 할 수 있습니다.

- 얼굴 확인하기
- 음성 인식하기
- 손 글씨 읽기
- 문장 번역하기
- 게임 하기(보드 게임이나 카드 게임)
- 자동차나 로봇 제어하기
- 그리고 더 많은 것들!

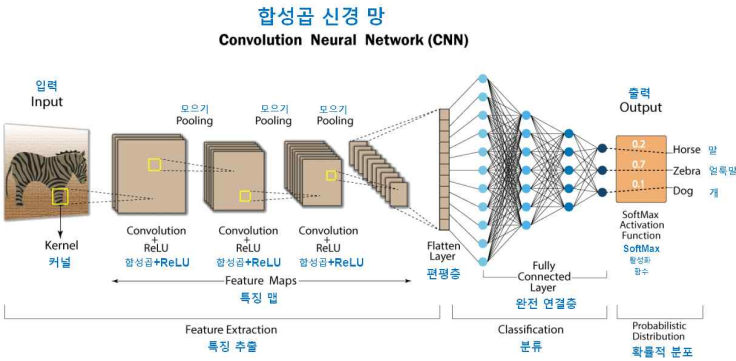
인공 신경망을 이용하면 세상에 있는 많은 문제들을 해결할 수 있습니다. 독자 여러분도 해결하고 싶은 문제가 있다면, 인공 신경망을 이용해 해결할 가능성이 있습니다. 인공 신경망을 통한 문제 해결은 이제 선택이 아닌 필수가 되어가고 있으며, 인공 신경망을 통한 문제 해결 능력은 여러분에게 더 많은 기회를 줄 것입니다.

인공 신경망의 구조

인공 신경망을 구성하는 방법은 다양합니다. 예를 들어 다음과 같은 형태로 인공 신경망을 구성할 수 있습니다. 다음 그림에서 노란색 노드로 표현된 은닉 층이 2층 이상일 때 심층 신경망(DNN)이라고 합니다.

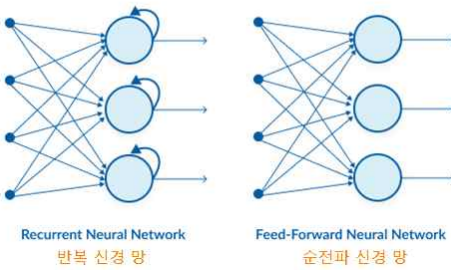


다음은 CNN 형태의 인공 신경망입니다. CNN은 이미지 인식에 뛰어난 인공 신경망으로 이미지의 특징을 뽑아내는 인공 신경망과 분류를 위한 인공 신경망으로 구성됩니다.

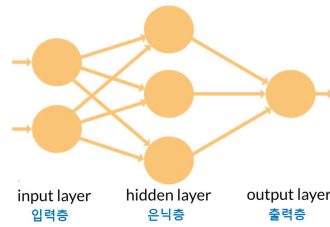


다음은 RNN 형태의 인공 신경망입니다. 아래 그림에서 왼쪽에 있는 그림은 RNN 형태의 신경망으로 노드에서 나온 값이 다시 되먹임 되는 형태로 인공 신경망이 구성됩니다. 오른쪽에 있는 그림은 한 방향으로만 신호가 흐르는 기본적인 인공 신경망입니다. RNN 형태의 인공 신경망은 문장 인식에 뛰어난 인공 신경망입니다.

반복 신경 망 구조
Recurrent Neural Network structure



인공 신경망은 구성 방법에 따라 동작 방식도 달라집니다. 가장 간단한 인공 신경망의 구조는 신호가 한 방향으로 흐르는 인공 신경망으로 다음과 같은 형태입니다.



일반적으로 인공 신경망은 3개의 층으로 구성됩니다. 각각 입력 층, 은닉 층, 출력 층이라고 합니다. 입력 층은 입력 신호를 받아서 다음 층에 있는 은닉 층으로 보냅니다. 은닉 층은 하나 이상 존재할 수 있습니다. 마지막에는 결과를 전달하는 출력 층이 옵니다.

02 인공 신경망의 학습 방법

전통적인 알고리즘들과는 달리 인공 신경망은 프로그래머의 의도대로 작업하도록 ‘프로그램 되거나’ 또는 ‘구성되거나’ 할 수 없습니다. 인간의 뇌처럼 인공 신경망은 하나의 일을 수행할 방법을 직접 배워야 합니다. 일반적으로 인공 신경망의 학습 방법에는 3가지 전략이 있습니다.

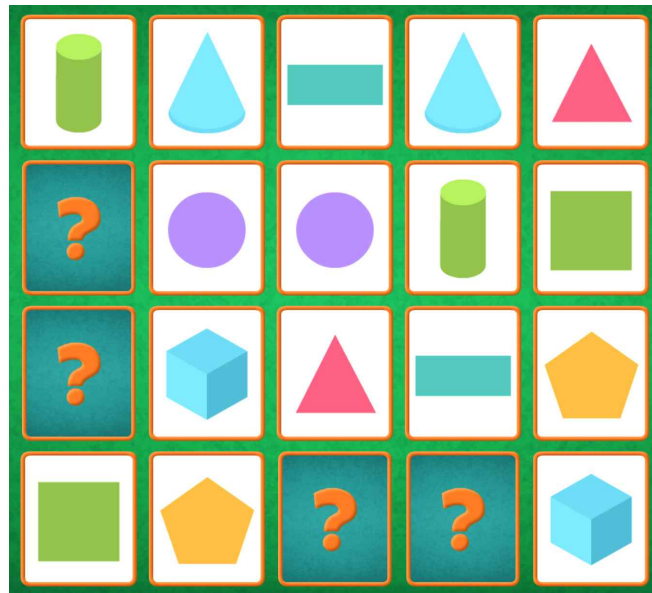
지도 학습

가장 간단한 학습 방법입니다. 미리 알려진 결과들이 있는 충분히 많은 데이터가 있을 때 사용하는 방법입니다. 지도 학습은 다음 순서로 진행됩니다. ❶ 하나의 입력 데이터를 처리합니다. ❷ 출력값을 미리 알려진 결과와 비교합니다. ❸ 인공 신경망을 수정합니다. ❹ 이 과정을 반복합니다. 이것이 지도 학습 방법입니다. 예를 들어 엄마가 어린 아이에게 그림판을 이용하여 사물을 학습시키는 방법은 지도 학습과 같습니다. 한글, 숫자 등에 대한 학습도 지도 학습의 형태입니다. 아래에 있는 그림판에는 동물, 과일, 채소 그림이 있고 해당 그림에 대한 이름이 있습니다. 아이에게 고양이를 가리키면서 ‘고양이’라고 알려주는 과정에서 아이는 학습을 하게 됩니다. 이와 같은 방식으로 인공 신경망도 학습을 시킬 수 있으며, 이런 방법을 지도 학습이라고 합니다.



비지도 학습

비지도 학습은 입력 값이 목표 값과 같을 때 사용하는 학습 방법입니다. 예를 들어, 메모리 카드 게임을 하는 방식을 생각해 봅시다. 메모리 카드 게임을 할 때 우리는 그림에 표현된 사물의 이름을 모르는 상태로 사물의 형태를 통째로 기억해야 합니다. 그리고 같은 그림을 찾아 내며 게임을 진행하게 됩니다. 이와 같이 입력 값과 출력 값이 같은 형태의 데이터를 학습할 때, 즉, 입력 값을 그대로 기억해 내야 하는 형태의 학습 방법을 비지도 학습이라고 합니다.



<https://www.curiousworld.com/blog/importance-playing-memory-games>

강화 학습

인공 신경망이 익숙하지 않은 환경에서 시행착오를 통해 이익이 되는 동작을 취할 확률은 높이고 손해가 되는 동작을 취할 확률은 낮추게 하는 학습 방법입니다. 즉, 이익이 되는 동작을 강화해가는 학습 방법입니다. 예를 들어, 우리가 익숙하지 않은 환경에서 어떤 동작을 취해야 하는지 모를 때, 일단 할 수 있는 동작을 취해보고 그 동작이 유리한지 불리한지를 체득하는 형태의 학습 방식과 같습니다. 이 과정에서 유리한 동작은 기억해서 점점 더 하게 되고 불리한 동작도 기억해서 점점 덜 하게 됩니다.



<https://www.unict.it/it/didattica/news/lauree-magistrali-giornate-di-orientamento>

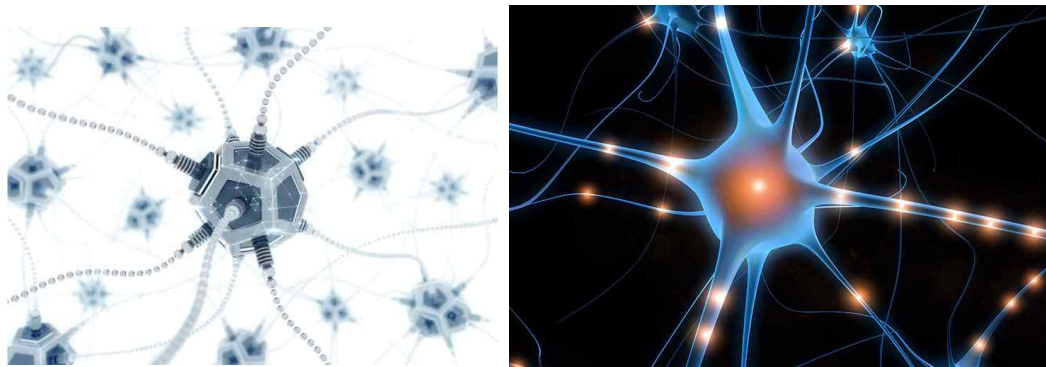
<https://safedownautobelay.com/benefits-of-getting-children-climbing/>

03 인공 신경 살펴보기

앞에서 우리는 인공 신경망에 대해 살펴보았습니다. 그러면 인공 신경망은 무엇으로 구성될까요? 여기서는 인공 신경망을 구성하는 인공 신경에 대해 생물학적 신경과 비교해 보면서 그 내부 구조를 살펴보도록 합니다.

인공 신경과 생물학적 신경

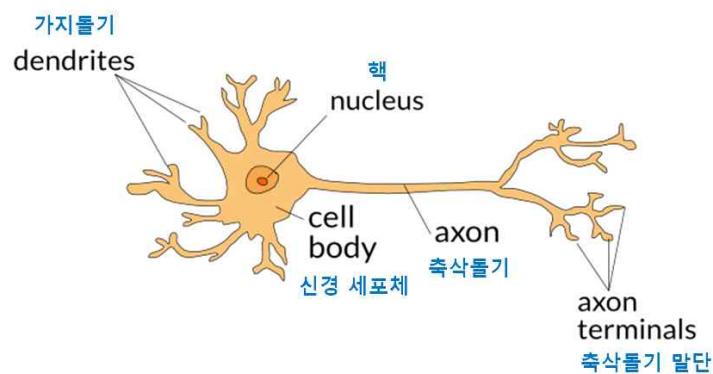
인공 신경망의 구성요소는 인공 신경입니다. 인공 신경이라는 이름은 생물학적 신경으로부터 얻어졌습니다. 인공 신경은 우리 두뇌의 구성 요소 중 하나인 생물학적 신경의 동작을 따라 만들어진 모형(model)입니다. 즉, 인공 신경은 생물학적 신경의 모형입니다.



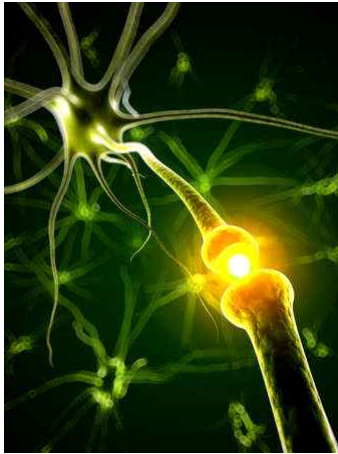
<https://www.discovermagazine.com/technology/we-almost-gave-up-on-building-artificial-brains>

<https://www.allerin.com/blog/4-benefits-of-using-artificial-neural-nets>

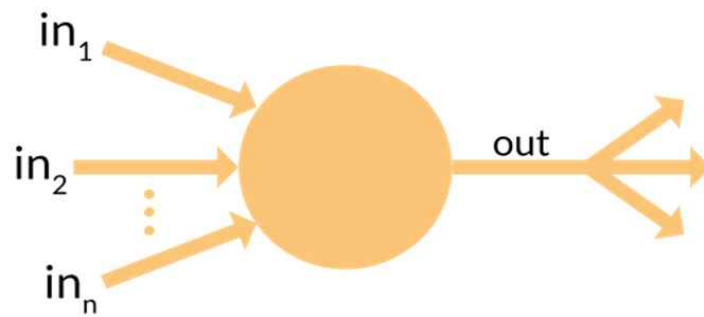
생물학적 신경은 신호를 받기 위한 여러 개의 가지 돌기(dendrites), 입력받은 신호를 처리하기 위한 신경 세포체(cell body), 다른 신경들로 신호를 내보내기 위한 축삭돌기(axon)와 축삭돌기 말단으로 구성됩니다.



특히 축삭돌기 말단과 다음 신경의 가지 돌기 사이의 틈을 시냅스라고 합니다.

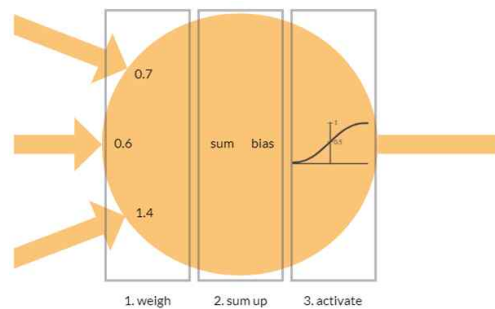


시냅스는 신경결합 부라고도 하며 한 신경에서 다른 신경으로 신호를 전달하는 연결지점을 말합니다. 인공 신경은 데이터를 받기 위한 여러 개의 입력 부, 입력받은 데이터를 처리하는 처리부, 그리고 여러 개의 다른 인공 신경들로 연결될 수 있는 하나의 출력부를 가집니다. 특히 인공 신경의 출력부에는 다음 인공 신경의 입력부에 맞는 형태의 데이터 변환을 위한 활성화함수가 있습니다.



인공 신경 내부 살펴보기

이제 인공 신경 안으로 들어가 봅시다. 어떻게 인공 신경은 입력을 처리할까요? 독자 여러분은 하나의 인공 신경 안에서 그 계산들이 실제로 얼마나 간단한지 알면 깜짝 놀랄 수도 있습니다. 인공 신경은 세 개의 처리 단계를 수행합니다.



❶ 각각의 입력 값은 가중치에 의해 커지거나 작아집니다.

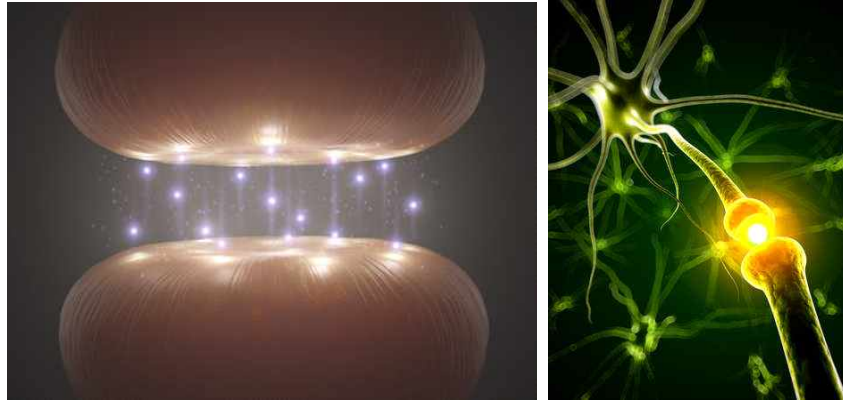
하나의 입력 신호(데이터)가 들어올 때 그 신호는 그 입력에 할당된 하나의 가중치(weight)에 의해 곱해 집니다. 예를 들어, 하나의 인공 신경이 그림과 같이 3 개의 입력을 가진다면, 그 인공 신경은 각 입력에 적용될 수 있는 3개의 가중치를 가집니다. 학습 과정에서 인공 신경망은 결과 값과 목표 값의 오차를 기반으로 가중치들의 크기를 조정합니다. 생물학적 신경의 가지 돌기가 그 두께에 따라 신호가 더 잘 전달되거나 덜 전달되는 것처럼 인공 신경의 가중치도 그 값에 따라 신호(데이터)가 커지거나 작아집니다. 가중치는 다른 말로 강도(strength)라고도 합니다. 즉, 가중치는 입력 신호가 전달되는 강도를 결정합니다. 입력 신호가 작더라도 가중치가 크면 신호가 커지며, 입력 신호가 크더라도 가중치가 작으면 내부로 전달되는 신호는 작아집니다. 인공 신경의 가중치는 생물학적 신경의 가지 돌기의 두께로 비유할 수 있습니다. 인공 신경의 학습과정은 입력값이 출력값에 맞도록 반복적으로 가중치 값의 크기를 조절하는 과정입니다.

❷ 모든 입력 신호들은 더해집니다.

가중치에 의해 곱해진 입력 신호들은 하나의 값으로 더해집니다. 그리고 추가적으로 보정 값(offset)도 하나 더해집니다. 이 보정 값은 편향(bias)이라고 불립니다. 인공 신경망은 학습 과정에서 편향도 조정합니다. 편향은 하나로 더해진 입력 신호에 더해지는 신호로 신호를 좀 더 크게 하거나 또는 좀 더 작게 하는 역할을 합니다. 즉, 신호를 조금 더 강화하거나 조금 더 약화하는 역할을 합니다. 편향은 가중치의 한 형태입니다.

❸ 신호를 활성화합니다.

앞에서 더해진 입력신호들은 활성화함수를 거쳐 하나의 출력 신호로 바뀝니다. 활성화 함수는 신호 전달 함수라고도 하며 신호의 형태를 다른 인공 신경의 입력에 맞게 변경하여 출력하는 역할을 합니다. 생물학적 신경을 시냅스가 연결하는 것처럼 활성화함수는 인공 신경을 연결하는 역할을 수행합니다.



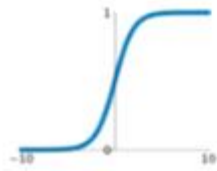
다음은 인공 신경망에 사용되는 활성화함수입니다. 활성화 함수는 인공 신경망의 활용 영역에 따라 달리 사용됩니다.

활성화 함수

Activation Functions

Sigmoid

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

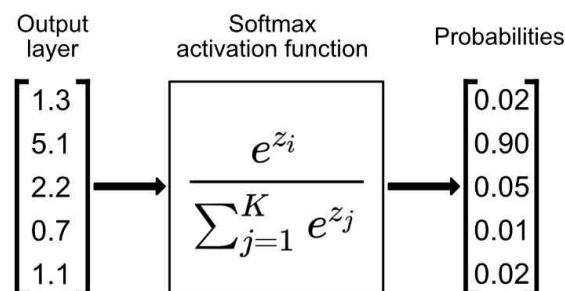


ReLU

$$\max(0, x)$$



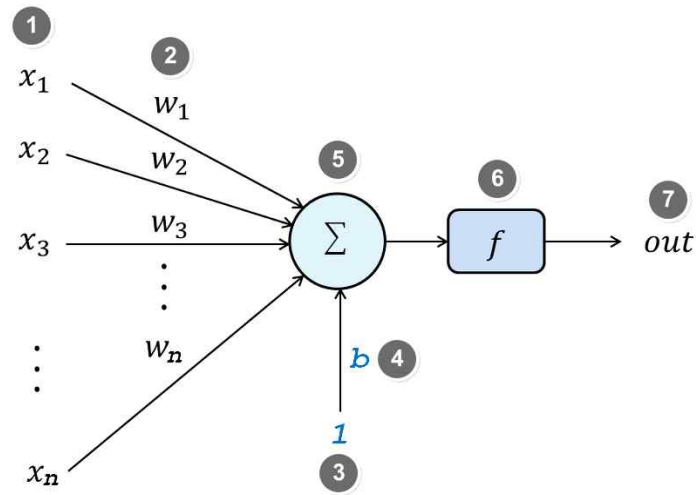
일반적으로 출력 값을 0에서 1사이의 값으로 하고자 할 경우엔 sigmoid 함수, 0보다 큰 출력 값만 내보내고자 할 경우엔 relu 함수를 사용합니다. 특히 다음은 분류를 위해 출력 층에 사용할 수 있는 활성화 함수로 softmax라고 합니다.



활성화 함수에 대해서는 뒤에서 자세히 살펴보도록 합니다. 여기서는 활성화 함수로 이러한 함수들이 사용된다는 정도로 이해하고 넘어갑니다.

인공 신경 함수 수식

다음은 하나의 인공 신경과 그 인공 신경으로 들어가는 ❶ 입력 값 x 의 집합, ❷ 입력 값에 대한 가중치 (신호 강도) w 의 집합, ❸ 편향 입력 값 1, ❹ 편향 b , ❺ 가중치와 편향을 통해 들어오는 입력 값들의 합, ❻ 그 합을 입력으로 받는 활성화 함수 f , ❼ 활성화 함수 f 의 출력 out 을 나타냅니다.



인공 신경의 수식은 일반적으로 다음과 같습니다.

$$out = f(x_1 \times w_1 + x_2 \times w_2 + x_3 \times w_3 + \dots + x_n \times w_n + 1 \times b)$$

$$out = f\left(\sum_{i=1}^n x_i \times w_i + 1 \times b\right)$$

예를 들어, 활성화 함수가 sigmoid 함수일 경우 인공 신경의 수식은 다음과 같습니다.

$$out = \frac{1}{1 + e^{-(x_1 \times w_1 + x_2 \times w_2 + x_3 \times w_3 + \dots + x_n \times w_n + 1 \times b)}}$$

$$out = \frac{1}{1 + e^{-\left(\sum_{i=1}^n x_i \times w_i + 1 \times b\right)}}$$

또, 활성화 함수가 relu 함수일 경우 인공 신경의 수식은 다음과 같습니다.

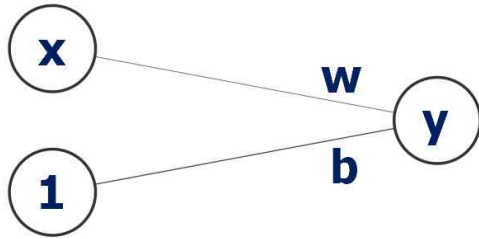
$$out = \max(0, x_1 \times w_1 + x_2 \times w_2 + x_3 \times w_3 + \dots + x_n \times w_n + 1 \times b)$$

$$out = \max\left(0, \sum_{i=1}^n x_i \times w_i + 1 \times b\right)$$

이러한 수식들은 뒤에서 자세히 구현해 보면서 그 동작들을 이해합니다. 여기서는 개략적으로 살펴보고 넘어가도록 합니다.

가장 간단한 인공 신경

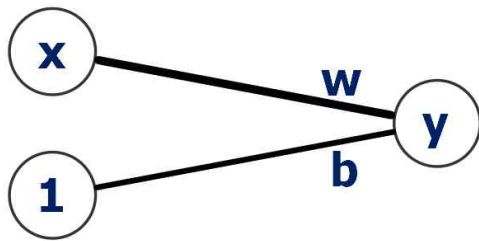
다음은 가장 간단한 형태의 인공 신경입니다.



이 인공 신경의 수식은 다음과 같습니다.

$$y = x*w + 1*b$$

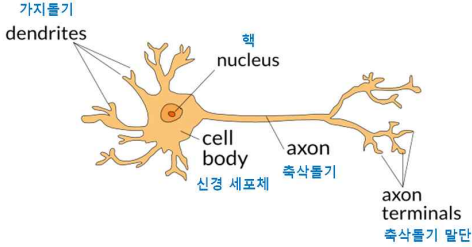
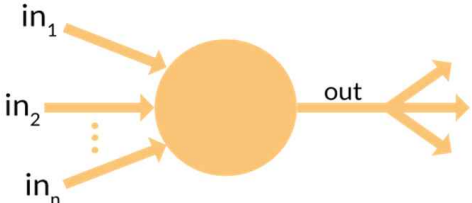
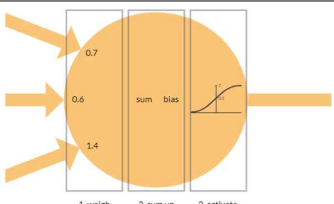
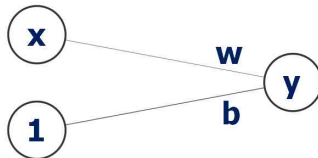
인공 신경의 학습과정은 가중치 w 와 편향 b 의 값을 조절하는 과정입니다. 인공 신경이 적절히 학습되면 입력값 x 에 대해 원하는 출력값 y 가 나오게 됩니다. 다음은 학습 과정을 거친 인공 신경입니다.



우리는 뒤에서 이 인공 신경을 직접 구현해보면서 인공 신경의 동작을 살펴봅니다.

이상에서 인간의 두뇌를 모델로 한 인공 신경망, 인공 신경망으로 할 수 있는 일들, 인공 신경망의 구조, 인공 신경망의 학습 방법, 생물학적 신경과 인공 신경과의 관계, 인공 신경의 구성 요소를 살펴보았습니다. 이 과정에서 인공 신경의 수식은 생물학적 신경으로부터 직관적으로 유도된 것을 알 수 있었습니다. 인공 신경의 수식은 간단한 형태의 수식이지만 이러한 인공 신경으로 망을 구성할 때는 아주 큰 힘을 발휘하게 됩니다.

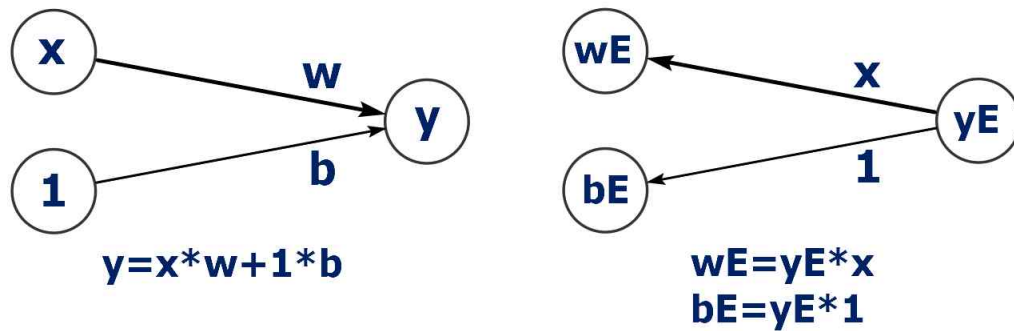
정리하기

1. 생체 신경	
2. 구조 단순화	
3. 동작 분석	
4. 단위 인공 신경망 구조	
5. 단위 인공 신경망 수식	$y = x*w + 1*b$

03 딥러닝 7 공식

여기서는 딥러닝 7 공식을 이용하여 딥러닝의 동작 원리를 이해해 봅니다. 또 딥러닝과 관련된 중요한 용어들, 예를 들어, 순전파, 목표값, 오차, 역전파 오차, 역전파, 학습률과 같은 용어들을 이해해 보도록 합니다.

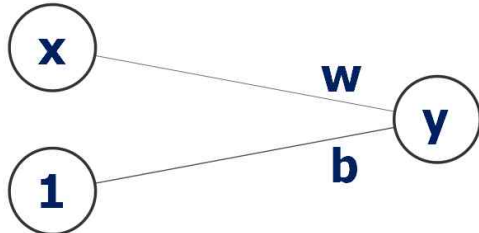
다음 그림은 이 단원에서 살펴볼 인공 신경의 순전파와 역전파를 나타내는 핵심 그림과 수식을 표현하고 있습니다.



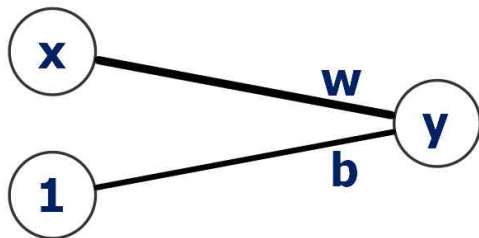
독자 여러분이 이 그림과 수식의 내용을 이해한다면 딥러닝의 가장 중요한 원리를 이해하게 되는 것입니다. 딥러닝의 핵심 원리 지금부터 이해해 봅시다!

01 딥러닝 제 1 공식 : 순전파

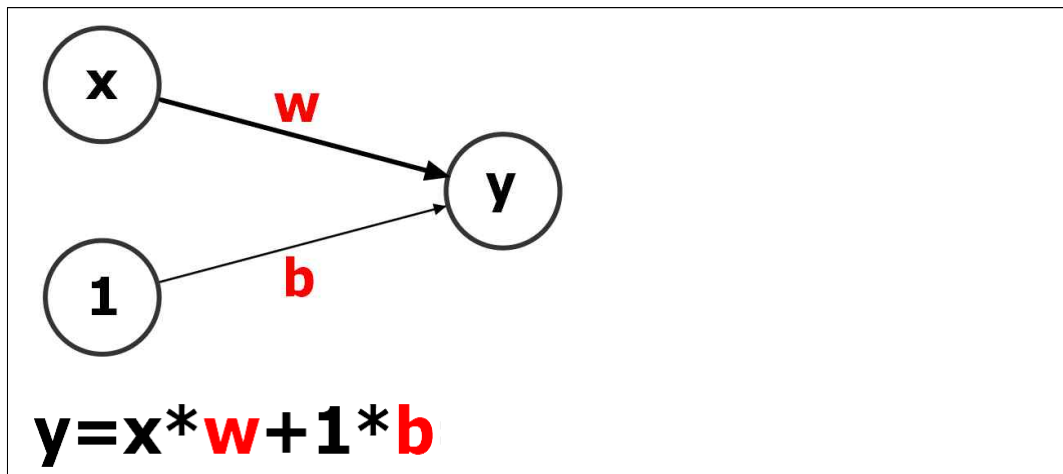
다음은 앞에서 소개한 단일 인공 신경의 그림입니다. 이 인공 신경은 입력 노드 1개, 출력 노드 1개, 편향으로 구성된 단일 인공 신경입니다.



이 인공 신경을 학습 시키면 다음과 같이 가중치와 편향의 값이 바뀌게 됩니다. 즉, 신호 전달 강도가 더 세지거나 약해집니다.



순전파 그림과 수식은 다음과 같습니다.



이 수식은 순전파 수식입니다. **y**는 예측값, **x**는 입력값, **w**는 가중치, **b**는 편향이라고 합니다.

순전파 살펴보기

이 수식에 대해서 구체적으로 생각해 봅시다. 다음과 같이 각 변수에 값을 줍니다.

$$\begin{aligned}x &= 2 \\w &= 3 \\b &= 1\end{aligned}$$

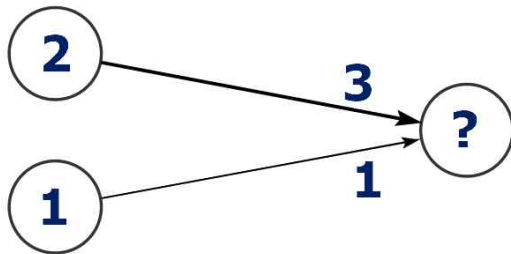
그러면 식은 다음과 같이 됩니다.

$$\begin{aligned}y &= 2*3 + 1*1 \\y &= ?\end{aligned}$$

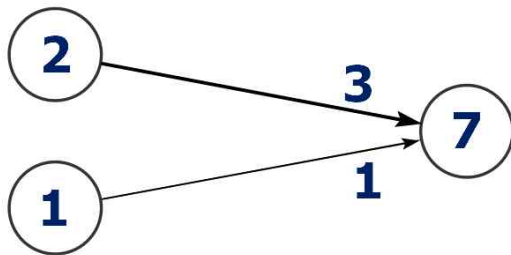
y는 얼마가 될까요? 다음과 같이 계산해서 y는 7이 됩니다.

$$2*3 + 1*1 = 7$$

이 상황을 그림으로 생각해 봅시다. 다음과 같이 x, w, b 값이 y로 흘러가는 인공 신경이 있습니다. 이 과정을 인공 신경의 순전파라고 합니다.



이 경우 y로 얼마가 나올까요? 앞에서 살펴본 대로 다음과 같이 7이 나오게 됩니다.



여기서 순전파를 통해 나온 7을 예측값이라고 합니다.

순전파 수행하기

돌다리도 두들겨 보고 건너다고 순전파를 파이썬 셸을 통해 확인해 봅니다.

다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> x=2
>>> w=3
>>> b=1
>>> y=x*w+1*b
>>> y
7
```

02 딥러닝 제 2 공식 : 평균 제곱 오차

다음은 딥러닝 제 2 공식을 나타냅니다.

$$E = (y - y_T)^2 / 2$$

이 수식은 평균 제곱 오차 수식이라고 합니다. E는 오차, y는 순전파에 의한 예측값, yT는 목표값 또는 라벨을 나타냅니다. yT는 입력값 x에 대해 실제로 나오기를 원하는 값입니다. 오차(error)는 손실(loss) 또는 비용(cost)이라고도 합니다. 오차값이 작을수록 예측을 잘하는 인공 신경망입니다.

앞에서 우리는 y값으로 7을 얻었습니다. 그런데 y로 10이 나오게 하고 싶습니다. 이 경우 yT 값은 10이 됩니다. 그러면 평균 제곱 오차는 다음과 같이 계산됩니다.

$$E = (7 - 10)^2 / 2 = (-3)^2 / 2 = 9 / 2 = 4.5$$

평균 제곱 오차 수행하기

다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> yT=10
>>> E=(y-yT)**2/2
>>> E
4.5
```

03 딥러닝 제 3 공식 : 역전파 오차

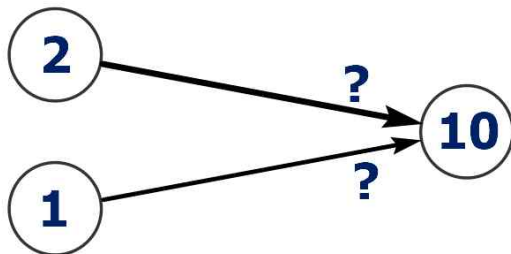
다음은 딥러닝 제 3 공식을 나타냅니다.

$$y^E = y - y^T$$

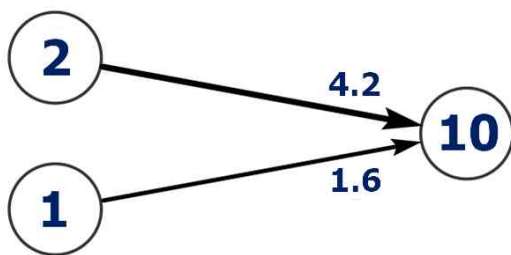
y^E 는 역전파 오차, y 는 순전파에 의한 예측값, y^T 는 목표값 또는 라벨을 나타냅니다. y^T 는 입력값에 대해 실제로 나오기를 원하는 값입니다. 역전파 오차를 구하는 수식은 단순합니다. 예측값 y 에서 목표값 y^T 를 뺀 값이 역전파 오차가 됩니다. y^E 값이 다음 단계에서 역전파되기 때문에 역전파 오차라고 합니다. y^E 의 정확한 의미는 y 에 대한 오차 E 의 순간변화율을 의미하며 편미분을 통해 유도됩니다.

딥러닝 제 3 공식에 대해서 구체적으로 살펴봅니다.

앞에서 우리는 y 값으로 7을 얻었습니다. 그런데 y 로 10이 나오게 하고 싶습니다. 이런 경우에 w 와 b 의 값을 어떻게 바꿔야 할까요?



y 값이 10이 되려면 3이 모자랍니다. y 의 오차는 w 와 b 의 오차로 인해 발생합니다. 따라서 w 와 b 값을 적당히 증가시키면 y 로 10에 가까운 값이 나오게 할 수 있겠죠? 예를 들어, w 를 4.2로, b 를 1.6으로 증가시키면 y 의 값은 10이 됩니다.



그러면 w , b 값을 어떤 기준으로 얼마나 증가시켜야 할까요? 이 과정을 이해하는 것이 바로 역전파의 핵심을 이해하는 것이고 나아가서 딥러닝의 핵심을 이해하는 것입니다. 이 과정을 자세히 살펴봅니다.

이전 그림에서 y 로 7이 흘러나갔는데 우리는 이 값이 10이 되기를 원합니다. 여기서 10은 목표값이 됩니다. 참고로 인공지능 학습 시 목표값은 라벨이라고 합니다. 다음 수식에서 y^T 는 목표값을 나타내며 10을 갖습니다.

$$\begin{aligned} y^T &= 10 \\ y &= 7 \\ y^E &= y - y^T = -3 \end{aligned}$$

y 값은 현재값 7인 상태이며, y^E 는 현재값에서 목표값을 뺀 값 -3이 됩니다. 이 때, y^E 값을 역전파 오차라고 하며, 역전파에 사용할 오차값입니다.

역전파 오차 수행하기

다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> yE=y-yT
>>> yE
-3
```

04 딥러닝 제 4 공식 : 입력 역전파

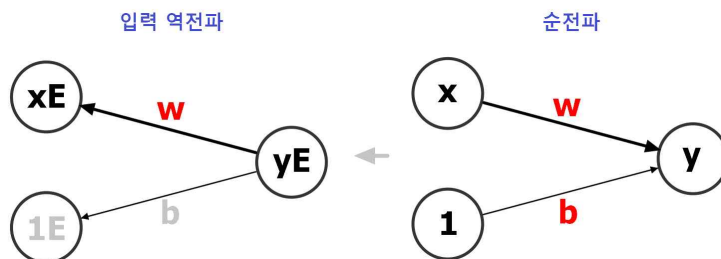
다음은 딥러닝 제 4 공식을 나타냅니다.



x_E 는 입력 역전파, y_E 는 역전파 오차로 딥러닝 제 3 공식에서 구한 값입니다. 회색으로 표시된 1_E 는 숫자 1의 오차라는 의미로 사용하지 않는 부분입니다. 딥러닝 제 4 공식은 다음과 같은 순서로 유도할 수 있습니다.

1. 딥러닝 제 1 공식의 그림을 복사합니다.
2. $y \rightarrow y_E$, $x \rightarrow x_E$ 로 변경합니다.
3. 화살표 방향을 반대로 합니다.
4. 1_E , b 는 사용하지 않습니다.

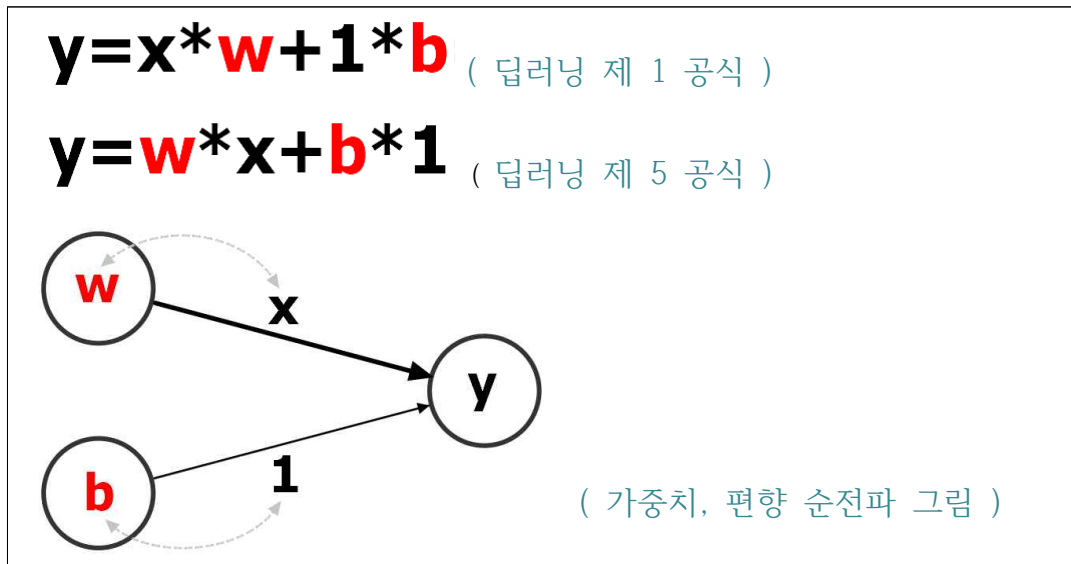
다음 그림을 참조합니다.



이 수식은 출력층에서 발생한 역전파 오차가 입력층으로 흘러가는 상황을 나타냅니다. 딥러닝 제 4 공식은 실제로 은닉층으로 전달되는 역전파에 사용하는 공식으로 뒤에서 사용합니다. x_E 의 정확한 의미는 x 에 대한 오차 E 의 순간변화율을 의미하며 편미분을 통해 유도됩니다.

05 딥러닝 제 5 공식 : 가중치, 편향 순전파

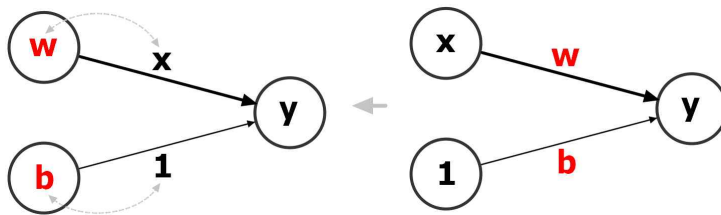
다음은 가중치와 편향의 순전파를 나타내는 식과 그림입니다.



딥러닝 제 5 공식은 다음과 같은 순서로 유도할 수 있습니다.

1. 딥러닝 제 1 공식을 복사합니다.
2. x와 w, 1과 b를 교환하여 딥러닝 제 5 공식을 유도합니다.
3. 딥러닝 제 1 공식의 그림과 같은 형태로 그림을 그립니다.

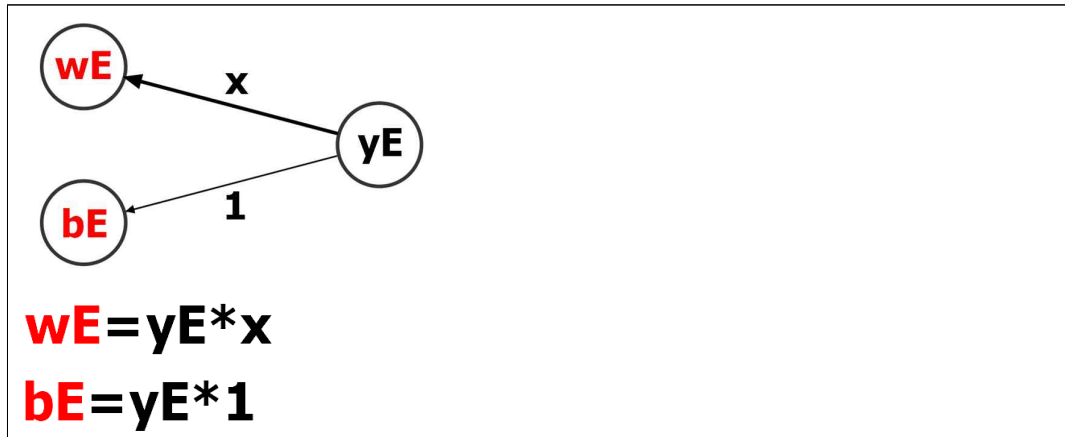
다음 그림을 참조합니다.



딥러닝 제 5 공식은 다음에 나올 딥러닝 제 6 공식을 유도하기 위해 사용하며, 실제로 딥러닝 제 1 공식과 같습니다. 딥러닝 제 5 공식은 구현을 위해 사용하지는 않습니다.

06 딥러닝 제 6 공식 : 가중치, 편향 역전파

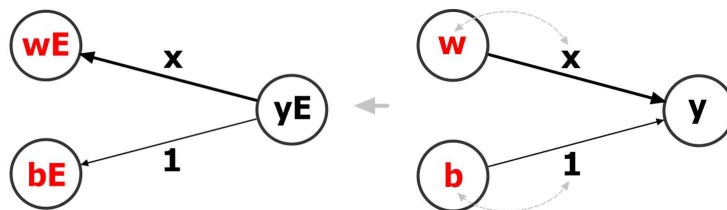
다음은 가중치와 편향의 역전파를 나타내는 그림과 수식입니다.



wE는 가중치 역전파 오차, bE는 편향 역전파 오차를 의미합니다. yE는 역전파 오차로 딥러닝 제 3 공식에서 구한 값입니다. 딥러닝 제 6 공식은 다음과 같은 순서로 유도할 수 있습니다.

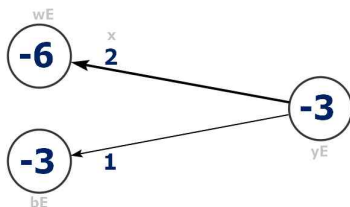
1. 딥러닝 제 5 공식의 그림을 복사합니다.
2. $y \rightarrow yE$, $w \rightarrow wE$, $b \rightarrow bE$ 로 변경합니다.
3. 화살표 방향을 반대로 합니다.

다음 그림을 참조합니다.



딥러닝 제 6 공식은 실제로 편미분을 연쇄적으로 적용하여 얻은 공식입니다. 편미분을 이용하여 역전파를 유도하는 방법은 부록을 참조합니다. wE의 정확한 의미는 w에 대한 오차 E의 순간변화율을 의미합니다. 마찬가지로 bE는 b에 대한 오차 E의 순간변화율을 의미합니다.

위 수식에 의해 wE, bE는 다음 그림과 같이 계산됩니다.



딥러닝 제 3 공식의 예에서 우리는 yE값으로 -3을 얻었습니다.

가중치, 편향 역전파 수행하기

다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> wE=yE*x
>>> bE=yE*1
>>> (wE,bE)
(-6, -3)
```

07 딥러닝 제 7 공식 : 신경망 학습

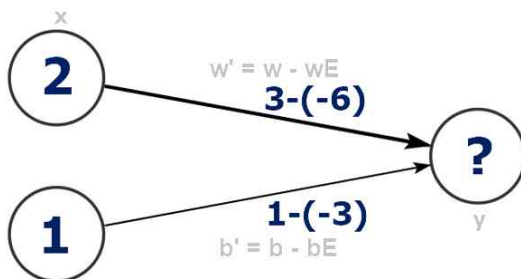
다음은 신경망 학습을 나타내는 수식입니다. 신경망은 가중치를 의미합니다.

$$\begin{aligned}w &= lr * wE \\ b &= lr * bE\end{aligned}$$

lr은 learning rate의 약자로 학습률을 의미합니다. wE 는 가중치 역전파 오차, bE 는 편향 역전파 오차로 딥러닝 제 6 공식에서 구한 값입니다. 딥러닝 제 7 공식은 실제로 경사하강법이라고 하며 미분을 이용하여 얻은 공식입니다. 이 수식을 적용하여 w , b 값을 갱신하는 과정을 인공 신경망의 학습이라고 합니다.

신경망 학습해 보기

앞에서 우리는 wE , bE 값으로 각각 -6, -3을 구하였습니다. wE , bE 값을 다음 그림과 같이 적용해 봅니다. 다음 그림에서 w 는 이전 가중치, w' 는 갱신된 가중치를 의미합니다. 그리고 앞에서 구한 wE , bE 의 값을 각각 w , b 에서 빼줍니다. 그러면 갱신된 w' , b' 값이 증가하게 됩니다. 이 부분을 좀 더 자세히 살펴봅시다. 앞에서 살펴본 딥러닝 제 3 공식을 변형하면 $y - yE = yT$ 가 되어 y 에서 yE 를 빼면 yT 가 됩니다. 즉, y 값 7에서 yE 값 -3을 빼주면 yT 값 10이 됩니다. 같은 원리로 w 에서 wE 를, b 에서 bE 를 빼주도록 합니다.

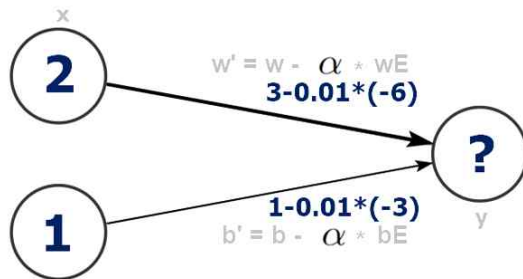


그런데 wE , bE 값이 너무 큼니다. 이 상태로 계산을 하면 새로운 y 값은 $(2*9+1*4)$ 와 같이 계산되어 22가 되게 되며, 우리가 원하는 목표값 10보다 더 큰 값이 나오게 됩니다. 구체적인 계산 과정은 다음과 같습니다.

$$\begin{aligned}x &= 2 \\ w' &= w - wE = 3 - (-6) = 9 \\ b' &= b - bE = 1 - (-3) = 4 \\ y &= x * w' + 1 * b' = 2 * 9 + 1 * 4 = 22\end{aligned}$$

학습률 적용하기

그래서 나온 방법이 학습률입니다. wE , bE 에 적당히 작은 값을 곱해주어 값을 줄이는 겁니다. 여기서는 0.01을 곱해줍니다. 그러면 다음과 같이 계산할 수 있습니다.



*** 여기서 α 는 학습률이라고 하며 뒤에서는 lr이라는 이름으로 구현합니다. lr은 learning rate의 약자로 학습률을 의미합니다.

이렇게 하면 $2 * (3.06) + 1.03 = 7.15$ 가 나옵니다. 이렇게 조금씩 늘려나가면 10을 만들 수 있습니다.

여기서 곱해준 0.01은 학습률이라고 하는 값입니다. 일반적으로 학습률 값은 0.001로 시작하여 학습이 진행되는 상황에 따라 조금씩 늘이거나 줄여서 사용합니다.

신경망 학습 수행하기

다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> lr=0.01
>>> w=w-lr*wE
>>> b=b-lr*bE
>>> (w,b)
(3.06, 1.03)
```

학습된 신경망으로 예측하기

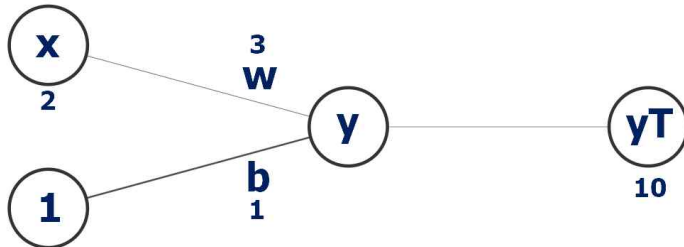
인공 신경망에 대해 딥러닝 제 1 공식~딥러닝 제 7 공식까지 1회 수행을 하면 한 번의 학습이 이루어지게 됩니다. 여기서는 1회 학습된 신경망으로 y 값을 예측해 봅니다.

다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> y=x*w+1*b
>>> y
7.15
```

08 딥러닝 반복 학습해 보기

지금까지의 과정을 반복해서 학습해 봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력값 x , 가중치 w , 편향 b 는 각각 2, 3, 1이고 목표값 yT 는 10입니다. x , yT 를 이용하여 w , b 에 대해 학습을 수행해 봅니다.

*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

반복 학습 2회 수행하기

여기서는 반복 학습 2회를 수행해 봅니다.

1. 다음과 같이 파이썬 스크립트를 작성합니다.


238.py

```
01 x = 2
02 w = 3
03 b = 1
04 yT = 10
05 lr = 0.01
06
07 for epoch in range(2):
08
09     y = x*w + 1*b
10     E = (y - yT)**2 / 2
11     yE = y - yT
12     wE = yE*x
13     bE = yE*1
14     w -= lr*wE
15     b -= lr*bE
16
17     print(f'epoch = {epoch}')
18     print(f' y : {y:.3f}')
19     print(f' w : {w:.3f}')
```

```
20 print(f' b : {b:.3f}')
```

07 : epoch값을 0에서 2 미만까지 바꾸어가며 09~20줄을 2회 수행합니다.

17 : print 함수를 호출하여 epoch 값을 출력해 줍니다.

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

```
epoch = 0
y : 7.000
w : 3.060
b : 1.030
epoch = 1
y : 7.150
w : 3.117
b : 1.058
```


y 값이 7에서 7.150으로 바뀌는 것을 확인합니다. w, b 값을 확인합니다. 또, w, b 값을 확인합니다. w, b의 값이 아주 조금 증가한 것을 확인합니다. 가지 돌기의 두께가 조금 두꺼워진 것과 같습니다.

반복 학습 20회 수행하기

여기서는 반복 학습 20회를 수행해 봅니다.

1. 다음과 같이 예제를 수정합니다.

```
07 for epoch in range(20):
```

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

```
epoch = 18
y : 8.808
w : 3.747
b : 1.374
epoch = 19
y : 8.868
w : 3.770
b : 1.385
```


y 값이 8.868까지 접근하는 것을 확인합니다. w, b의 값이 처음에 비해 증가한 것을 확인합니다. 가지돌기의 두께가 더 두꺼워진 것과 같습니다.

반복 학습 200회 수행하기

여기서는 반복 학습 200회를 수행해 봅니다.

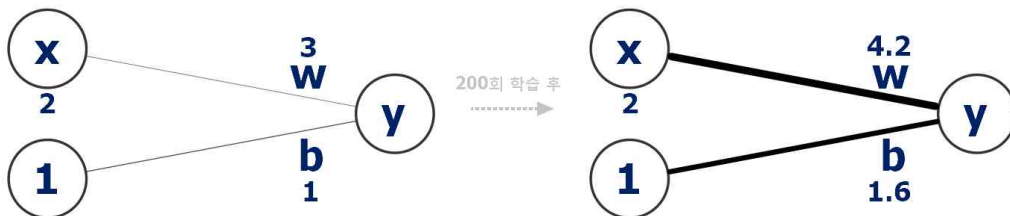
1. 다음과 같이 예제를 수정합니다.

```
07 for epoch in range(200):
```

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

```
epoch = 198  
y : 10.000  
w : 4.200  
b : 1.600  
epoch = 199  
y : 10.000  
w : 4.200  
b : 1.600
```

y 값이 10.000에 수렴하는 것을 확인합니다. 이 때, 가중치 w는 4.2, 편향 b는 1.6에 수렴합니다.



200회 학습 후에는 그림처럼 w, b의 강도가 세지는 것을 볼 수 있습니다. 인공 신경망의 학습은 이처럼 가중치 w, 편향 b의 값을 조정하는 과정입니다.

오차 조건 추가하기


여기서는 오차값이 충분히 작아지면 학습을 자동으로 중단하도록 해 봅니다.

1. 다음과 같이 이전 예제를 수정합니다.

238_3.py

```
01 x = 2
02 w = 3
03 b = 1
04 yT = 10
05 lr = 0.01
06
07 for epoch in range(200):
08
09     y = x*w + 1*b
10     E = (y - yT)**2 / 2
11     yE = y - yT
12     wE = yE*x
13     bE = yE*1
14     w = w - lr*wE
15     b = b - lr*bE
16
17     print(f'epoch = {epoch}')
18     print(f' y : {y:.3f}')
19     print(f' w : {w:.3f}')
20     print(f' b : {b:.3f}')
21
22     if E < 0.0000001 :
23         break
```

22~23 : 오차값 E가 0.0000001(1천만분의1)보다 작으면 break문을 수행하여 07줄의 for문을 빠져 나갑니다.

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

```
epoch = 171
y : 10.000
w : 4.200
b : 1.600
epoch = 172
y : 10.000
w : 4.200
b : 1.600
```


epoch 값이 172일 때 for 문을 빠져 나갑니다. y값은 10에 수렴합니다.

학습률 변경하기

여기서는 학습률 값을 변경시켜 보면서 학습의 상태를 살펴봅니다.

1. 다음과 같이 예제를 수정합니다.

```
05 lr = 0.05
```


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

```
epoch = 30
y : 9.999
w : 4.200
b : 1.600
epoch = 31
y : 10.000
w : 4.200
b : 1.600
```

epoch 값이 31일 때 학습이 완료되는 것을 볼 수 있습니다.

3. 다음과 같이 예제를 수정합니다.

```
05 lr = 0.005
```


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

```
epoch = 198
y : 9.980
w : 4.192
b : 1.596
epoch = 199
y : 9.981
w : 4.192
b : 1.596
```

epoch 값이 199일 때 학습이 완료되지 않은 상태로 종료되는 것을 볼 수 있습니다.

5. 다음과 같이 예제를 수정합니다.

```
07 for epoch in range(2000):
```

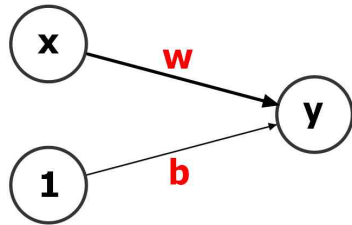
6.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 347
y : 10.000
w : 4.200
b : 1.600
epoch = 348
y : 10.000
w : 4.200
b : 1.600
```

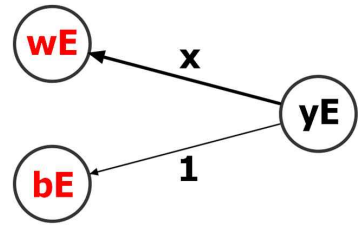
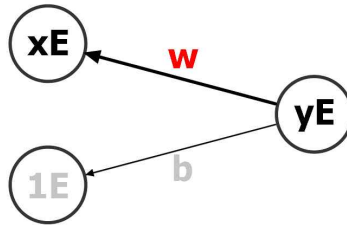
epoch 값이 349일 때 학습이 완료되는 것을 볼 수 있습니다.

09 딥러닝 7 공식 정리하기

순전파



역전파



04 딥러닝 7 공식 확장하기

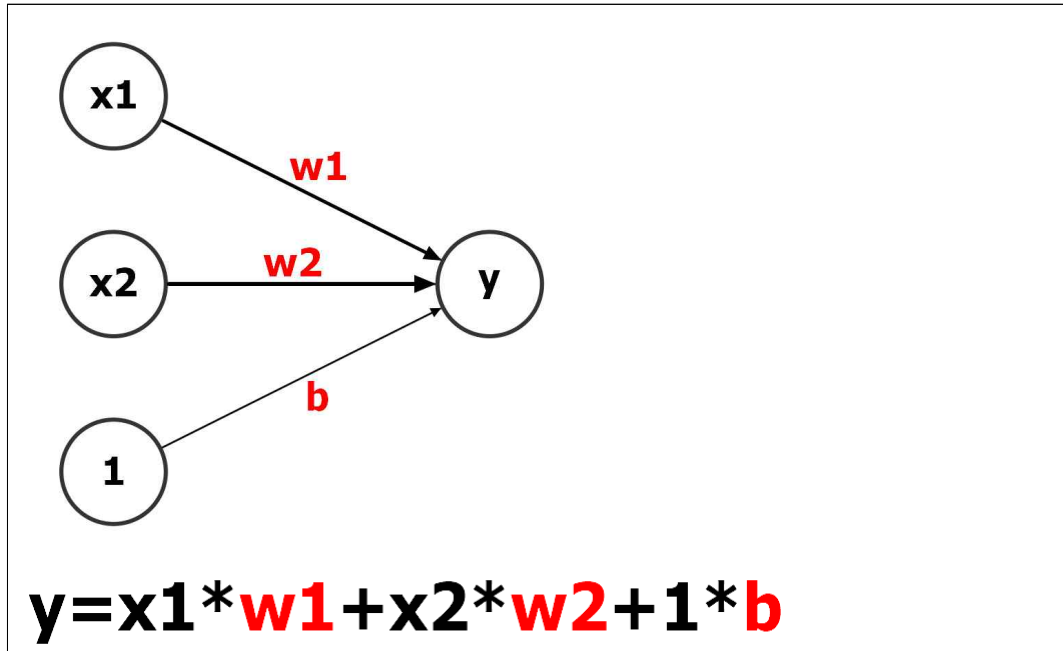
지금까지 우리는 가장 간단한 형태인 [1입력 1출력] 인공 신경에 대해 딥러닝 7 공식을 살펴보고 구현해 보았습니다. 여기서는 [2입력 1출력] 인공 신경과 [2입력 2출력] 인공 신경망으로 딥러닝 7 공식을 확장해 봅니다.

01 2입력 1출력 인공 신경

먼저 [2입력 1출력] 인공 신경에 대해 딥러닝 7 공식을 확장하고 구현해 봅니다.

딥러닝 제 1 공식 : 순전파

다음은 [2입력 1출력] 인공 신경의 딥러닝 제 1 공식을 나타냅니다.



이 수식은 순전파 수식입니다. y 는 예측값, $x1$, $x2$ 는 입력값, $w1$, $w2$ 는 가중치, b 는 편향입니다. 앞에서 살펴본 [1입력 1출력] 인공 신경에 입력과 가중치가 하나씩 추가되었습니다.

다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> x1,x2=2,3
>>> w1,w2=3,4
>>> b=1
>>> y=x1*w1+x2*w2+1*b
>>> y
19
```

딥러닝 제 2 공식 : 평균 제곱 오차

다음은 [2입력 1출력] 인공 신경의 딥러닝 제 2 공식을 나타냅니다.

$$E = (y - y_T) * (y - y_T) / 2$$

[2입력 1출력] 인공 신경은 [1입력 1출력] 인공 신경과 출력의 개수가 같기 때문에 평균 제곱 오차 수식이 같습니다.

다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> yT=27
>>> E=(y-yT)**2/2
>>> E
32.0
```

딥러닝 제 3 공식 : 역전파 오차

다음은 [2입력 1출력] 인공 신경의 딥러닝 제 3 공식을 나타냅니다.

$$yE = y - yT$$

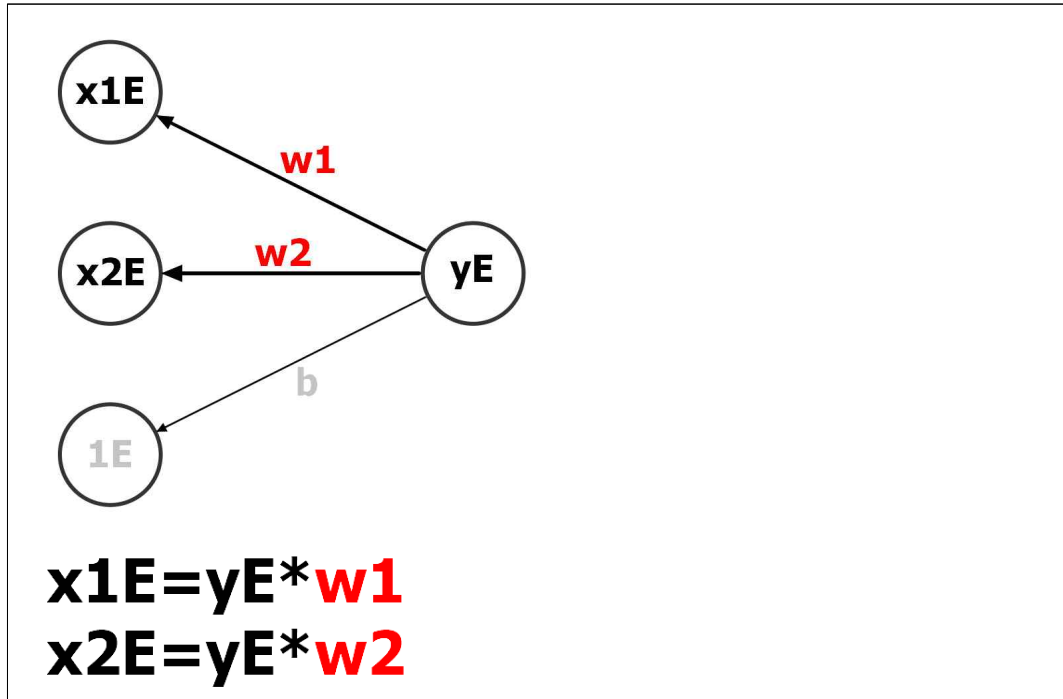
[2입력 1출력] 인공 신경은 [1입력 1출력] 인공 신경과 출력의 개수가 같기 때문에 역전파 오차 수식이 같습니다.

다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> yE=y-yT
>>> yE
-8
```

딥러닝 제 4 공식 : 입력 역전파

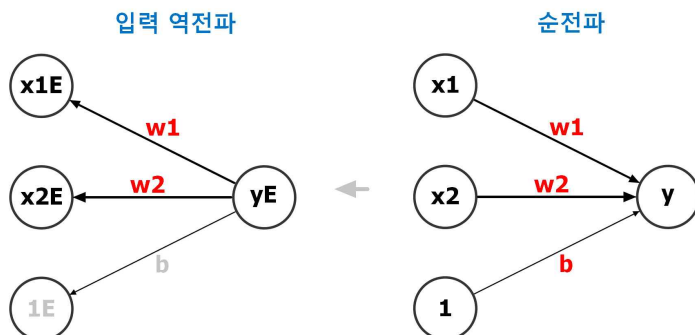
다음은 [2입력 1출력] 인공 신경의 딥러닝 제 4 공식을 나타냅니다.



$x1E$, $x2E$ 는 입력 역전파, yE 는 역전파 오차로 딥러닝 제 3 공식에서 구한 값입니다. 회색으로 표시된 $1E$ 는 숫자 1의 오차라는 의미로 사용하지 않는 부분입니다. 딥러닝 제 4 공식은 다음과 같은 순서로 유도할 수 있습니다.

1. 딥러닝 제 1 공식의 그림을 복사합니다.
2. $y \rightarrow yE$, $x1 \rightarrow x1E$, $x2 \rightarrow x2E$ 로 변경합니다.
3. 화살표 방향을 반대로 합니다.
4. $1E$, b 는 사용하지 않습니다.

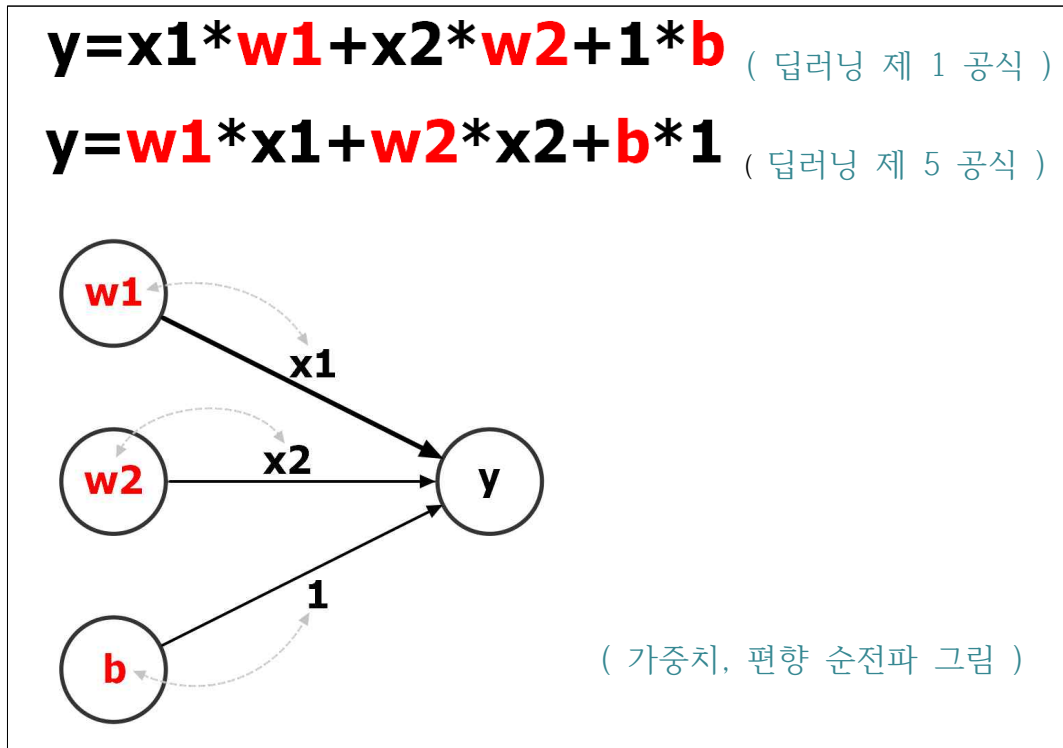
다음 그림을 참조합니다.



이 수식은 출력층에서 발생한 역전파 오차가 입력층으로 흘러가는 상황을 나타냅니다. 딥러닝 제 4 공식은 실제로 은닉층으로 전달되는 역전파에 사용하는 공식으로 뒤에서 사용합니다.

05 딥러닝 제 5 공식 : 가중치, 편향 순전파

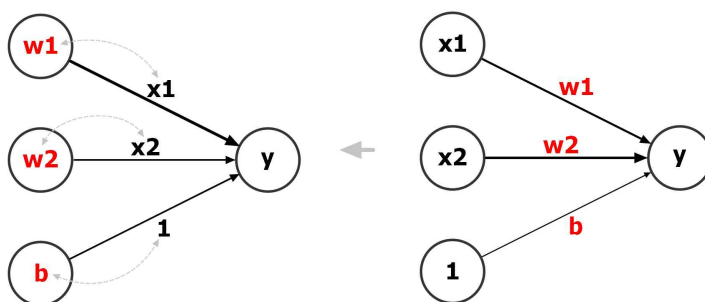
다음은 [2입력 1출력] 인공 신경의 가중치와 편향의 순전파를 나타내는 식과 그림입니다.



딥러닝 제 5 공식은 다음과 같은 순서로 유도할 수 있습니다.

1. 딥러닝 제 1 공식을 복사합니다.
2. $x1$ 과 $w1$, $x2$ 와 $w2$, 1 과 b 를 교환하여 딥러닝 제 5 공식을 유도합니다.
3. 딥러닝 제 1 공식의 그림과 같은 형태로 그림을 그립니다.

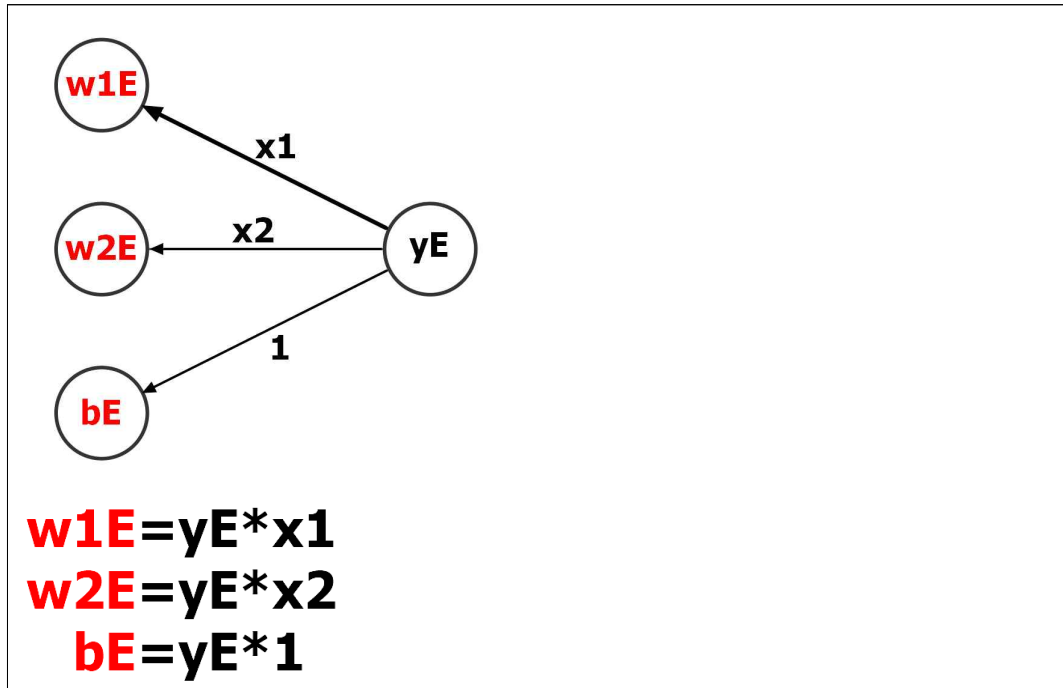
다음 그림을 참조합니다.



딥러닝 제 5 공식은 다음에 나올 딥러닝 제 6 공식을 유도하기 위해 사용하며, 실제로 딥러닝 제 1 공식과 같습니다. 딥러닝 제 5 공식은 구현을 위해 사용하지는 않습니다.

딥러닝 제 6 공식 : 가중치, 편향 역전파

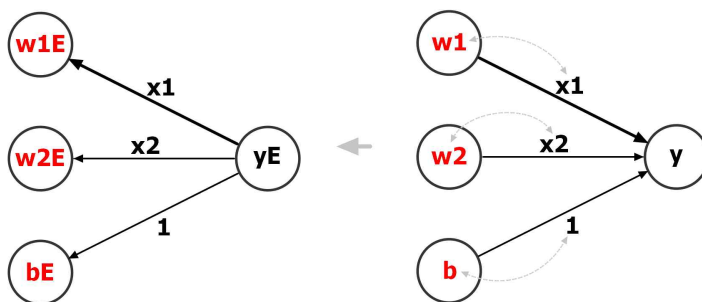
다음은 [2입력 1출력] 인공 신경의 가중치와 편향의 역전파를 나타내는 그림과 수식입니다.



$w1E$, $w2E$ 는 가중치 역전파 오차, bE 는 편향 역전파 오차를 의미합니다. yE 는 역전파 오차로 딥러닝 제 3 공식에서 구한 값입니다. 딥러닝 제 6 공식은 다음과 같은 순서로 유도할 수 있습니다.

1. 딥러닝 제 5 공식의 그림을 복사합니다.
2. $y \rightarrow yE$, $w1 \rightarrow w1E$, $w2 \rightarrow w2E$, $b \rightarrow bE$ 로 변경합니다.
3. 화살표 방향을 반대로 합니다.

다음 그림을 참조합니다.



딥러닝 제 6 공식은 실제로 편미분을 연쇄적으로 적용하여 얻은 공식입니다. 편미분을 이용하여 역전파를 유도하는 방법은 부록을 참조합니다.

다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> w1E=yE*x1
>>> w2E=yE*x2
>>> bE=yE*1
>>> (w1E,w2E,bE)
(-16, -24, -8)
```

07 딥러닝 제 7 공식 : 신경망 학습

다음은 [2입력 1출력] 인공 신경의 신경망 학습을 나타내는 수식입니다.

$$\begin{aligned}w1 &-= lr * w1E \\w2 &-= lr * w2E \\b &-= lr * bE\end{aligned}$$

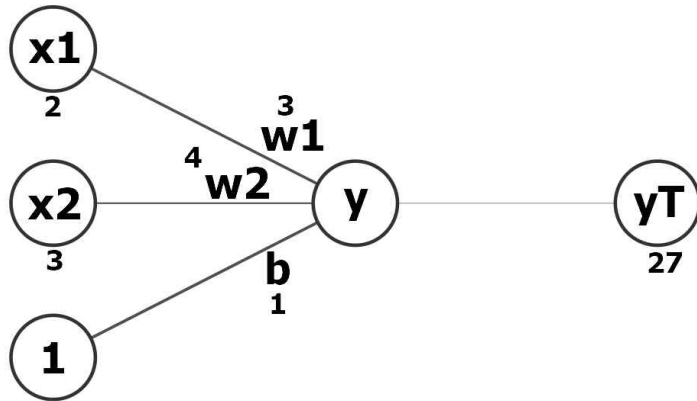
lr은 learning rate의 약자로 학습률을 의미합니다. w1E, w2E는 가중치 역전파 오차, bE는 편향 역전파 오차로 딥러닝 제 6 공식에서 구한 값입니다.

다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> lr=0.01
>>> w1=w1-lr*w1E
>>> w2=w2-lr*w2E
>>> b=b-lr*bE
>>> (w1,w2,b)
(3.16, 4.24, 1.08)
```

딥러닝 반복 학습해 보기

지금까지의 과정을 반복해서 학습해 봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력값 x_1 , x_2 는 각각 2, 3, 가중치 w_1 , w_2 는 각각 3, 4, 편향 b 는 1이고 목표값 yT 는 27입니다. x_1 , x_2 , yT 를 이용하여 w_1 , w_2 , b 에 대해 학습을 수행해 봅니다.


*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

1. 다음과 같이 파이썬 스크립트를 작성합니다.

241_6.py

```
01 x1, x2 = 2, 3
02 w1, w2 = 3, 4
03 b = 1
04 yT = 27
05 lr = 0.01
06
07 for epoch in range(200):
08
09     y = x1*w1 + x2*w2 + 1*b
10     E = (y - yT)**2 / 2
11     yE = y - yT
12     w1E = yE*x1
13     w2E = yE*x2
14     bE = yE*1
15     w1 -= lr*w1E
16     w2 -= lr*w2E
17     b = b - lr*bE
18
19     print(f'epoch = {epoch}')
```

```
20     print(f' y : {y:.3f}')
21     print(f' w1 : {w1:.3f}')
22     print(f' w2 : {w2:.3f}')
23     print(f' b : {b:.3f}')
24
25     if E < 0.0000001:
26         break
```

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

```
epoch = 64
y : 26.999
w1 : 4.143
w2 : 5.714
b : 1.571
epoch = 65
y : 27.000
w1 : 4.143
w2 : 5.714
b : 1.571
```

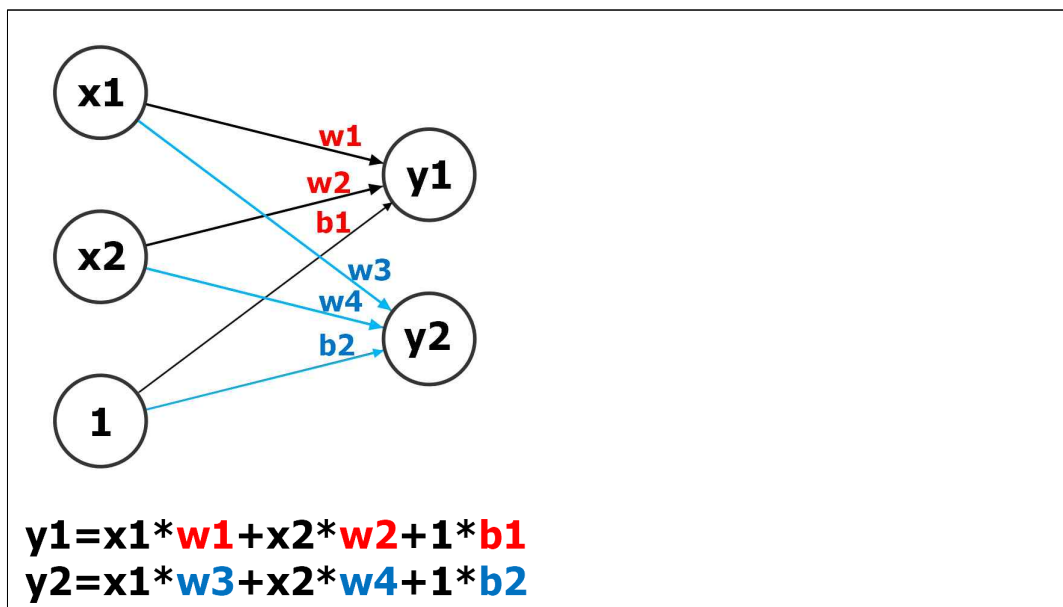
epoch 값이 65일 때 학습이 완료되는 것을 볼 수 있습니다. 가중치 w1, w2는 각각 4.143, 5.714, 편향 b는 1.571에 수렴합니다.

02 2입력 2출력 인공 신경망

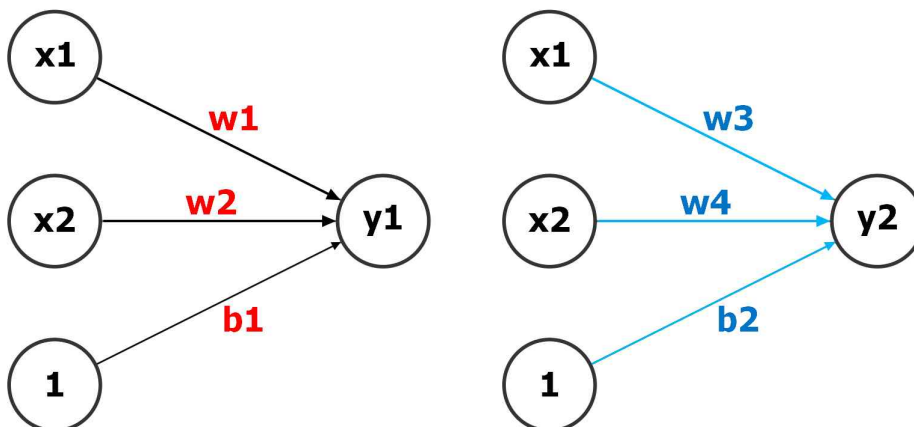
다음은 [2입력 2출력] 인공 신경망에 대해 딥러닝 7 공식을 확장하고 구현해 봅니다. 참고로 [2입력 2출력] 인공 신경망에 대해 적용되는 딥러닝 7 공식은 [m입력 n출력, m, n은 자연수]의 일반적인 인공 신경망에도 적용됩니다.

딥러닝 제 1 공식 : 순전파

다음은 [2입력 2출력] 인공 신경망의 딥러닝 제 1 공식을 나타냅니다.



위의 수식은 순전파 수식입니다. y1, y2는 예측값, x1, x2는 입력값, w1, w2, w3, w4는 가중치, b1, b2는 편향입니다. [2입력 2출력] 인공 신경망은 다음 그림과 같이 [2입력 1출력] 인공 신경 2개로 구성됩니다.



다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> x1,x2=2,3
>>> w1,w2=3,4
>>> w3,w4=5,6
>>> b1,b2=1,2
>>> y1=x1*w1+x2*w2+1*b1
>>> y2=x1*w3+x2*w4+1*b2
>>> (y1,y2)
(19, 30)
```


딥러닝 제 2 공식 : 평균 제곱 오차

다음은 [2입력 2출력] 인공 신경망의 딥러닝 제 2 공식을 나타냅니다.

$$E = (y1-y1T)*(y1-y1T)/2 \\ + (y2-y2T)*(y2-y2T)/2$$

[2입력 2출력] 인공 신경망은 [2입력 1출력] 인공 신경 2개로 구성됩니다. 따라서 각 인공 신경 오차가 더해집니다.

다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> y1T,y2T=27,-30
>>> E=(y1-y1T)**2/2+(y2-y2T)**2/2
>>> E
1832.0
```

딥러닝 제 3 공식 : 역전파 오차

다음은 [2입력 2출력] 인공 신경의 딥러닝 제 3 공식을 나타냅니다.

$$\begin{aligned}y1E &= y1 - y1T \\ y2E &= y2 - y2T\end{aligned}$$

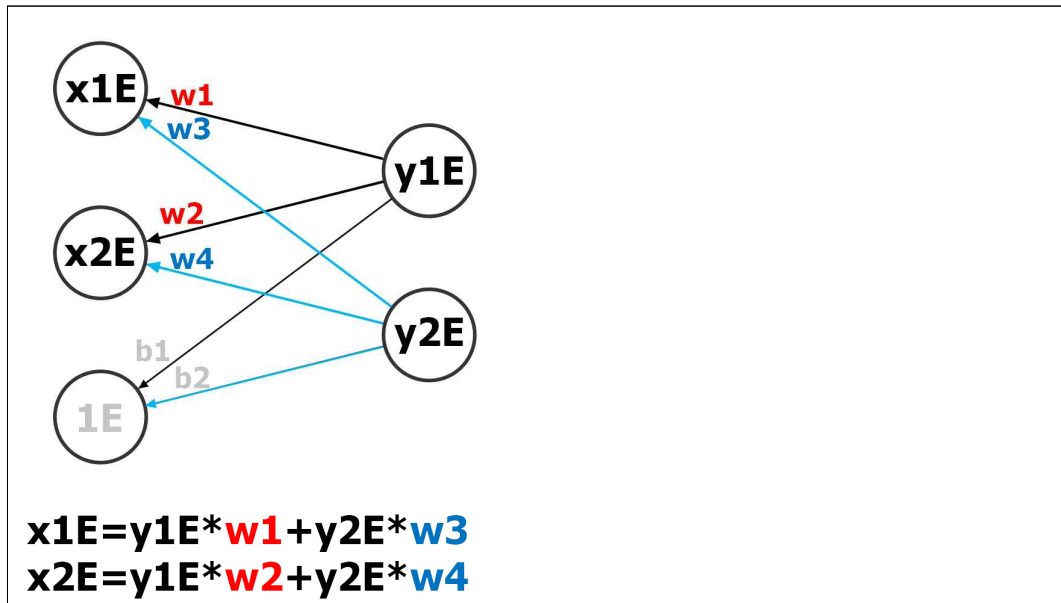
[2입력 2출력] 인공 신경망은 [2입력 1출력] 인공 신경 2개로 구성됩니다. 따라서 각 출력에 대해 역전파 오차가 계산됩니다.

다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> y1E=y1-y1T
>>> y2E=y2-y2T
>>> (y1E,y2E)
(-8, 60)
```

딥러닝 제 4 공식 : 입력 역전파

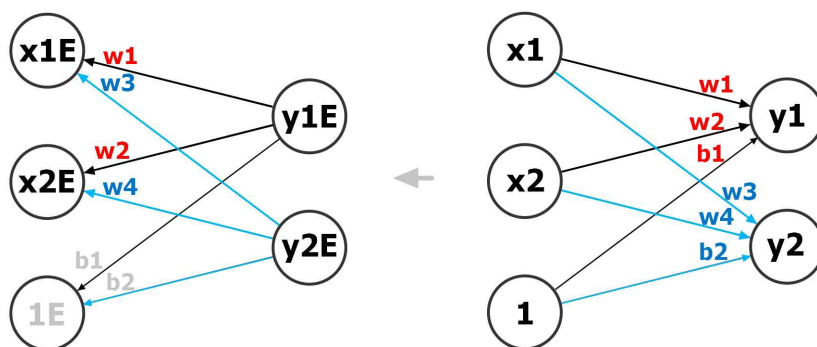
다음은 [2입력 2출력] 인공 신경망의 딥러닝 제 4 공식을 나타냅니다.



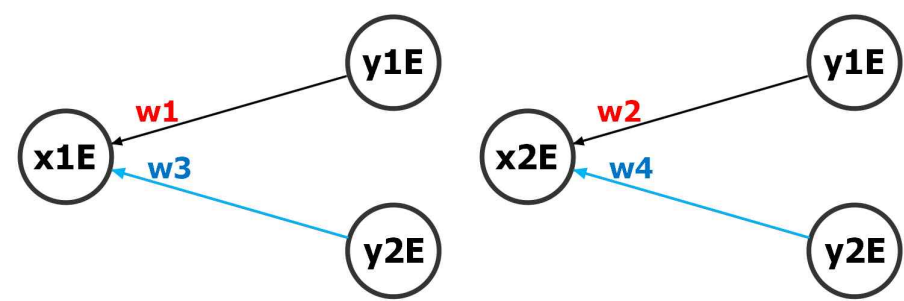
x1E, x2E는 입력 역전파, y1E, y2E는 역전파 오차로 딥러닝 제 3 공식에서 구한 값입니다. 회색으로 표시된 1E는 숫자 1의 오차라는 의미로 사용하지 않는 부분입니다. 딥러닝 제 4 공식은 다음과 같은 순서로 유도할 수 있습니다.

1. 딥러닝 제 1 공식의 그림을 복사합니다.
2. y1 -> y1E, y2 -> y2E, x1 -> x1E, x2 -> x2E로 변경합니다.
3. 화살표 방향을 반대로 합니다.
4. 1E, b1, b2는 사용하지 않습니다.

다음 그림을 참조합니다.



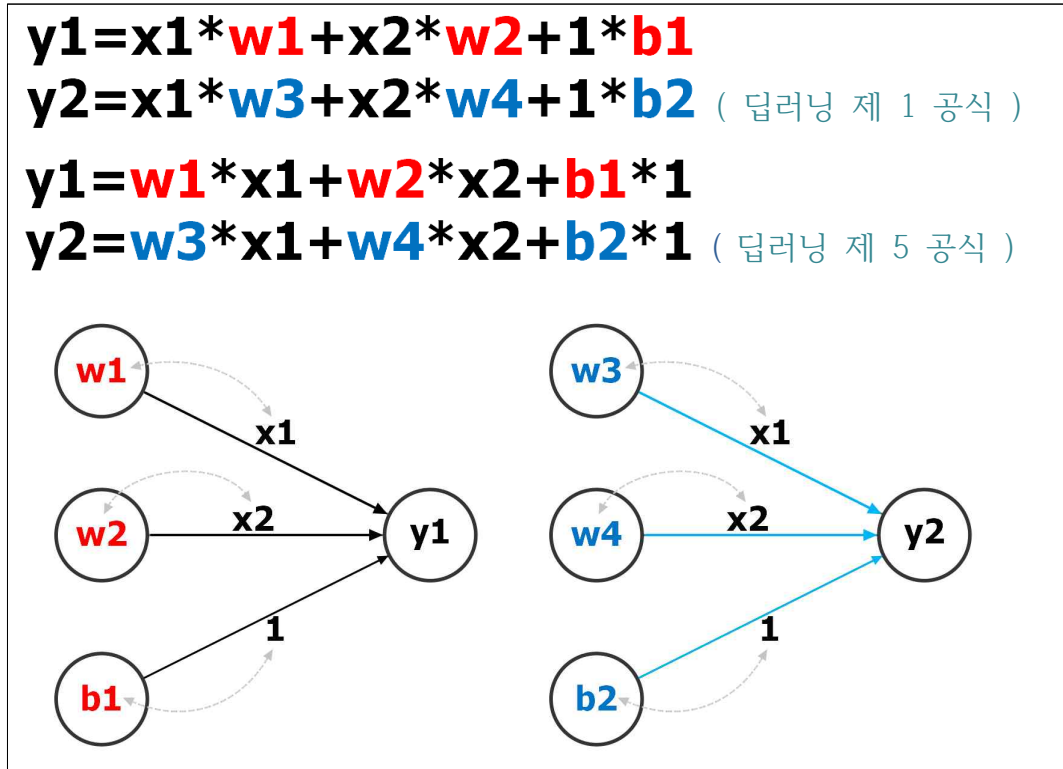
[2입력 2출력] 입력 역전파 그림은 다음과 같이 2개 오차 역전파 망으로 나누어 생각할 수 있습니다.



$y1E$, $y2E$ 는 $w1$, $w3$ 를 따라 $x1E$ 로 전파됩니다. 또, $y1E$, $y2E$ 는 $w2$, $w4$ 를 따라 $x2E$ 로 전파됩니다.

딥러닝 제 5 공식 : 가중치, 편향 순전파

다음은 [2입력 2출력] 인공 신경망의 가중치와 편향의 순전파를 나타내는 식과 그림입니다.



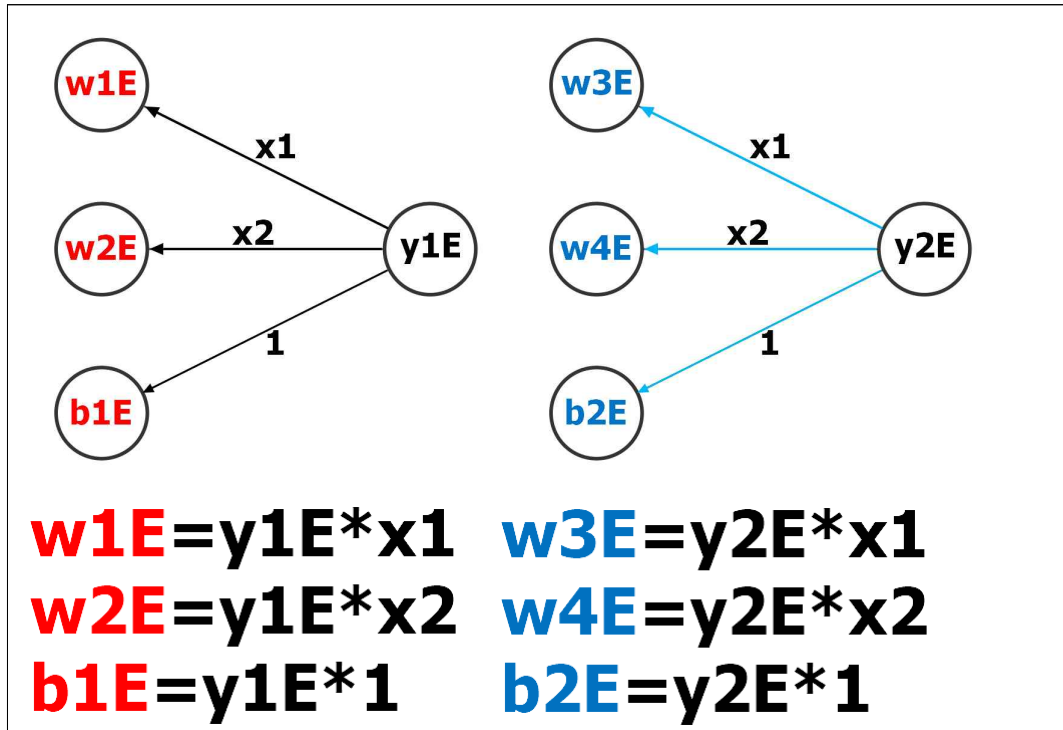
딥러닝 제 5 공식은 다음과 같은 순서로 유도할 수 있습니다.

1. 딥러닝 제 1 공식을 복사합니다.
2. x1과 w1, x2와 w2, 1과 b1,
x1과 w3, x2와 w4, 1과 b2를 교환하여 딥러닝 제 5 공식을 유도합니다.
3. 딥러닝 제 1 공식의 그림과 같은 형태로 그림을 그립니다.

딥러닝 제 5 공식은 다음에 나올 딥러닝 제 6 공식을 유도하기 위해 사용하며, 실제로 딥러닝 제 1 공식과 같습니다. 딥러닝 제 5 공식은 구현을 위해 사용하지는 않습니다.

딥러닝 제 6 공식 : 가중치와 편향 역전파

다음은 [2입력 2출력] 인공 신경망의 가중치와 편향의 역전파를 나타내는 그림과 수식입니다.



w1E, w2E, w3E, w4E는 가중치 역전파 오차, b1E, b2E는 편향 역전파 오차를 의미합니다. y1E, y2E는 역전파 오차로 딥러닝 제 3 공식에서 구한 값입니다. 딥러닝 제 6 공식은 다음과 같은 순서로 유도할 수 있습니다.

1. 딥러닝 제 5 공식의 그림을 복사합니다.
2. $y1 \rightarrow y1E$, $w1 \rightarrow w1E$, $w2 \rightarrow w2E$, $b1 \rightarrow b1E$,
 $y2 \rightarrow y2E$, $w3 \rightarrow w3E$, $w4 \rightarrow w4E$, $b2 \rightarrow b2E$ 로 변경합니다.
3. 화살표 방향을 반대로 합니다.

딥러닝 제 6 공식은 실제로 편미분을 연쇄적으로 적용하여 얻은 공식입니다. 편미분을 이용하여 역전파를 유도하는 방법은 부록을 참조합니다.

다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> w1E=y1E*x1
>>> w2E=y1E*x2
>>> b1E=y1E*1
>>> w3E=y2E*x1
>>> w4E=y2E*x2
>>> b2E=y2E*1
>>> (w1E,w2E,b1E)
(-16, -24, -8)
>>> (w3E,w4E,b2E)
(120, 180, 60)
```

07 딥러닝 제 7 공식 : 신경망 학습

다음은 [2입력 2출력] 인공 신경망의 신경망 학습을 나타내는 수식입니다.

$$\begin{aligned}w1 &-= lr * w1E \\w2 &-= lr * w2E \\b1 &-= lr * b1E \\w3 &-= lr * w3E \\w4 &-= lr * w4E \\b2 &-= lr * b2E\end{aligned}$$

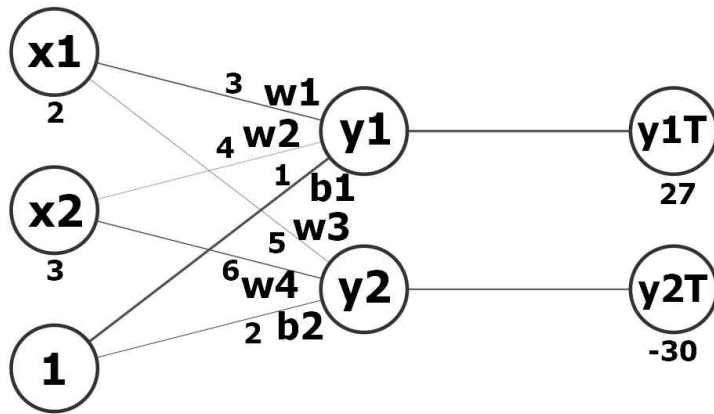
lr은 learning rate의 약자로 학습률을 의미합니다. w1E, w2E, w3E, w4E는 가중치 역전파 오차, b1E, b2E는 편향 역전파 오차로 딥러닝 제 6 공식에서 구한 값입니다.

다음과 같이 파이썬 셸에 명령을 주고 결과를 확인합니다.

```
>>> lr=0.01
>>> w1=w1-lr*w1E
>>> w2=w2-lr*w2E
>>> b1=b1-lr*b1E
>>> w3=w3-lr*w3E
>>> w4=w4-lr*w4E
>>> b2=b2-lr*b2E
>>> (w1,w2,b1)
(3.16, 4.24, 1.08)
>>> (w3,w4,b2)
(3.8, 4.2, 1.4)
```


딥러닝 반복 학습해 보기

지금까지의 과정을 반복해서 학습해 봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력값 x_1, x_2 는 각각 2, 3, 가중치 w_1, w_2 , 편향 b_1 은 각각 3, 4, 1, 가중치 w_3, w_4 , 편향 b_2 는 각각 5, 6, 2이고 목표값 y_{1T}, y_{2T} 는 각각 27, -30입니다. x_1, x_2, y_{1T}, y_{2T} 를 이용하여 $w_1, w_2, b_1, w_3, w_4, b_2$ 에 대해 학습을 수행해 봅니다.

*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

1. 다음과 같이 파이썬 스크립트를 작성합니다.


242_6.py

```
01 x1, x2 = 2, 3
02 w1, w2 = 3, 4
03 w3, w4 = 5, 6
04 b1, b2 = 1, 2
05 y1T, y2T = 27, -30
06 lr = 0.01
07
08 for epoch in range(200):
09
10     y1 = x1*w1 + x2*w2 + 1*b1
11     y2 = x1*w3 + x2*w4 + 1*b2
12     E = ((y1-y1T)**2 + (y2-y2T)**2) / 2
13     y1E = y1 - y1T
14     y2E = y2 - y2T
15     w1E = y1E*x1
16     w2E = y1E*x2
17     b1E = y1E*1
18     w3E = y2E*x1
19     w4E = y2E*x2
```

```

20     b2E = y2E*1
21     w1 = w1 - lr*w1E
22     w2 = w2 - lr*w2E
23     b1 = b1 - lr*b1E
24     w3 = w3 - lr*w3E
25     w4 = w4 - lr*w4E
26     b2 = b2 - lr*b2E
27
28     print(f'epoch = {epoch}')
29     print(f' y1 : {y1:.3f}')
30     print(f' y2 : {y2:.3f}')
31     print(f' w1 : {w1:.3f}')
32     print(f' w2 : {w2:.3f}')
33     print(f' b1 : {b1:.3f}')
34     print(f' w3 : {w3:.3f}')
35     print(f' w4 : {w4:.3f}')
36     print(f' b2 : {b2:.3f}')
37
38     if E < 0.0000001:
39         break

```

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

```

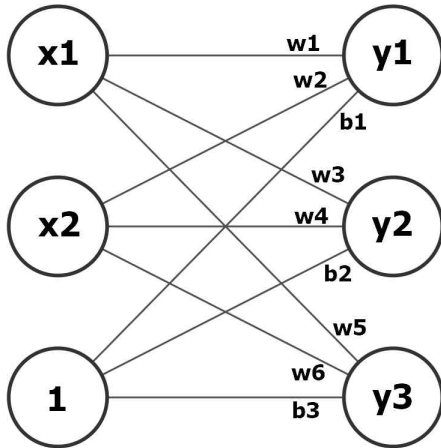
epoch = 79
y1 : 27.000
y2 : -30.000
w1 : 4.143
w2 : 5.714
b1 : 1.571
w3 : -3.571
w4 : -6.857
b2 : -2.286

```

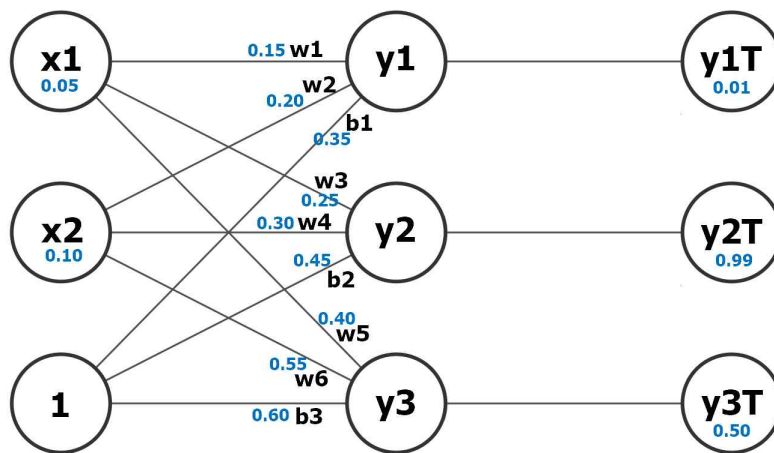
epoch 값이 79일 때 학습이 완료되는 것을 볼 수 있습니다. 가중치 w1, w2는 각각 4.143, 5.714, 편향 b1은 1.571, 가중치 w3, w4는 각각 -3.571, -6.857 편향 b2는 -2.286에 수렴합니다.

연습문제 1

1. 다음은 [2입력 3출력]의 인공 신경망입니다. 이 인공 신경망의 딥러닝 7 공식을 구합니다.

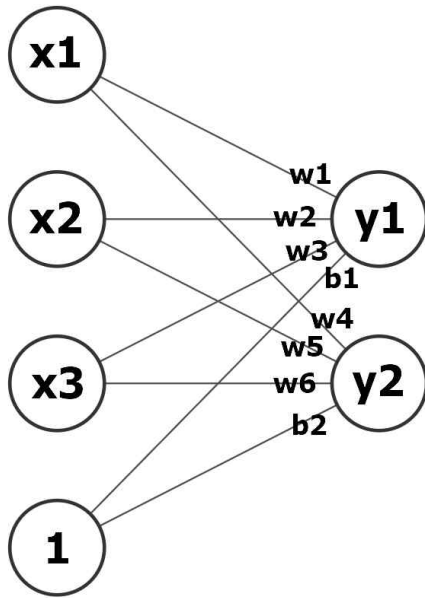


2. 앞에서 구한 딥러닝 7 공식을 이용하여 다음과 같이 초기화된 인공 신경망을 구현하고 학습시켜 봅시다.

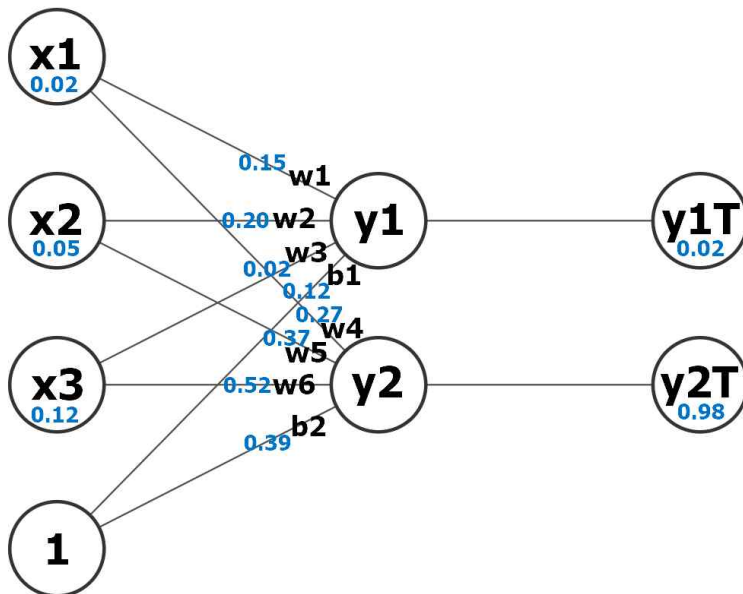


연습문제 2

1. 다음은 [3입력 2출력]의 인공 신경망입니다. 이 인공 신경망의 딥러닝 7 공식을 구합니다.



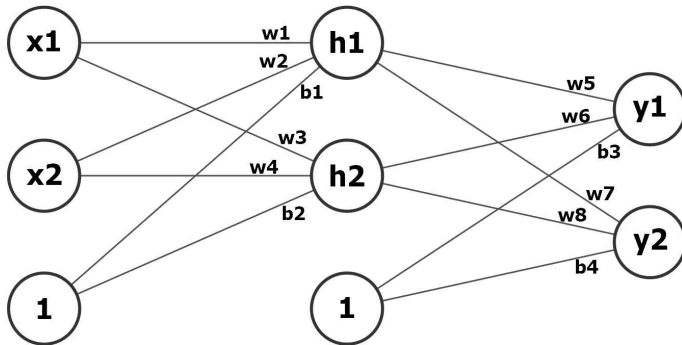
2. 앞에서 구한 딥러닝 7 공식을 이용하여 다음과 같이 초기화된 인공 신경망을 구현하고 학습시켜 봅시다.



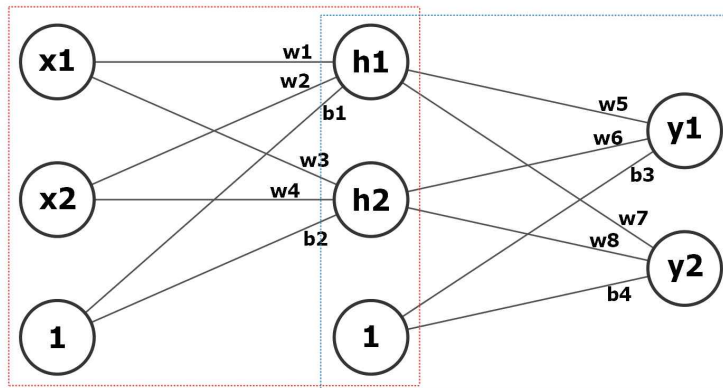
03 2입력 2은닉 2출력 인공 신경망

여기서는 앞에서 정리한 행렬 기반 딥러닝 7 공식을 이용하여 2입력 2은닉 2출력 인공 신경망을 학습시켜 봅니다.

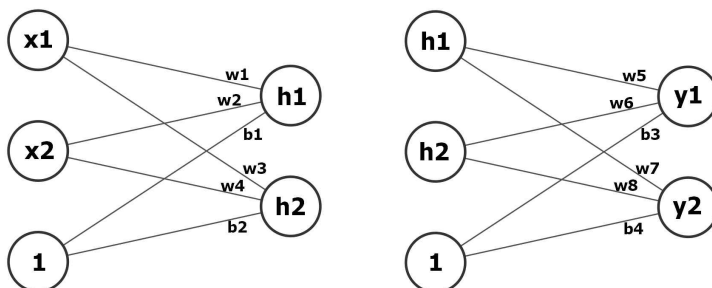
다음 그림은 [2입력 2은닉 2출력]으로 구성된 인공 신경망을 나타냅니다. 인공 신경망에서 입력층과 출력층 사이에 오는 층을 은닉층이라고 합니다.



[2입력 2은닉 2출력] 인공 신경망은 다음 그림과 같이 2개의 [2입력 2출력] 인공 신경으로 구성됩니다.



그래서 다음과 같이 [2입력 2출력] 인공 신경 2개로 나눌 수 있습니다.



1. 순전파

$$h1=x1*w1+x2*w2+b1$$

$$h2=x1*w3+x2*w4+b2$$

$$y1=h1*w5+h2*w6+b3$$

$$y2=h1*w7+h2*w8+b4$$

2. 오차

$$E=(y1-y1T)^2/2+(y2-y2T)^2/2$$

3. 역전파 오차

$$y1E=y1-y1T$$

$$y2E=y2-y2T$$

4. 입력 역전파

$$h1E=y1E*w5+y2E*w7$$

$$h2E=y1E*w6+y2E*w8$$

6. 가중치, 편향 역전파

$$w5E=y1E*h1$$

$$w6E=y1E*h2$$

$$w7E=y2E*h1$$

$$w8E=y2E*h2$$

$$b3E=y1E*1$$

$$b4E=y2E*1$$

$$w1E=h1E*x1$$

$$w2E=h1E*x2$$

$$w3E=h2E*x1$$

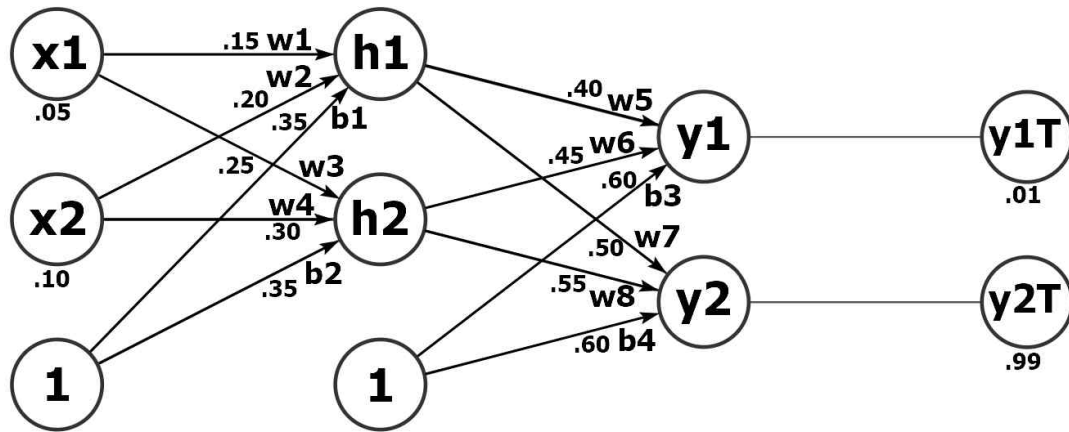
$$w4E=h2E*x2$$

$$b1E=h1E*1$$

$$b2E=h2E*1$$

인공 신경망 구현하기

지금까지 정리한 수식으로 신경망을 학습시켜 봅시다. 다음 그림을 살펴봅시다.



*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

1. 다음과 같이 예제를 작성합니다.

413.py

```
01 x1, x2 = 0.05, 0.10
02
03 w1, w2 = 0.15, 0.20
04 w3, w4 = 0.25, 0.30
05 b1, b2 = 0.35, 0.35
06
07 w5, w6 = 0.40, 0.45
08 w7, w8 = 0.50, 0.55
09 b3, b4 = 0.60, 0.60
10
11 y1T, y2T = 0.01, 0.99
12
13 lr = 0.01
14
15 for epoch in range(2000):
16
17     h1 = x1*w1 + x2*w2 + 1*b1
18     h2 = x1*w3 + x2*w4 + 1*b2
19
20     y1 = h1*w5 + h2*w6 + 1*b3
21     y2 = h1*w7 + h2*w8 + 1*b4
```


```

22
23     E = ((y1-y1T)**2 + (y2-y2T)**2) / 2
24
25     y1E = y1 - y1T
26     y2E = y2 - y2T
27
28     w5E = y1E*h1
29     w6E = y1E*h2
30     w7E = y2E*h1
31     w8E = y2E*h2
32     b3E = y1E*1
33     b4E = y2E*1
34
35     h1E = y1E*w5 + y2E*w7
36     h2E = y1E*w6 + y2E*w8
37
38     w1E = h1E*x1
39     w2E = h1E*x2
40     w3E = h2E*x1
41     w4E = h2E*x2
42     b1E = h1E*1
43     b2E = h2E*1
44
45     w5 = w5 - lr*w5E
46     w6 = w6 - lr*w6E
47     w7 = w7 - lr*w7E
48     w8 = w8 - lr*w8E
49     b3 = b3 - lr*b3E
50     b4 = b4 - lr*b4E
51
52     w1 = w1 - lr*w1E
53     w2 = w2 - lr*w2E
54     w3 = w3 - lr*w3E
55     w4 = w4 - lr*w4E
56     b1 = b1 - lr*b1E
57     b2 = b2 - lr*b2E
58
59     print(f'epoch = {epoch}')
60     print(f' y1 : {y1:.3f}')
61     print(f' y2 : {y2:.3f}')

```



```
62
63     if E < 0.0000001:
64         break
```

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

```
epoch = 665
y1 : 0.010
y2 : 0.990
```

학습 회수에 따라 y1, y2값이 바뀌는 것을 확인합니다. y1, y2값이 각각 0.01, 0.99에 가까워지는 것을 확인합니다. 입력값 0.05, 0.10에 대해 목표값은 0.01, 0.99입니다.

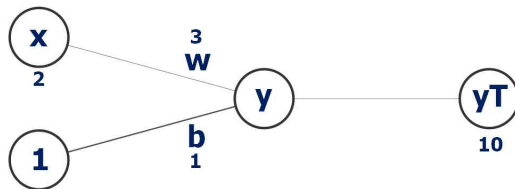
05 teachable machine 사용해 보기

06 텐서플로우로 딥러닝 7 공식 구현하기

여기서는 딥러닝 7 공식을 Tensorflow로 구현해 보며 텐서플로우의 내부적인 동작을 이해해 봅니다. Tensorflow의 내부 동작을 잘 이해하여 Tensorflow의 활용도를 높일 수 있도록 합니다.

01 1입력 1출력 인공 신경 구현하기

다음은 [입력1 출력1]의 인공 신경 학습에 사용할 행렬을 나타냅니다.



$$\begin{aligned} X &= [2] \\ T &= [10] \\ W &= [3] \\ B &= [1] \end{aligned}$$

X를 입력 값으로, T를 목표 값으로 하여 가중치 W와 편향 B에 대해 Tensorflow로 학습해 봅니다.

1. 다음과 같이 예제를 수정합니다.

dnn_test_1_1.py

```
01 import tensorflow as tf
02 import numpy as np
03
04 X=np.array([[2]]) # 입력데이터
05 YT=np.array([[10]]) # 목표데이터(라벨)
06 W=np.array([[3]]) # 가중치
07 B=np.array([1]) # 편향
08
09 model=tf.keras.Sequential([
10     tf.keras.Input(shape=(1,)), # 입력 층 노드의 개수를 1로 설정합니다.
11     tf.keras.layers.Dense(1) # 출력 층 노드의 개수를 1로 설정합니다.
12 ]) # 신경망 모양 결정(W, B 내부적 준비)
13
14 model.layers[0].set_weights([W,B])
15
16 model.compile(optimizer='sgd', # 7공식, 학습
17               loss='mse') # 2공식, 오차계산
18
19 Y=model.predict(X)
20 print(Y)
```

```

21
22 model.fit(X,YT,epochs=1000)
23
24 print('W=',model.layers[0].get_weights()[0])
25 print('B=',model.layers[0].get_weights()[1])
26
27 Y=model.predict(X)
28 print(Y)

```

01 : import문을 이용하여 tensorflow 모듈을 tf라는 이름으로 불러옵니다. tensorflow 모듈은 구글에서 제공하는 인공 신경망 라이브러리입니다.

02 : import문을 이용하여 numpy 모듈을 np라는 이름으로 불러옵니다. numpy 모듈은 행렬 계산을 편하게 해주는 라이브러리입니다. 인공 신경망은 일반적으로 행렬 계산식으로 구성하게 됩니다.

04~06 : X, YT, W 변수를 이차 배열의 numpy 행렬로 초기화합니다.

07 : B 변수를 일차 배열의 numpy 벡터로 초기화합니다.

09~12 : tf.keras.Sequential 클래스를 이용하여 인공 신경망을 생성합니다.

10 : tf.keras.Input 클래스를 이용하여 입력 층을 생성합니다. 입력 노드의 개수를 1로 설정합니다.

11 : tf.keras.layers.Dense 클래스를 이용하여 신경 망 층을 생성합니다. 출력 노드 1개로 구성된 인공 신경망 층을 생성합니다.

14 : 인공 신경망에 임의로 설정된 가중치와 편향을 W, B로 변경합니다.


16~17 : model.compile 함수를 호출하여 내부적으로 인공 신경망을 구성합니다. 인공 신경망을 구성할 때에는 2개의 함수를 정해야 합니다. loss 함수와 optimizer 함수, 즉, 손실 함수와 최적화 함수를 정해야 합니다. 손실 함수로는 tf.keras.losses.MeanSquaredError 함수를 사용하고 최적화 함수는 확률적 경사 하강(sgd : stochastic gradient descent) 함수인 tf.keras.optimizers.SGD를 사용합니다.

19 : model.predict 함수를 호출하여 인공 신경망을 사용합니다. 이 과정은 순전파를 수행하는 예측을 수행하는 과정입니다.

22 : model.fit 함수를 호출하여 인공 신경망에 대한 학습을 시작합니다. fit 함수에는 X, YT 데이터가 입력이 되는데 인공 신경망을 X, YT 데이터에 맞도록 학습한다는 의미를 갖습니다. fit 함수에는 학습을 몇 회 수행할지도 입력해 줍니다. epochs는 학습 횟수를 의미하며, 여기서는 1000회 학습을 수행하도록 합니다.

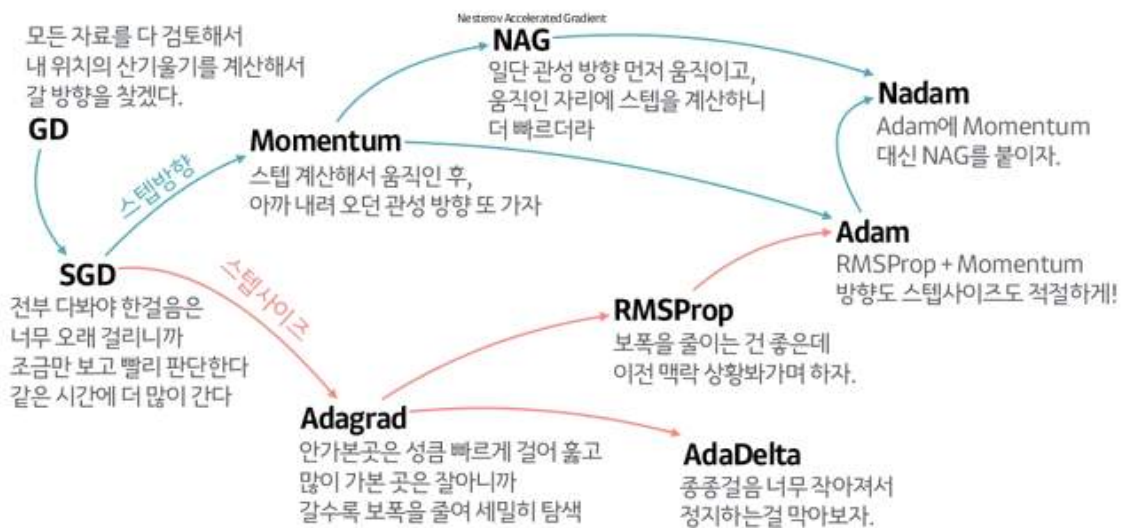
24~25 : 학습이 끝난 W, B 값을 출력해 봅니다.

27 : 학습이 끝난 신경망을 이용하여 예측을 수행해 봅니다.

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

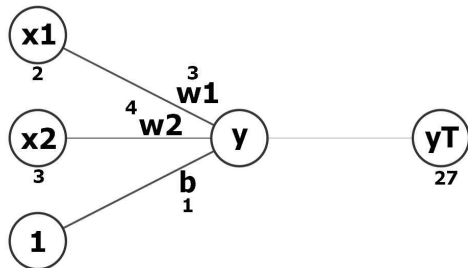
```
Epoch 1000/1000
1/1 [=====] - 0s 1ms/step - loss: 8.1855e-12
W= [[4.1999974]]
B= [[1.6000019]]
1/1 [=====] - 0s 17ms/step
[[9.999997]]
```

*** optimizer 살펴보기



02 2입력 1출력 인공 신경 구현하기

다음은 입력2 출력1의 인공 신경 학습에 사용할 행렬을 나타냅니다.



$$X = \begin{bmatrix} 2 & 3 \end{bmatrix}$$

$$T = \begin{bmatrix} 27 \end{bmatrix}$$

$$W = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 \end{bmatrix}$$


X를 입력 값으로, T를 목표 값으로 하여 가중치 W와 편향 B에 대해 Tensorflow로 학습해 봅니다.

1. 다음과 같이 예제를 수정합니다.

dnn_test_2_1.py

```
01~03 # 이전 예제와 같습니다.
04 X=np.array([[2,3]]) # 입력데이터
05 YT=np.array([[27]]) # 목표데이터(라벨)
06 W=np.array([[3],[4]]) # 가중치
07 B=np.array([1]) # 편향
08
09 model=tf.keras.Sequential([
10     tf.keras.Input(shape=(2,)),
11     tf.keras.layers.Dense(1)
12 ]) # 신경망 모양 결정(W, B 내부적 준비)
13~끝 # 이전 예제와 같습니다.
```

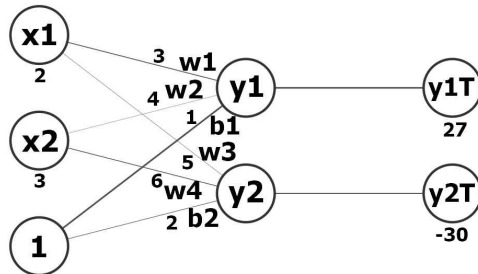
10 : 입력 층 노드의 개수를 2로 변경합니다.

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

```
Epoch 1000/1000
1/1 [=====] - 0s 1ms/step - loss: 3.6380e-12
W= [[4.1428566]
     [5.714285 ]]
B= [1.5714294]
1/1 [=====] - 0s 17ms/step
[[26.999998]]
```


03 2입력 2출력 인공 신경망 구현하기

다음은 [입력2 출력2]의 인공 신경망 학습에 사용할 행렬을 나타냅니다.



$$X = \begin{bmatrix} 2 & 3 \end{bmatrix} \quad \textcircled{9}$$
$$T = \begin{bmatrix} 27 & -30 \end{bmatrix} \quad \textcircled{10}$$
$$W = \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix} \quad \textcircled{11}$$
$$B = \begin{bmatrix} 1 & 2 \end{bmatrix} \quad \textcircled{12}$$


X를 입력 값으로, T를 목표 값으로 하여 가중치 W와 편향 B에 대해 Tensorflow로 학습해 봅니다.

1. 다음과 같이 예제를 작성합니다.

dnn_test_2_2.py

```
01~03 # 이전 예제와 같습니다.
04 X=np.array([[2,3]]) # 입력데이터
05 YT=np.array([[27,-30]]) # 목표데이터(라벨)
06 W=np.array([[3,5],[4,6]]) # 가중치
07 B=np.array([1,2]) # 편향
08
09 model=tf.keras.Sequential([
10     tf.keras.Input(shape=(2,)),
11     tf.keras.layers.Dense(2)
12 ]) # 신경망 모양 결정(W, B 내부적 준비)
13~끝 # 이전 예제와 같습니다.
```

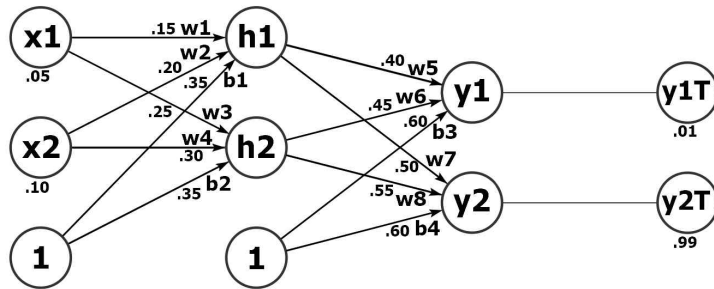
11 : 출력 층 노드의 개수를 2로 변경합니다.

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

```
Epoch 1000/1000
1/1 [=====] - 0s 1ms/step - loss: 3.2742e-11
W= [[ 4.1428556 -3.5714262]
     [ 5.714284  -6.857143 ]]
B= [ 1.5714294 -2.2857132]
1/1 [=====] - 0s 17ms/step
[[ 26.999994 -29.999994]]
```

04 2입력 2은닉 2출력 인공 신경망 구현하기

다음은 [입력2 은닉2 출력2]의 인공 신경망을 나타냅니다.



1. 다음과 같이 예제를 작성합니다.

dnn_test_2_2_2.py

```
01 import tensorflow as tf
02 import numpy as np
03
04 X=np.array([[.05,.10]]) # 입력데이터 <=
05 YT=np.array([[.01,.99]]) # 목표데이터(라벨)
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),
14     tf.keras.layers.Dense(2)
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='mse') # 2공식, 오차계산
22
23 Y=model.predict(X)
24 print(Y)
25
```

```

26 model.fit(X,YT,epochs=1000)
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)

```

12 : 입력 층 노드의 개수를 2로 설정합니다.


13 : 은닉 층 노드의 개수를 2로 설정합니다.

14 : 출력 층 노드의 개수를 2로 설정합니다.

17 : 인공 신경망 은닉 층에 임의로 설정된 가중치와 편향을 W, B로 변경합니다.

18 : 인공 신경망 출력 층에 임의로 설정된 가중치와 편향을 W2, B2로 변경합니다.

28~31: print 함수를 호출하여 학습이 수행된 W, B, W2, B2 행렬값을 출력합니다.

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

```

Epoch 1000/1000
1/1 [=====] - 0s 2ms/step - loss: 5.9980e-11
W= [[0.14315726 0.24180055]
     [0.18631484 0.2836011 ]]
B= [0.2131526 0.18600278]
W2= [[0.20264395 0.53349453]
      [0.2525947 0.5828034 ]]
B2= [-0.09561112 0.73054373]
1/1 [=====] - 0s 17ms/step
[[0.01000983 0.9899955 ]]

```