



셋째마당

딥러닝의 시작, 신경망

9장 오차 역전파에서 딥러닝으로

- 1 딥러닝의 태동, 오차 역전파
- 2 활성화 함수와 고급 경사 하강법
- 3 속도와 정확도 문제를 해결하는 고급 경사 하강법

오차 역전파에서 딥러닝으로

- 오차 역전파에서 딥러닝으로
 - 해결되지 않던 XOR 문제를 다층 퍼셉트론으로 해결
 - 한 가지 문제를 만남
 - 은닉층에 포함된 가중치를 업데이트할 방법이 없었던 것
 - 이는 기나긴 인공지능의 겨울을 지나 오차 역전파라는 방법을 만나고 나서야 해결
 - 오차 역전파는 후에 지금 우리가 아는 딥러닝의 탄생으로 이어짐
 - 오차 역전파는 어떻게 해서 은닉층의 오차를 업데이트할 수 있었을까?

오차 역전파에서 딥러닝으로

● 오차 역전파에서 딥러닝으로

- 여기서 두 가지 길을 여러분께 제시
- 하나는 오차 역전파의 개념을 이해하고 넘어가는 것, 나머지 하나는 개념과 함께 계산 방법까지 익히고 다음으로 넘어가는 것
- 이 책은 두 가지가 모두 준비되어 있습니다만, 오차 역전파의 개념만 알아도 앞으로 우리가 배울 과정을 마치는 데는 아무런 문제가 없음
- 개념을 숙지하는 것을 목표로 학습할 분들은 이 장의 내용만 익히면 됨
- 반면, 딥러닝 알고리즘을 더 깊이 이해하고 싶거나, 텐서플로 같은 자동화 라이브러리에 맡기지 않고 직접 코딩을 해야 한다면 이 장의 학습에 이어 심화 학습 1에 준비되어 있는 **오차 역전파의 계산법** 편을 숙지하기 바람



1 딤러닝의 태동, 오차 역전파

1 딥러닝의 태동, 오차 역전파

● 딥러닝의 태동, 오차 역전파

- 앞서 XOR 문제를 해결했지만, 입력 값과 출력 값을 알고 있는 상태에서 **가중치(w)**와 **바이어스(b)**를 미리 알아본 후 이를 집어넣었음
- 이것은 '모델링'이라고 할 수 없음
- 둘째 마당의 회귀 모델에서 살펴본 바와 같이 우리가 원하는 것은 데이터를 통해 스스로 가중치를 조절하는 '학습'의 실현
- 오차 역전파 방법은 어떻게 해서 숨겨진 은닉층의 가중치를 업데이트할 수 있었을까?
- 이를 설명하기 위해 다시 경사 하강법 이야기로 돌아가 보자

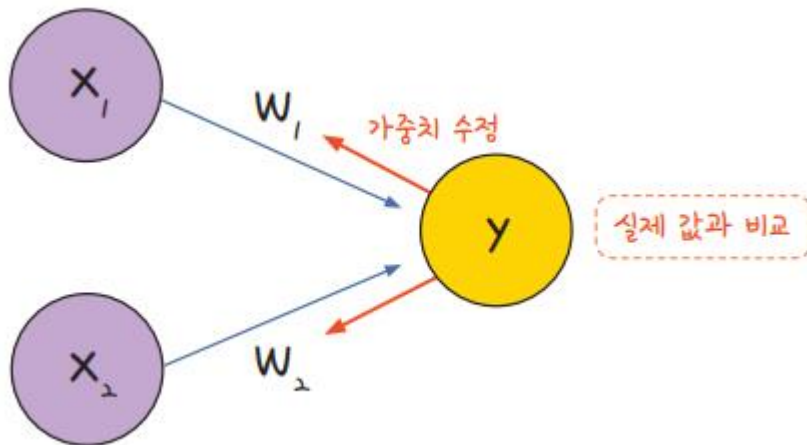
1 딥러닝의 태동, 오차 역전파

● 딥러닝의 태동, 오차 역전파

- 경사 하강법은 임의의 가중치를 선언하고 결괏값을 이용해 오차를 구한 후 이 오차가 최소인 지점으로 계속해서 조금씩 이동시키는 것
- 이 오차가 최소인 지점은 미분했을 때 기울기가 0이 되는 지점이라고 했음
- 지금 이야기하고 있는 경사 하강법은 '단일 퍼셉트론', 즉 입력층과 출력층만 존재할 때 가능
- 은닉층이 생기면서 우리는 두 번의 경사 하강법을 실행해야 함
- 즉, 그림 9-1과 같이 가중치를 한 번 수정하면 되던 작업이 그림 9-2와 같이 가중치를 두 번 수정해야 하는 것

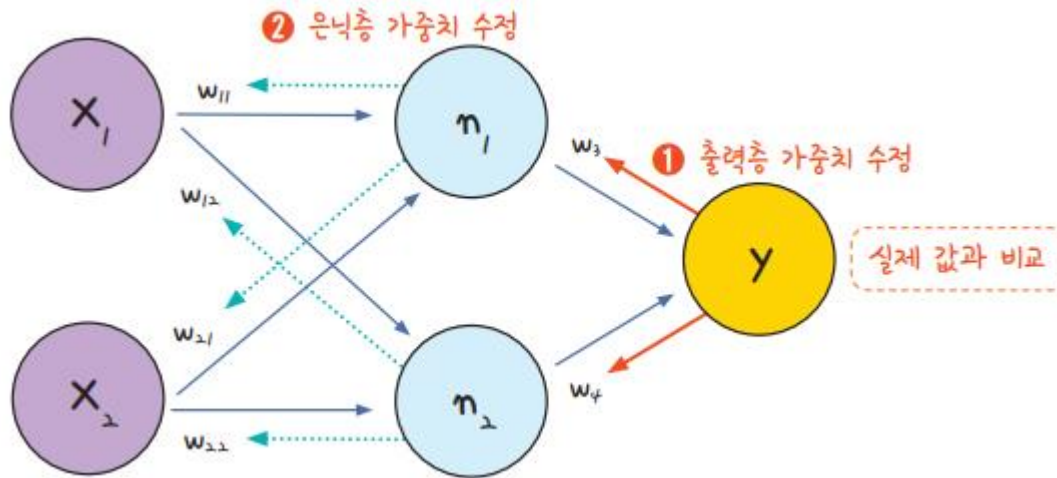
1 딥러닝의 태동, 오차 역전파

▼ 그림 9-1 | 단일 퍼셉트론에서 오차 수정



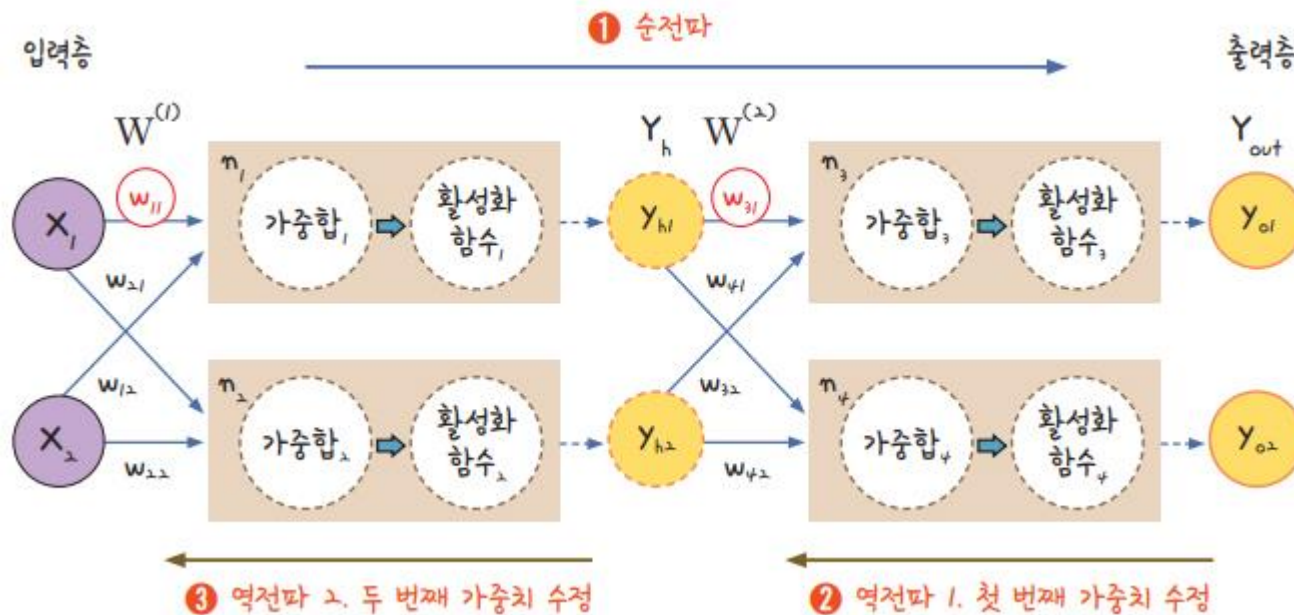
1 딥러닝의 태동, 오차 역전파

▼ 그림 9-2 | 다층 퍼셉트론에서 오차 수정

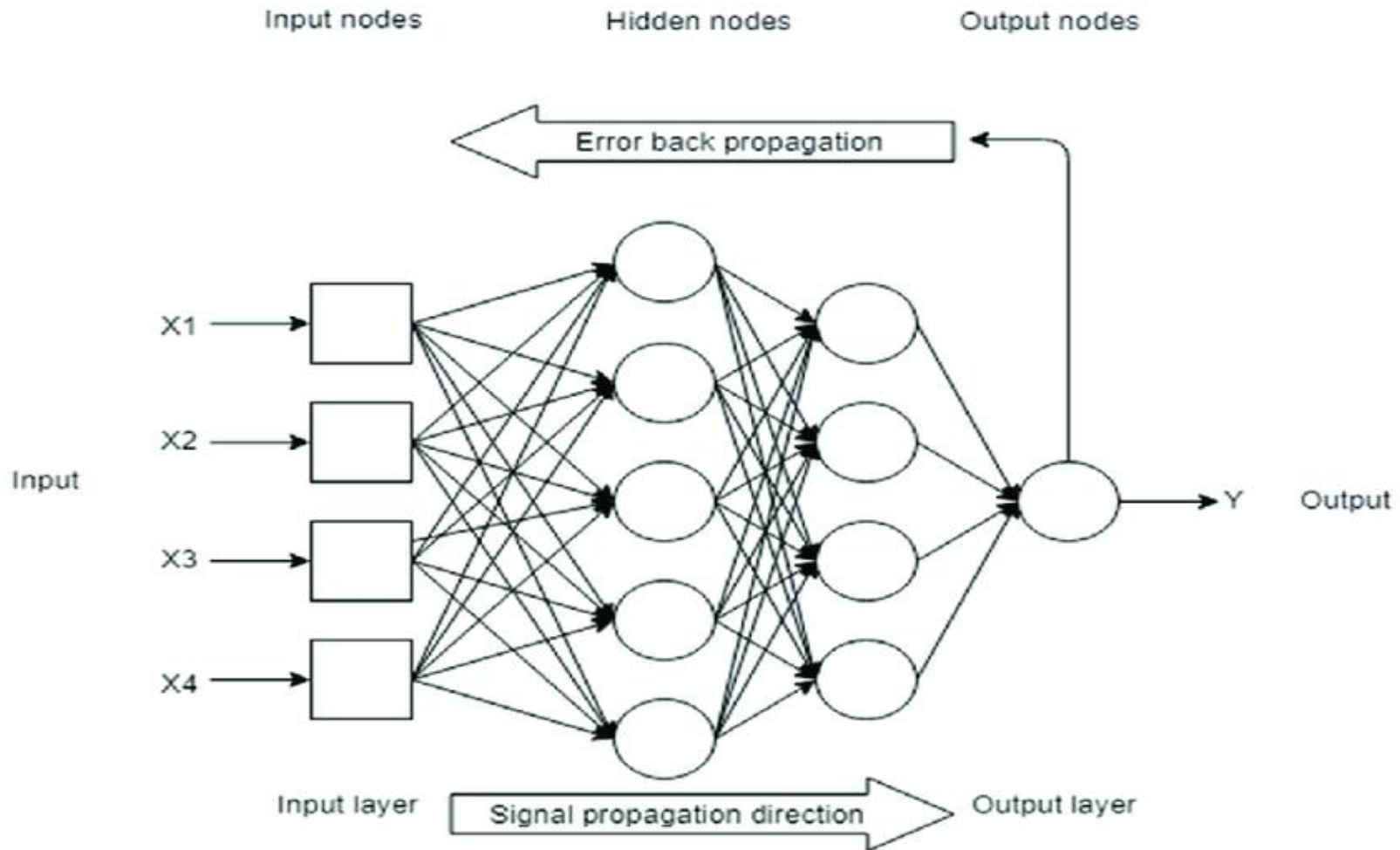


1 딥러닝의 태동, 오차 역전파

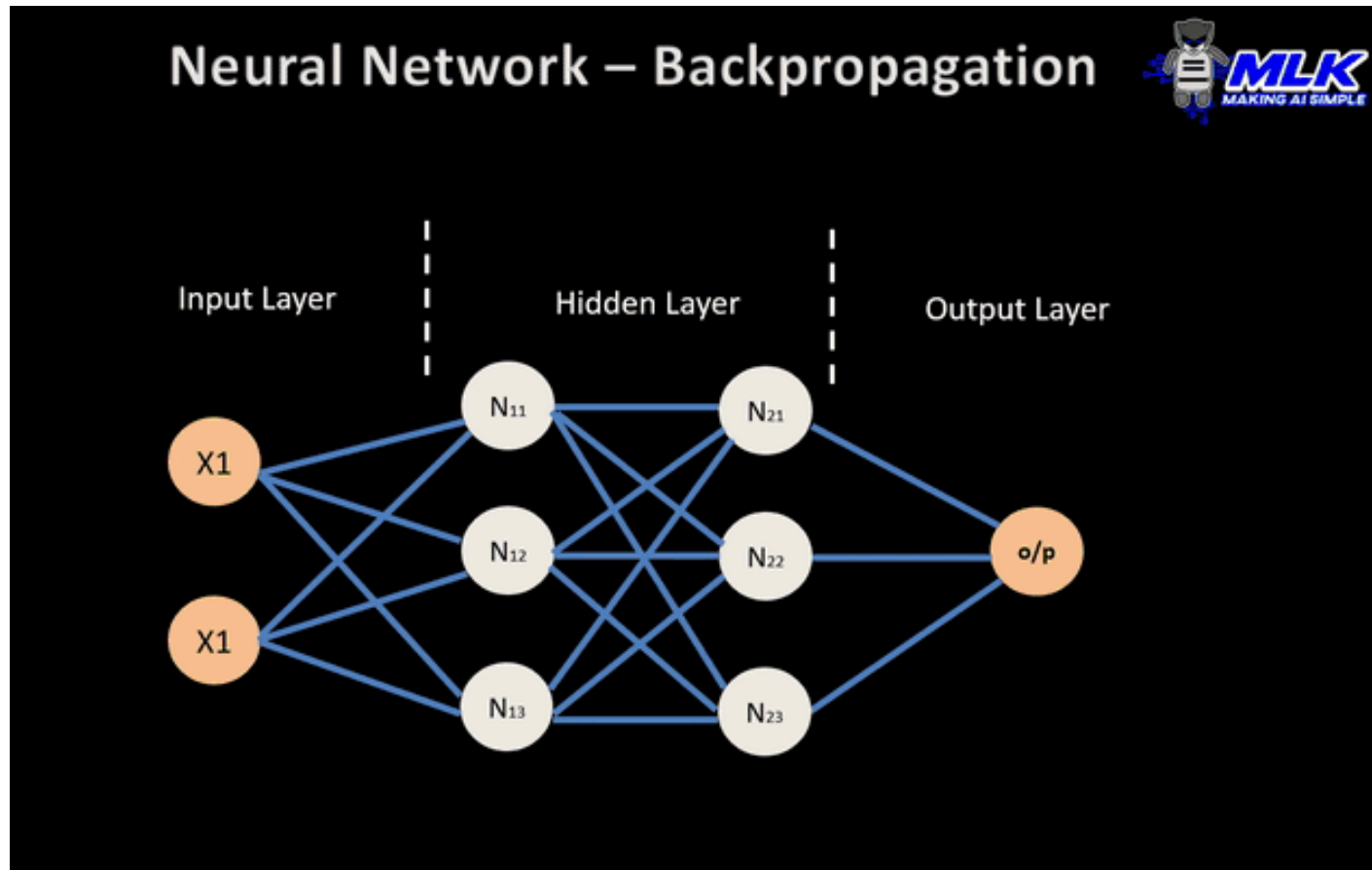
▼ 그림 9-3 | 오차 역전파의 개념



1 딥러닝의 태동, 오차 역전파



1 딥러닝의 태동, 오차 역전파



참조 : machinelearningknowledge.ai

1 딥러닝의 태동, 오차 역전파

- 딥러닝의 태동, 오차 역전파

- 먼저 ❶ 처럼 한 번의 순전파가 일어남
- 이 과정을 통해 각 가중치의 초깃값이 정해짐
- 이 초깃값의 가중치로 만들어진 값과 실제 값을 비교해 출력층의 오차를 계산
- 목표는 이때 계산된 출력층의 오차를 최소화시키는 것
- 이를 위해 ❷ 첫 번째 가중치를 수정하는 과정과 ❸ 두 번째 가중치를 수정하는 과정이 이어짐

1 딥러닝의 태동, 오차 역전파

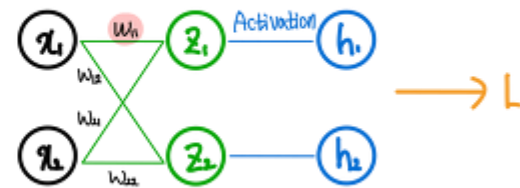
● 딥러닝의 태동, 오차 역전파

- 예를 들어 첫 번째 가중치 중 하나인 w_{31} 을 업데이트한다고 하자
- 경사 하강법에서 배운 대로 w_{31} 을 업데이트하기 위해서는 오차 공식을 구하고 w_{31} 에 대해 편미분해야 함
- 합성 함수의 미분이므로 **체인 룰**을 적용해 편미분을 구하고 이를 이용해 w_{31} 을 업데이트
- 이제 두 번째 가중치를 업데이트할 차례
- 예를 들어 w_{11} 을 업데이트한다고 하면 마찬가지로 오차 공식을 구하고 w_{11} 에 대해 편미분하면 됨
- 여기서 문제가 생김
- 앞서 우리는 출력층의 결과와 실제 값을 비교해 오차를 얻었음
- 은닉층은 겉으로 드러나지 않으므로 그 값을 알 수 없음
- 오차를 구할 만한 적절한 출력 값도 없다는 것

1 오차 역전파, 체인 룰(Chain Rule)

● 체인 룰(Chain Rule)

- 인공 신경망이 순전파 과정을 진행하여 예측값과 실제값의 오차를 계산하였을 때, 역전파 과정에서 체인 룰을 사용하여 가중치를 업데이트 하는 방법

	일상생활에서의 Chain Rule	딥러닝에서의 Chain Rule
현상	어제 밤 늦게까지 술먹음 → 다음날 피곤함 → 여자친구에게 재응답 → 차임	
관계	$A \rightarrow B \rightarrow C \rightarrow D$	$w_{11} \rightarrow z_1 \rightarrow h_1 \rightarrow L$
Chain Rule	$\frac{\partial D}{\partial A} = \frac{\partial B}{\partial A} \times \frac{\partial C}{\partial B} \times \frac{\partial D}{\partial C}$	$\frac{\partial L}{\partial w_{11}} = \frac{\partial z_1}{\partial w_{11}} \times \frac{\partial h_1}{\partial z_1} \times \frac{\partial L}{\partial h_1}$

1 오차 역전파, 체인 룰(Chain Rule)

$$z = g(y), y = f(x)$$

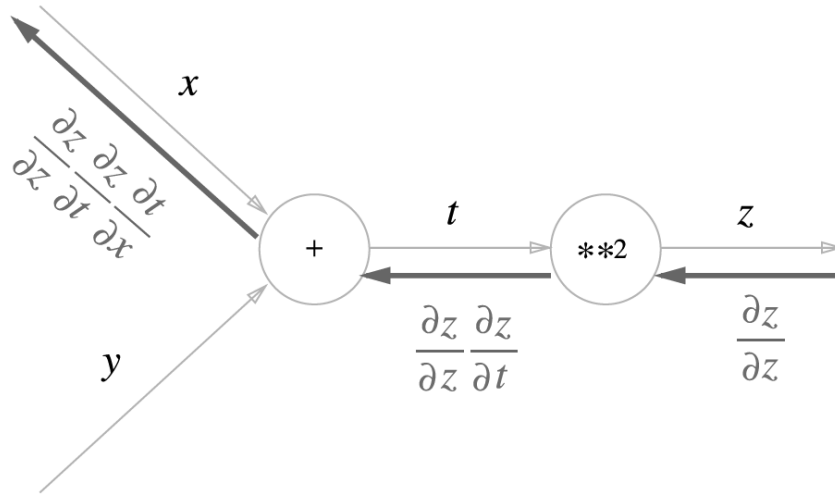
미분

$$\rightarrow (g \circ f)'(x) = g'(y)f'(x)$$

$$z = (g \circ f)(x) = g(f(x))$$

↓ Chain rule

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$



1 딥러닝의 태동, 오차 역전파

● 딥러닝의 태동, 오차 역전파

- 이 문제는 다시 출력층의 오차를 이용하는 것으로 해
- w_{31} 의 경우 y_{o1} 의 오차만 필요했었지만, w_{11} 은 y_{o1} 과 y_{o2} 가 모두 관여되어 있음
- 오차 두 개를 모두 계산해 이를 편미분하게 됨
- 물론 계산식은 조금 더 복잡해짐
- 이 과정을 마치면 첫 번째 가중치 업데이트 공식과 두 번째 가중치 업데이트 공식이 다음과 같이 정리

$$\text{첫 번째 가중치 업데이트 공식} = \underline{(y_{o1} - y_{\text{실제 값}}) \cdot y_{o1}(1 - y_{o1})} \cdot y_{h1}$$

$$\text{두 번째 가중치 업데이트 공식} = \underline{(\delta y_{o1} \cdot w_{31} + \delta y_{o2} \cdot w_{41})y_{h1}(1 - y_{h1})} \cdot x_1$$

1 딥러닝의 태동, 오차 역전파

● 딥러닝의 태동, 오차 역전파

- 이 공식 중 밑줄 친 부분을 잘 보면 두 식 모두 'out(1-out)' 형태를 취하고 있다는 것을 알 수 있음
- 이를 델타식이라고 함
- 은닉층의 숫자가 늘어도 이러한 형태가 계속해서 나타나게 되므로, 이를 이용해 깊은 층의 계산도 할 수 있게 됨
- 이렇게 깊은 층을 통해 학습할 수 있는 계기가 마련되면서 드디어 **딥러닝**이 태동하게 된 것
- 오차 역전파 과정을 순수 파이썬 코딩으로 만든 예제가 '심화 학습 2. 파이썬 코딩으로 짜 보는 신경망'에 준비되어 있음
- 우리는 텐서플로를 이용해 이 과정을 진행하므로 여기에 삽입하지는 않겠음
- 모든 과정이 파이썬 코딩으로 어떻게 표현되는지 공부할 분들은 참고

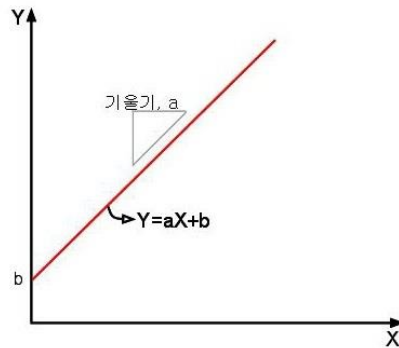


2 활성화 함수와 고급 경사 하강법

선형(linear), 비선형(non-linear)

● 선형(linear)

- 그래프의 형태가 1개의 직선으로 표현된다는 뜻



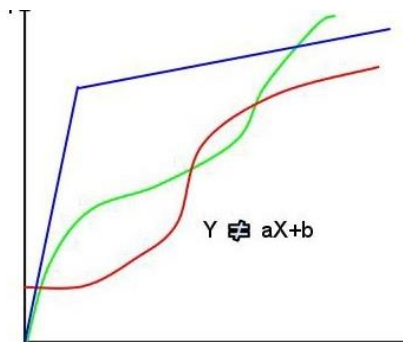
함수 f 에 대해,

- 가산성(Additivity), 즉, 임의의 수 x, y 에 대해 $f(x + y) = f(x) + f(y)$ 가 항상 성립하고
- 동차성(Homogeneity), 즉, 임의의 수 x 와 α 에 대해 $f(\alpha x) = \alpha f(x)$ 가 항상 성립할 때

함수 f 는 선형이라고 한다.

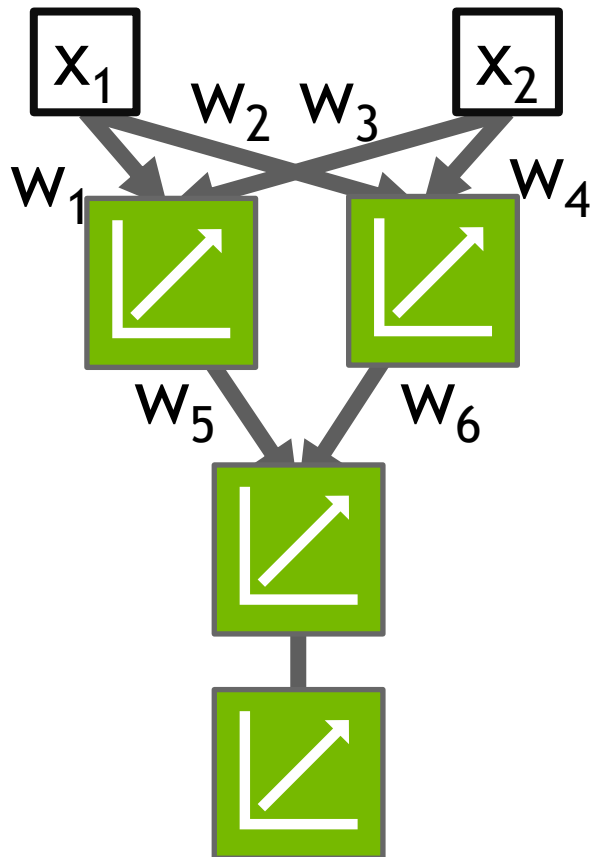
● 비선형(non-linear)

- 1개의 직선으로 표현되지 않는 모든 형태로 1차 방정식으로 표현되지 않는 모든 형태



활성화 함수(Activation Function)을 사용하는 이유

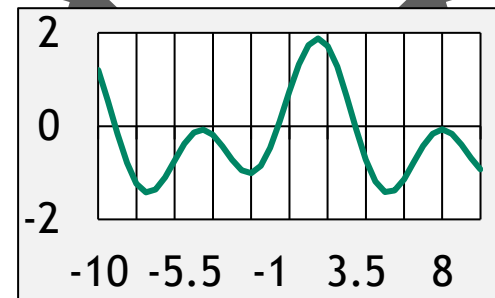
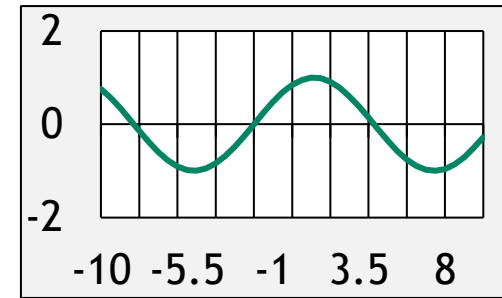
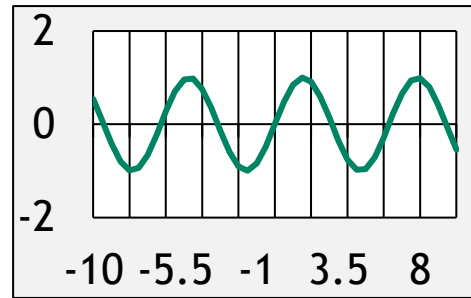
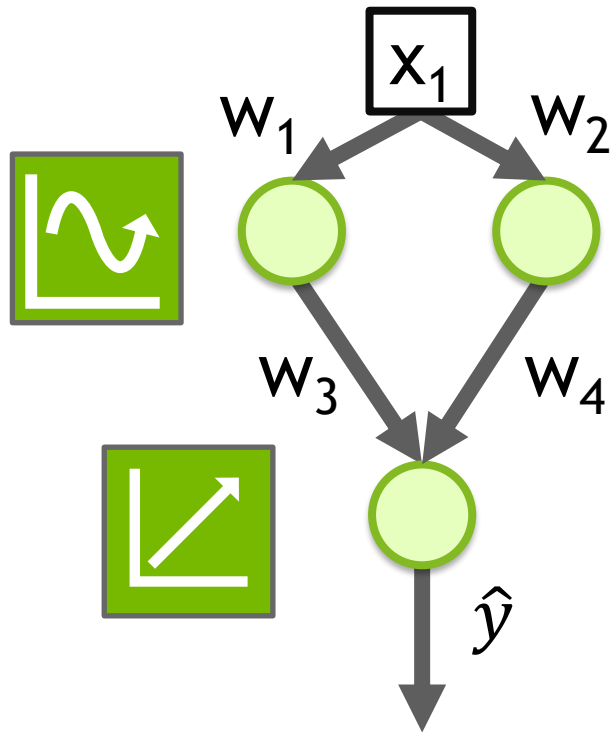
- 딥뉴럴 네트워크에서 임의의 함수를 통해 복잡한 관계를 찾기 위해 비선형성이 필요



- 더 많은 변수로의 확장
- 뉴런 간의 연결
- 모든 회귀가 선형이면 출력 결과도 선형 회귀
 - 아무리 층이 많아도 하나의 선형 층 임

활성화 함수(Activation Function)을 사용하는 이유

- 깊은 층을 쌓기 위해서는 비선형 함수가 필요함
- 심층 신경망(Deep Neural Network)은 비선형(non-linear) 시스템 임

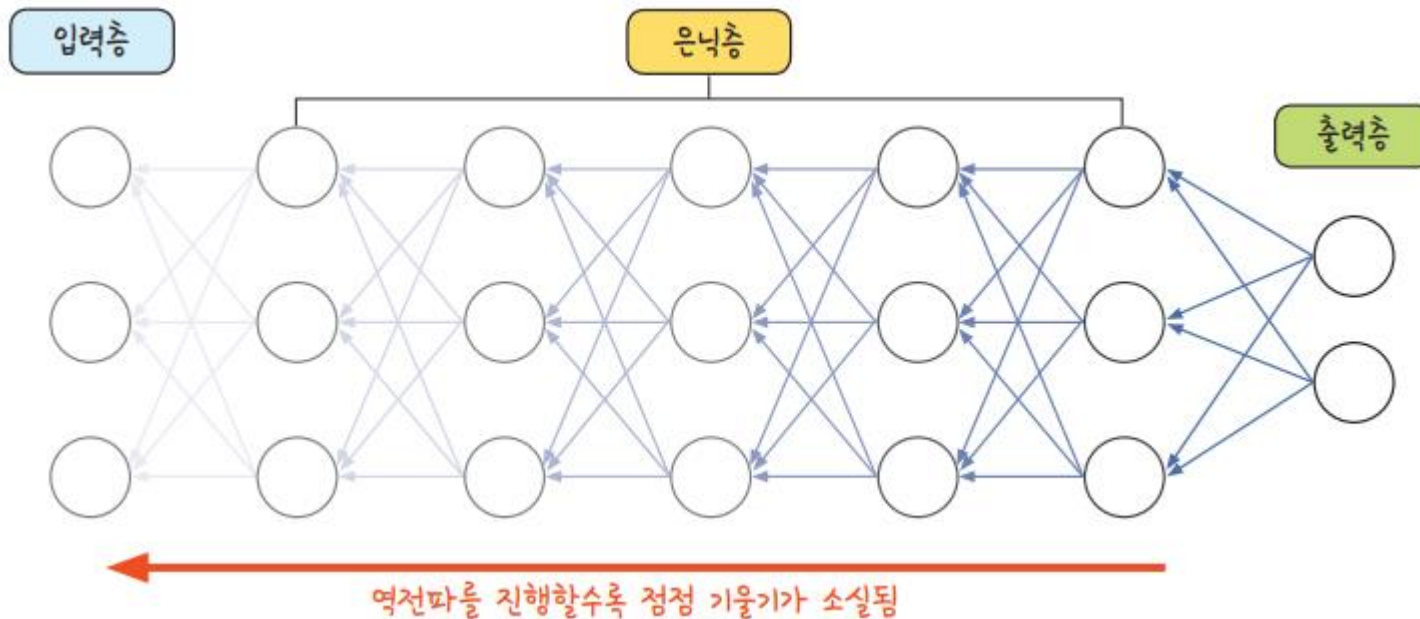


2 활성화 함수와 고급 경사 하강법

● 활성화 함수와 고급 경사 하강법

- 앞서 델타식을 이용해 깊은 신경망의 계산이 가능해졌음을 이야기했음
- 이제 수많은 층을 연결해 학습하면 여러 난제를 해결하는 인공지능이 완성될 것 같아 보임
- 아직 한 가지 문제가 더 남아 있음

▼ 그림 9-4 | 기울기 소실 문제 발생



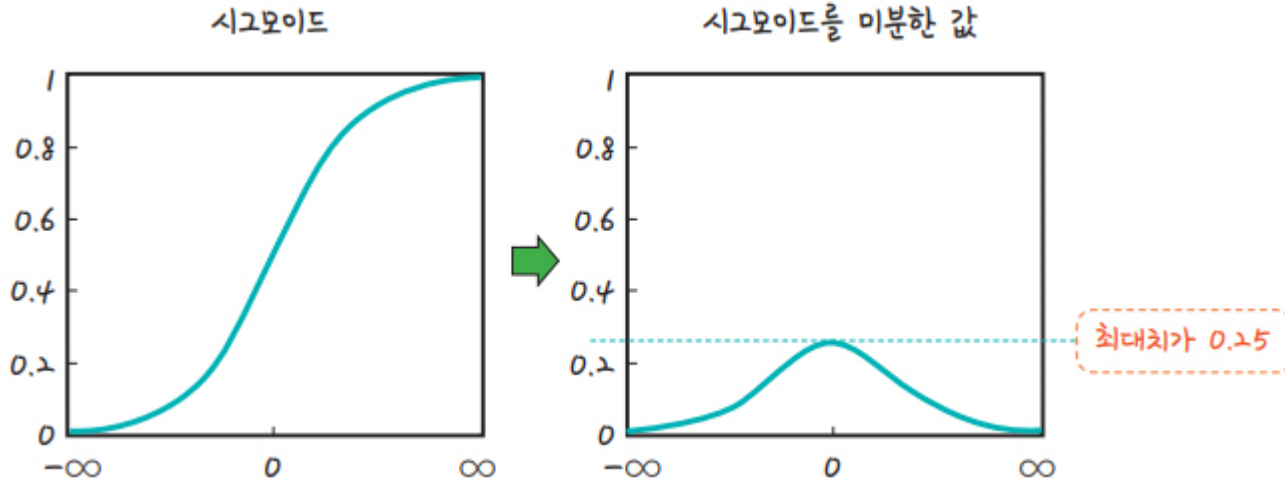
2 활성화 함수와 고급 경사 하강법

● 활성화 함수와 고급 경사 하강법

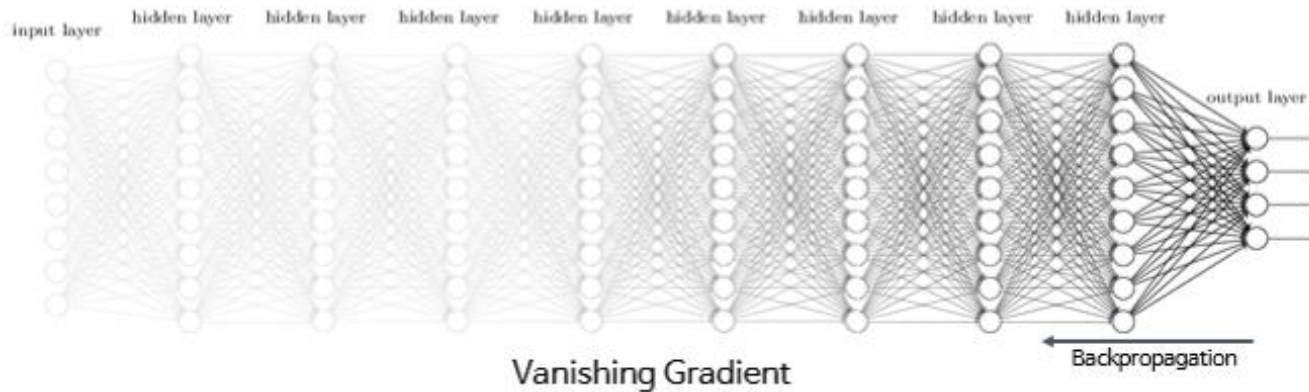
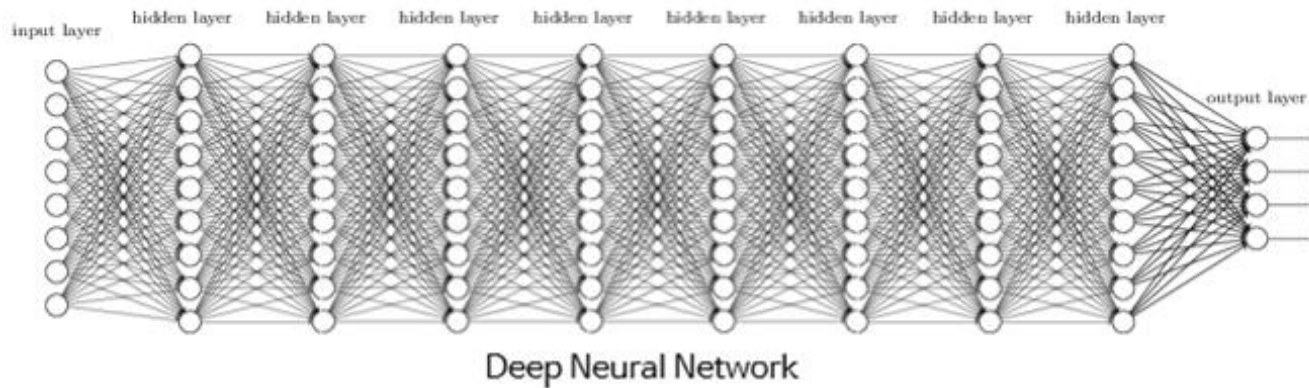
- 그림 9-4와 같이 깊은 층을 만들어 보니 출력층에서 시작된 가중치 업데이트가 처음 층까지 전달되지 않는 현상이 생기는 문제가 발견
- 이는 활성화 함수로 사용된 **시그모이드 함수**의 특성 때문임
- 그림 9-5와 같이 **시그모이드 함수를 미분하면 최대치는 0.25**
- 1보다 작으므로 계속 곱하다 보면 0에 가까워짐
- 여러 층을 거칠수록 기울기가 사라져 가중치를 수정하기 어려워지는 것

2 활성화 함수와 고급 경사 하강법

▼ 그림 9-5 | 시그모이드의 미분



가중치 소실 문제 (Vanishing gradient)

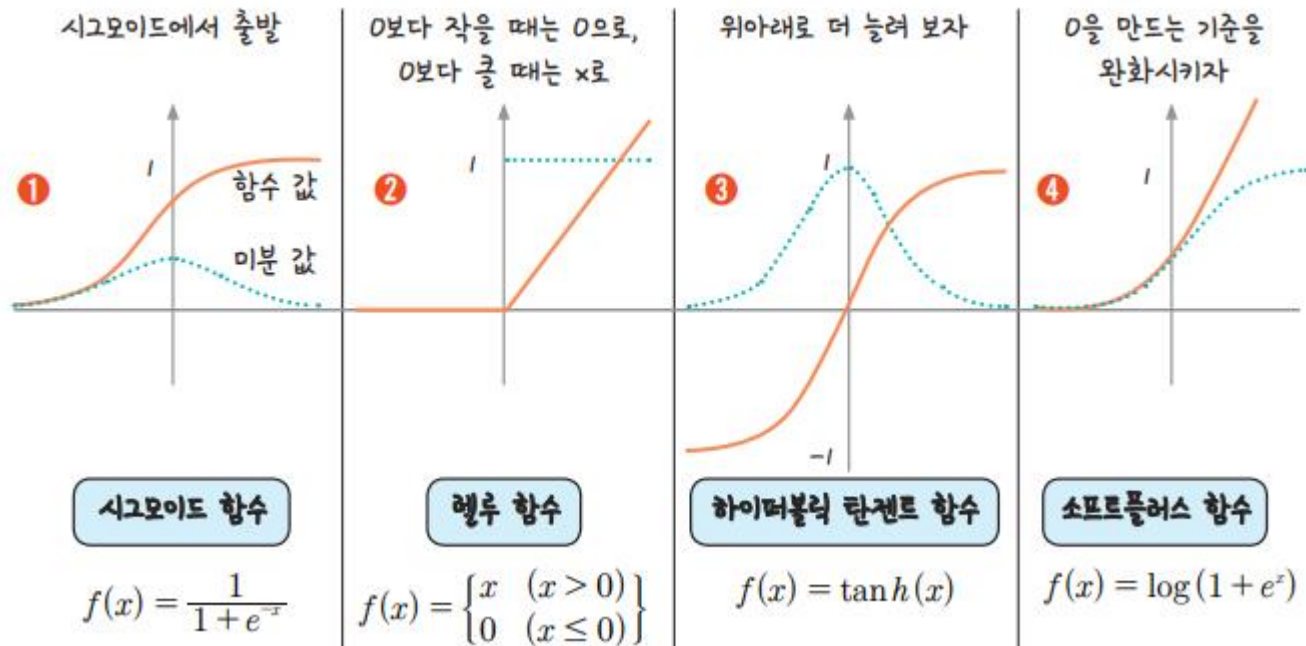


2 활성화 함수와 고급 경사 하강법

● 활성화 함수와 고급 경사 하강법

- 이를 해결하고자 제프리 힌튼 교수는 **렐루(ReLU)**라는 새로운 활성화 함수를 제안

▼ 그림 9-6 | 여러 활성화 함수의 도입



2 활성화 함수와 고급 경사 하강법

● 활성화 함수와 고급 경사 하강법

- 그림 9-6 ②에 있는 렐루는 x 가 0보다 작을 때 모든 값을 0으로 처리하고, 0보다 큰 값은 x 를 그대로 사용하는 방법
- 파란색 점선이 미분을 한 결과인데, 그림에서 보이듯 x 가 0보다 크기만 하면 미분 값은 1이 됨
- 활성화 함수로 렐루를 쓰면 여러 번 오차 역전파가 진행되어도 맨 처음 층까지 값이 남아 있게 됨
- 학습은 결국 오차를 최소화하는 가중치를 찾는 과정이라고 했음
- 출력층에서 알아낸 오차가 역전파를 통해 입력층까지 거슬러 올라가면서 잘못된 가중치들을 수정할 수 있게 되자, 더 깊은 층을 쌓아 올리는 것이 가능해졌음
- 활성화 함수는 그 이후로도 여러 데이터 과학자에 의해 연구되어 ③ 하이퍼볼릭 탄젠트(hyperbolic tangent) 함수나 ④ 소프트플러스(softplus) 함수 등 좀 더 나은 활성화 함수를 만들기 위한 노력이 이어지고 있음



3 속도와 정확도 문제를 해결하는 고급 경사 하강법

3 속도와 정확도 문제를 해결하는 고급 경사 하강법



- 속도와 정확도 문제를 해결하는 고급 경사 하강법

- 우리는 가중치를 업데이트하는 방법으로 경사 하강법을 배웠음
- 경사 하강법은 정확하게 가중치를 찾아가지만, 계산량이 매우 많다는 단점이 있음
- 이러한 점을 보완한 고급 경사 하강법이 등장하면서 딥러닝의 발전 속도는 더 빨라졌음

3 속도와 정확도 문제를 해결하는 고급 경사 하강법



● 속도와 정확도 문제를 해결하는 고급 경사 하강법

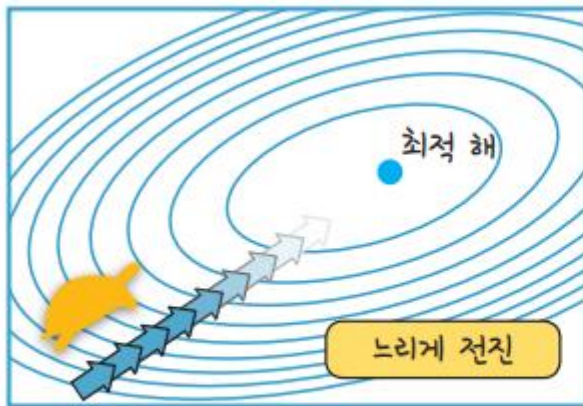
■ 확률적 경사 하강법

- 경사 하강법은 한 번 업데이트할 때마다 전체 데이터를 미분하므로 속도가 느릴 뿐 아니라, 최적 해를 찾기 전에 최적화 과정이 멈출 수도 있음
- **확률적 경사 하강법(Stochastic Gradient Descent, SGD)**은 경사 하강법의 이러한 단점을 보완한 방법
- 전체 데이터를 사용하는 것이 아니라, 랜덤하게 추출한 일부 데이터만 사용하기 때문에 빠르고 더 자주 업데이트할 수 있다는 장점이 있음
- 그림 9-7은 경사 하강법과 확률적 경사 하강법의 차이를 보여 줌
- 랜덤한 일부 데이터를 사용하는 만큼 확률적 경사 하강법은 중간 결과의 진폭이 크고 불안정해 보일 수도 있음
- 속도가 확연히 빠르면서도 최적 해에 근사한 값을 찾아낸다는 장점 덕분에 경사 하강법의 대안으로 사용되고 있음

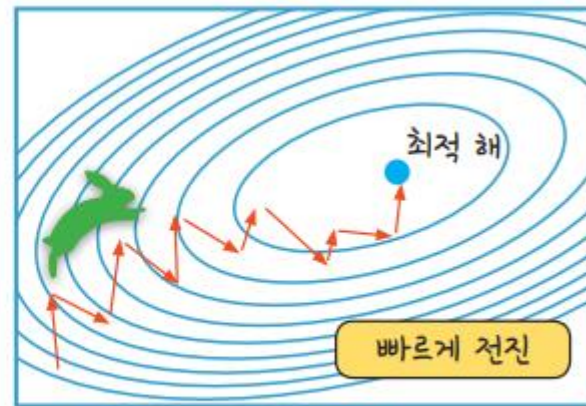
3 속도와 정확도 문제를 해결하는 고급 경사 하강법



▼ 그림 9-7 | 경사 하강법과 확률적 경사 하강법의 비교



경사 하강법



확률적 경사 하강법

3 속도와 정확도 문제를 해결하는 고급 경사 하강법



- 속도와 정확도 문제를 해결하는 고급 경사 하강법

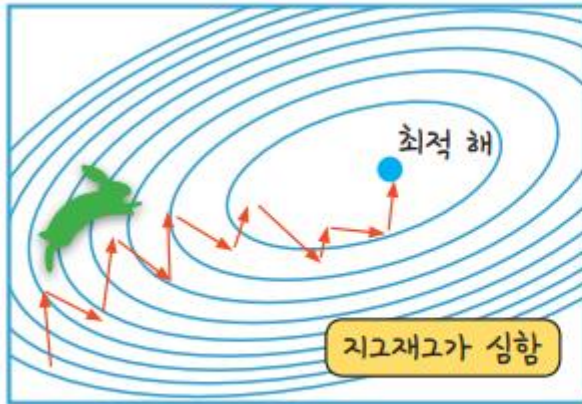
- 모멘텀

- 모멘텀(momentum)이란 단어는 '관성, 탄력, 가속도'라는 뜻
 - 모멘텀 확률적 경사 하강법(모멘텀 SGD)이란 말 그대로 경사 하강법에 탄력을 더해 주는 것
 - 다시 말해 경사 하강법과 마찬가지로 매번 기울기를 구하지만, 이를 통해 오차를 수정하기 전 바로 앞 수정 값과 방향(+, -)을 참고해 같은 방향으로 일정한 비율만 수정되게 하는 방법
 - 수정 방향이 양수(+) 방향으로 한 번, 음수(-) 방향으로 한 번 지그재그로 일어나는 현상이 줄어들고, 이전 이동 값을 고려해 일정 비율만큼 다음 값을 결정하므로 관성 효과를 낼 수 있음

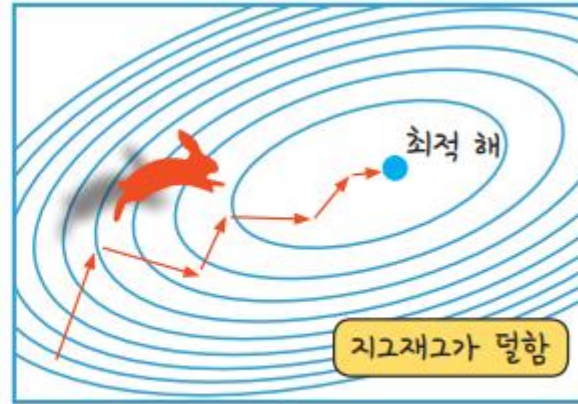
3 속도와 정확도 문제를 해결하는 고급 경사 하강법



▼ 그림 9-8 | 모멘텀을 적용했을 때



확률적 경사 하강법



모멘텀을 적용한 확률적 경사 하강법

3 속도와 정확도 문제를 해결하는 고급 경사 하강법



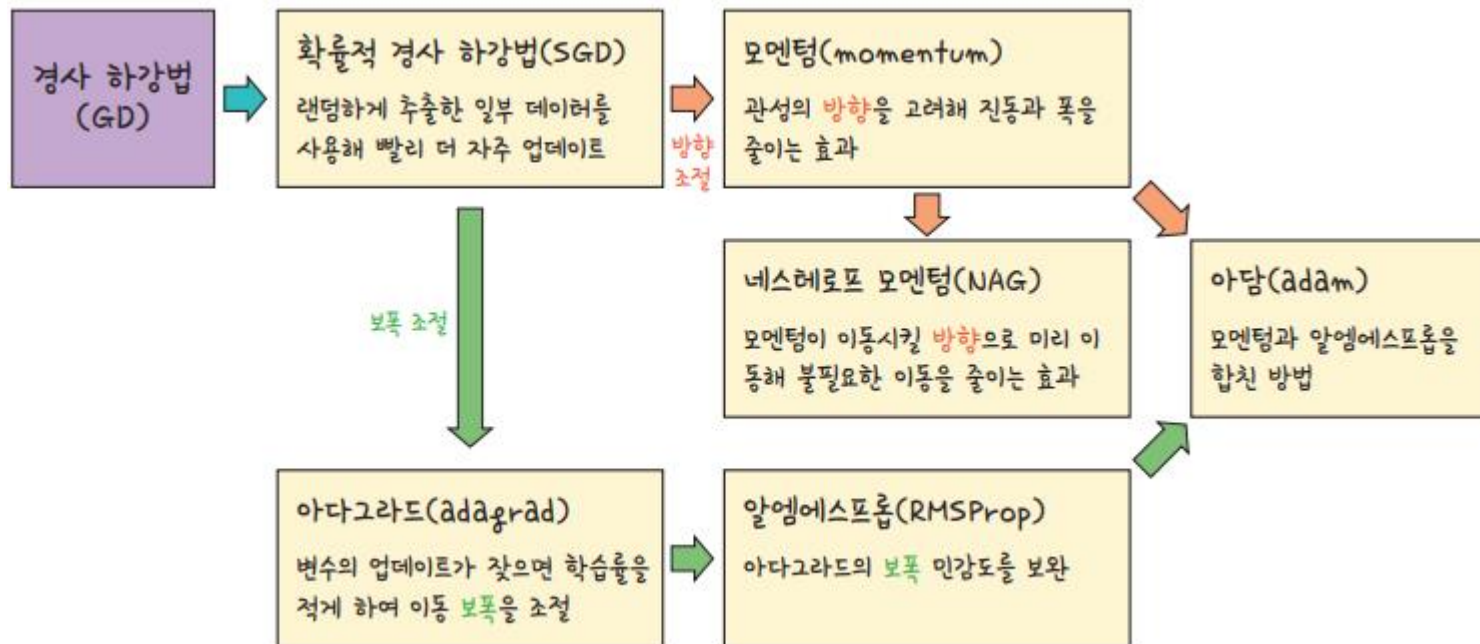
- 속도와 정확도 문제를 해결하는 고급 경사 하강법

- 이 밖에도 딥러닝의 학습을 더 빠르고 정확하게 만들기 위한 노력이 계속되었음
- 지금은 정확도와 속도를 모두 향상시킨 **아담(adam)**이라는 고급 경사 하강법이 가장 많이 쓰이고 있음
- 그림 9-9는 경사 하강법이 어떻게 해서 아담에 이르게 되었는지 보여 줌

3 속도와 정확도 문제를 해결하는 고급 경사 하강법



▼ 그림 9-9 | 딥러닝에 사용되는 고급 경사 하강법의 변천



3 속도와 정확도 문제를 해결하는 고급 경사 하강법



● 속도와 정확도 문제를 해결하는 고급 경사 하강법

- 이렇게 오차를 최소화하는 경사 하강법들을 딥러닝에서는 '옵티마이저'라고 한다고 했음
- 앞에 소개한 고급 경사 하강법들은 텐서플로에 포함되어 있는 **optimizers**라는 객체에 이름을 적어 주는 것만으로 손쉽게 실행할 수 있음
- 또 앞 절에서 소개된 시그모이드, 렐루 등 활성화 함수도 **activation**이라는 객체에 이름을 적어 주는 것으로 손쉽게 실행할 수 있음
- 이어지는 장에서 이러한 부분을 실습해 보자
- 이렇게 해서 4장부터 9장까지, 텐서플로에서 사용되는 대부분의 개념과 용어를 상세히 배웠습음
- 드디어 텐서플로를 이용한 모델링을 할 준비가 되었음