

The background features a large, abstract, wavy shape in shades of green, resembling a stylized wave or a flowing ribbon. It starts from the left, curves upwards and then downwards, and ends on the right side. The color transitions from a lighter green on the left to a darker green on the right. A solid dark green horizontal bar is positioned at the very bottom of the image.

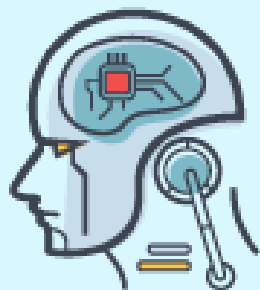
Introduction to Deep Learning

인공 지능 / 머신 러닝 / 딥러닝

Artificial Intelligence

인공지능

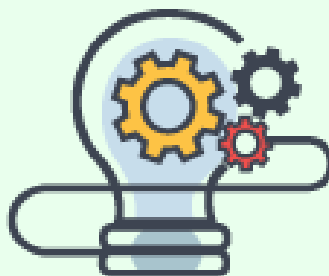
사고나 학습등 인간이 가진
지적 능력을 컴퓨터를 통해
구현하는 기술



Machine Learning

머신러닝

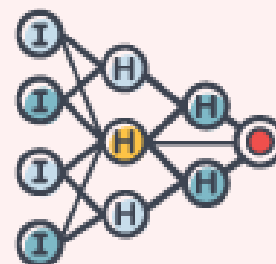
컴퓨터가 스스로 학습하여
인공지능의 성능을
향상 시키는 기술 방법



Deep Learning

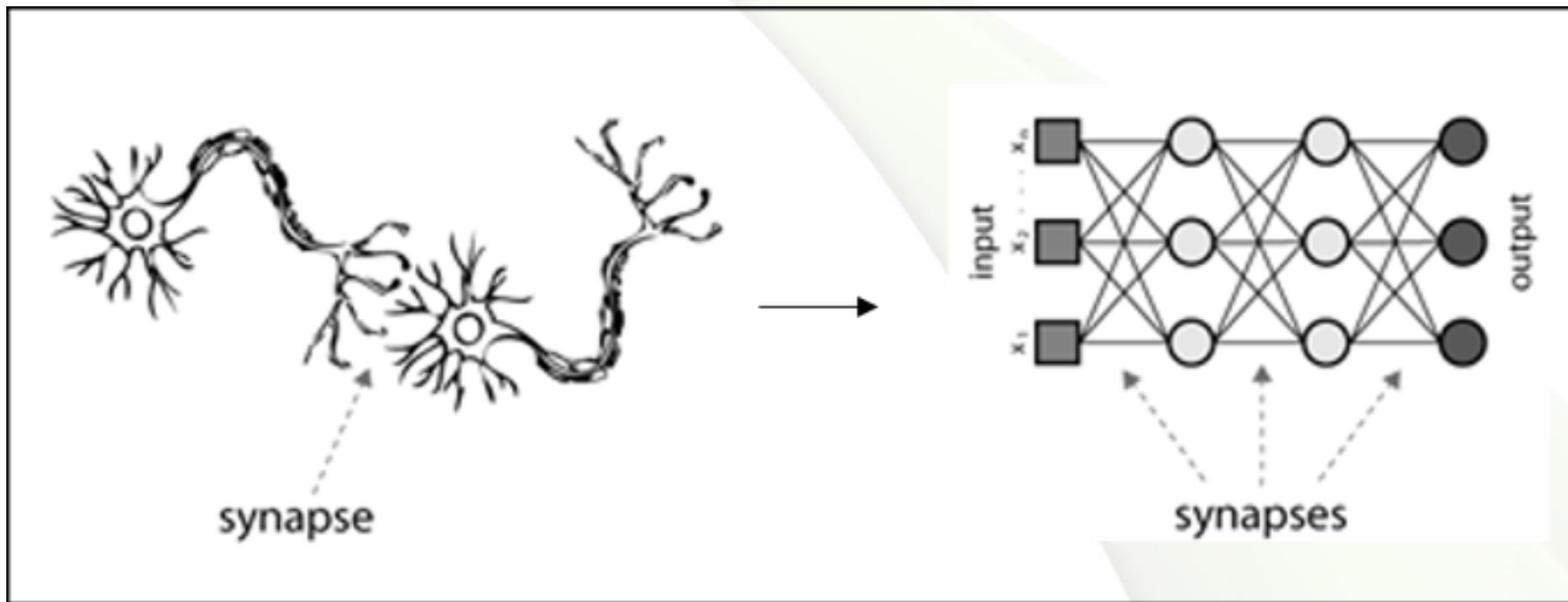
딥러닝

인간의 뉴런과 비슷한
인공신경망 방식으로
정보를 처리



신경망?

- 생물학의 신경망에서 영감을 얻은 통계학적 학습 알고리즘
- 시냅스의 결합으로 네트워크를 형성한 인공 뉴런(노드)이 학습을 통해 시냅스의 결합 강도를 변화시켜 문제 해결 능력을 가지는 모델

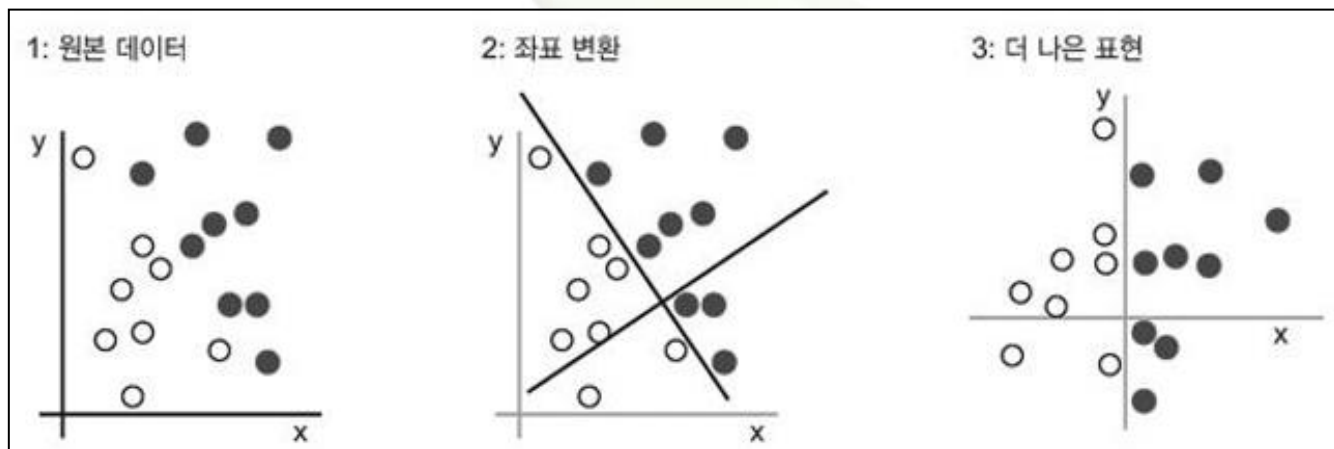


- 딥러닝의 핵심 → 복잡한 대규모 머신러닝 문제를 다루는 데 적합

딥러닝

- 머신 러닝의 한 분야로 연속된 층에서 점진적으로 의미 있는 표현을 학습
- 데이터로부터 표현을 학습하는 새로운 방식 (수학 모델)

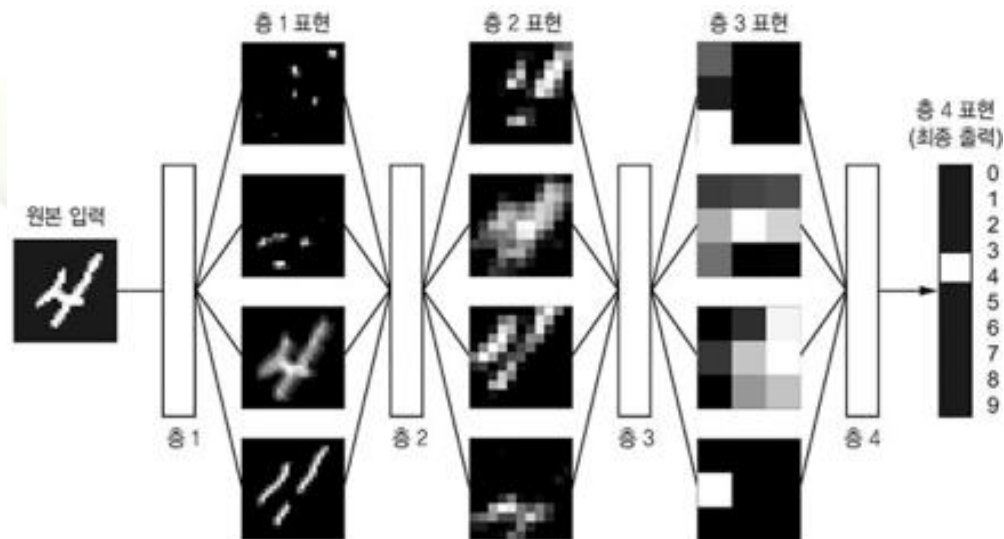
↓ 데이터를 인코딩하거나 묘사하기 위해 데이터를 바라보는 다른 방법



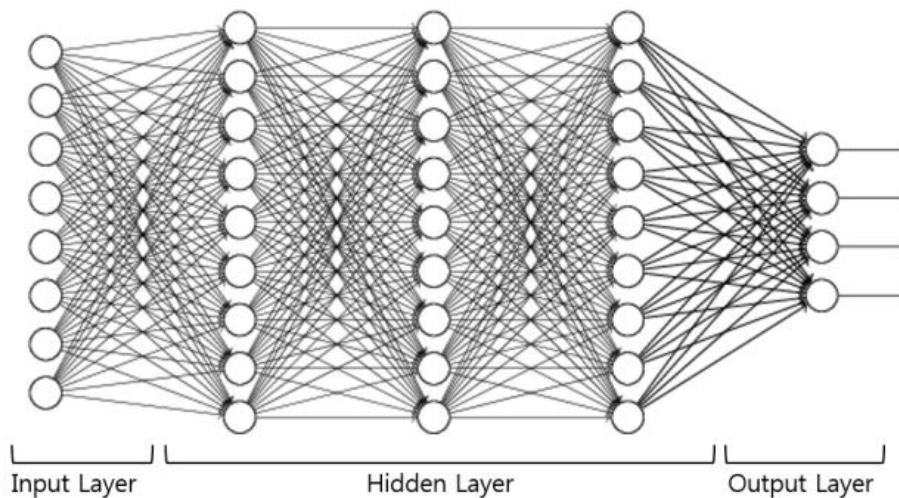
- 연속된 층으로 표현을 학습 → 얼마나 많은 층을 사용 했는지가 모델의 깊이
- 층 기반 표현 학습 또는 계층적 표현 학습

딥러닝

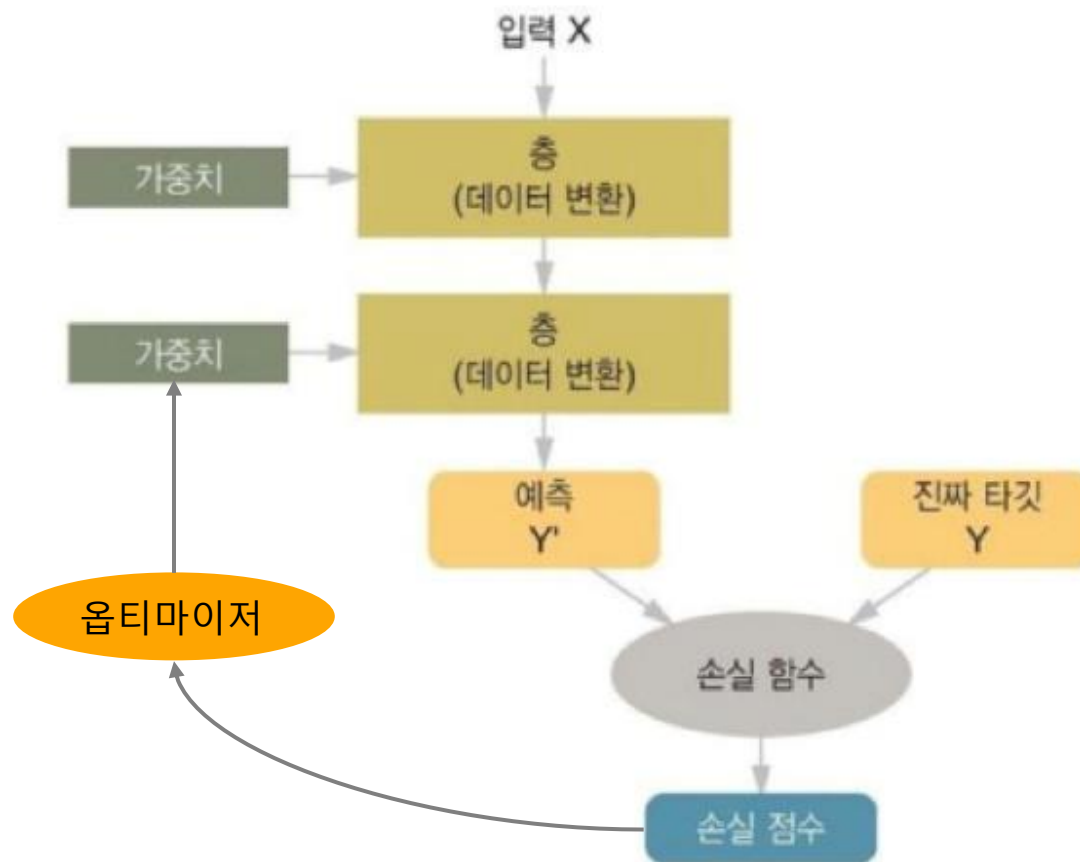
- 기본 층을 겹겹이 쌓아 올려 구성한 신경망 (Neural Network) 이라는 모델을 사용하여 표현을 학습



- 최근의 딥러닝 모델은 수십, 수백 개의 연속된 층으로 구성



딥러닝 작동 원리



딥러닝 성장 배경

- 1943년 최초의 인경신경망 소개 이후 성장기와 침체기를 반복하다가 2012년 이후 크게 성장
 - 하드웨어
 - » CPU의 성능 향상
 - » 딥러닝을 지원하는 그래픽 카드 개발과 API 지원
 - 데이터
 - » 데이터는 인공 지능이라는 기관을 움직이는 원료
 - » 저장 장치의 급격한 발전 및 대량의 데이터 세트를 수집하고 배포할 수 있는 인터넷의 성장
 - 알고리즘
 - » 신경망 층에 잘 맞는 활성화 함수
 - » 가중치 초기화
 - » 가중치 업데이트를 위한 최적화 알고리즘

신경망을 위한 데이터 표현

■ 텐서 (Tensor)

- » 수치형 데이터를 위한 컨테이너 (다차원 Numpy 배열)
- » 임의의 차원 개수를 가지는 행렬의 일반화된 모습
- » 최근의 모든 머신 러닝 시스템의 기본 데이터 구조

■ 종류

종류	설명
스칼라 (0D 텐서)	하나의 숫자만 저장 / 축 개수 0
벡터 (1D 텐서)	숫자의 배열 / 축 개수 1
행렬 (2D 텐서)	벡터의 배열 / 축 개수 2 (첫 번째 축 행 / 두 번째 축 열)
3D 이상 고차원 텐서	행렬들을 하나의 새로운 배열로 결합하면 3D

- » 딥러닝에서는 보통 0D에서 4D까지의 텐서 사용
- » 동영상 데이터를 다루는 경우 5D 텐서까지 사용

신경망을 위한 데이터 표현

- 텐서의 핵심 속성
 - » 축의 수 (랭크)
 - » 크기 (Shape) → 텐서의 각 축에 포함된 차원 / (3, 5) (4, 2, 6) 등으로 표현
 - » 데이터 타입 → 텐서에 포함된 데이터의 타입
- 텐서의 실제 사례

데이터	텐서 표현
벡터 데이터	(sample, features)
시계열 데이터 / 시퀀스 데이터	(sample, timestamp, features)
이미지	(samples, height, width, channels) (samples, channels, height, width)
동영상	(samples, frames, height, width, channels) (samples, frames, channels, height, width)

텐서 연산

- 심층 신경망이 학습한 모든 변환을 수치 데이터 텐서에 적용하는 몇 종류의 텐서 연산으로 표현 가능
- 텐서 원소별 연산
 - » 텐서에 있는 각 원소에 개별적으로 적용되는 연산

```
a = np.array([ 1, 2, 3, 4, 5])  
b = np.array([10, 20, 30, 40, 50])  
c = 100
```

a + b

```
array([11, 22, 33, 44, 55])
```

a * c

```
array([100, 200, 300, 400, 500])
```

- 브로드캐스팅
 - » 크기가 다른 두 텐서의 연산에서 모호하지 않고 실행 가능하다면 작은 텐서가 큰 텐서의 크기에 맞게 변경

```
a = np.array([1, 2, 3])  
b = np.array([[10, 20, 30], [40, 50, 60]])  
c = a + b
```

```
print(c.shape)
```

c

```
(2, 3)
```

```
array([[11, 22, 33],  
       [41, 52, 63]])
```

텐서 연산

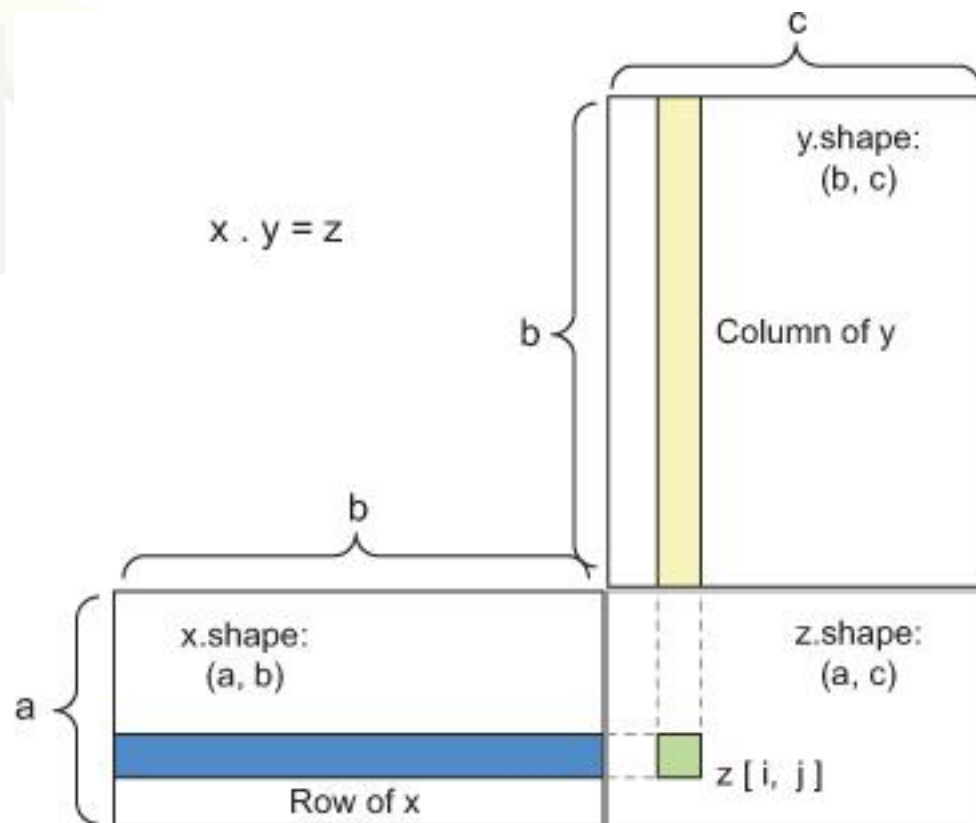
- 텐서 점곱 (dot product)
 - » 입력 텐서의 원소들을 결합

```
a = np.array([[1, 1, 1],  
              [1, 1, 1]])  
b = np.array([[10, 20, 30, 40],  
              [50, 60, 70, 80],  
              [90, 100, 110, 120]])
```

```
print( a.shape, b.shape )  
np.dot(a, b)
```

```
(2, 3) (3, 4)
```

```
array([[220, 280, 340, 400],  
       [490, 640, 790, 940]])
```



텐서 연산

■ 텐서 크기 변환

- » 특정 크기에 맞게 행과 열을 재배열
- » 원래 텐서와 원소 개수는 동일

```
a = np.array([[10, 20, 30, 40],  
             [50, 60, 70, 80],  
             [90, 100, 110, 120]])  
b = a.reshape((2, 6))
```

```
print( a.shape, b.shape )  
b
```

```
(3, 4) (2, 6)
```

```
array([[ 10,  20,  30,  40,  50,  60],  
       [ 70,  80,  90, 100, 110, 120]])
```

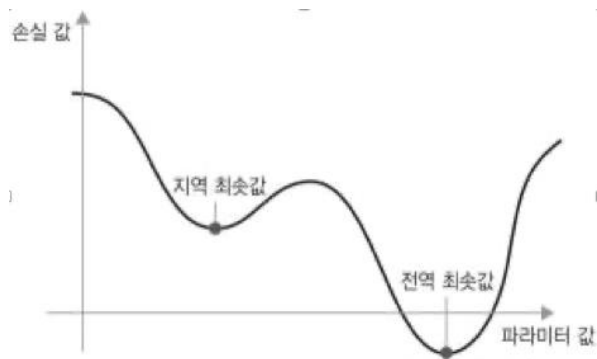
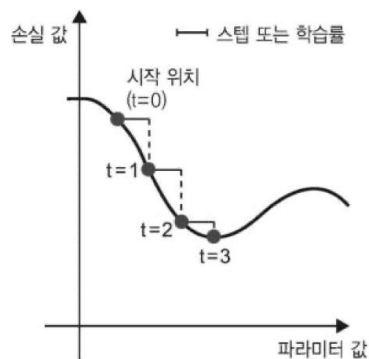
■ 텐서 연산의 의미

- » 모든 텐서 연산은 입력 데이터의 기하학적 변환
- » 단순한 단계들이 길게 이어져 구현된 신경망은 고차원 공간에서 매우 복잡한 기하학적 변환으로 해석 가능



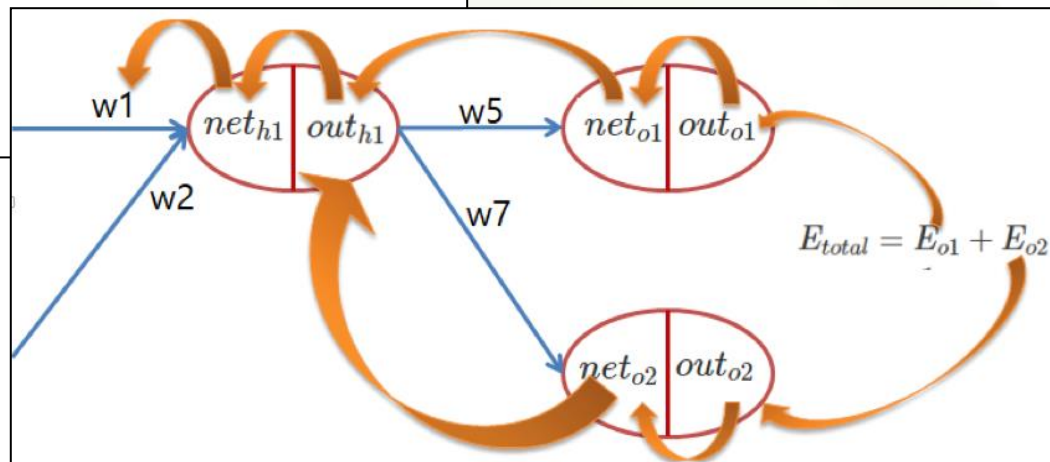
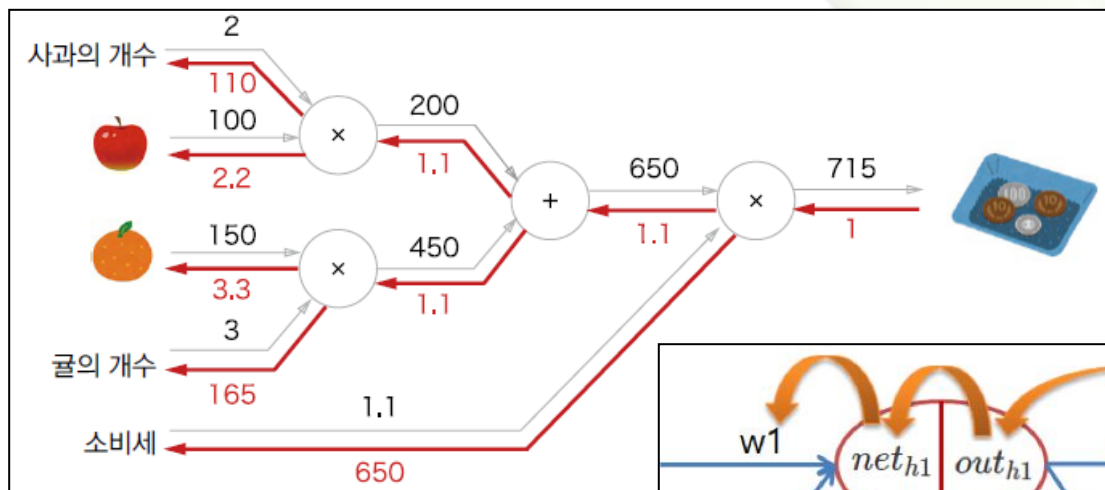
그라디언트 기반 최적화

- 가중치에 대한 점진적인 조정 또는 훈련이 머신 러닝 / 딥러닝 학습의 핵심
- 학습 단계 (반복)
 - » 훈련 샘플 x 와 이에 상응하는 타겟 y 의 배치 추출
 - » x 를 사용하여 네트워크 실행 \rightarrow 예측 y_{pred} 도출
 - » y_{pred} 와 y 의 차이를 계산하고 현재 배치에 대한 손실 계산
 - » 손실이 조금 감소되도록 네트워크의 모든 가중치 업데이트
- 가중치 업데이트 방법
 - » 경사 하강법, 확률적 경사 하강법, 배치 경사 하강법
 - » Momentum, Adagrad, RMSProp 등의 SGD 변형 옵티마이저



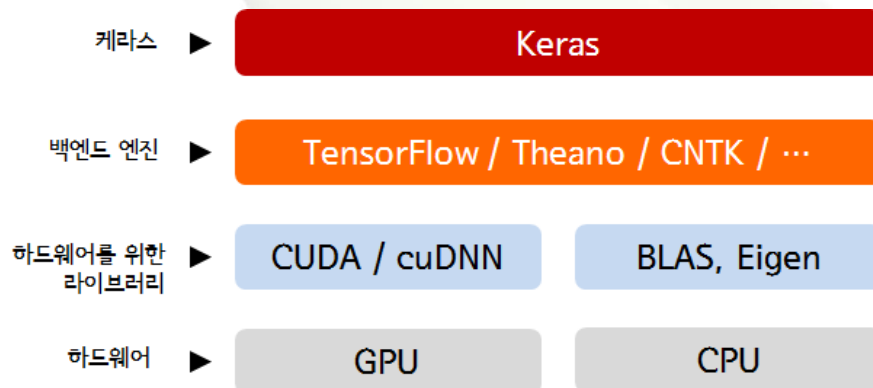
역전파 알고리즘

- 수치 미분의 단점 → 계산 비용이 커서 시간이 오래 걸리는 문제
- 오차 역전파법은 가중치 매개변수의 기울기를 효율적으로 계산하는 기법
 - » 네트워크의 연산 결과 (출력 값)에 대한 입력 값의 기울기를 출력 층에서 시작하여 입력 층 방향으로 전파하면서 각 층의 기울기를 연산하는 기법



케라스

- 거의 모든 종류의 딥러닝 모델을 간편하게 만들고 훈련시킬 수 있는 파이썬을 위한 딥러닝 프레임워크
- 신속하게 실험을 해야 하는 연구자들을 위해 개발
- 특징
 - » 동일한 코드로 CPU와 GPU에서 실행
 - » 사용하기 쉬운 API 제공 → 딥러닝 모델의 프로토타입을 빠르게 구현
 - » 합성곱 신경망, 순환 신경망, 적대적 생성 신경망 등 다양한 신경망 기법 지원
- 모델 수준의 라이브러리
 - » 텐서 조작이나 미분 같은 저수준 연산은 백엔드 엔진을 통해 수행하고 케라스는 고수준의 구성 요소 제공



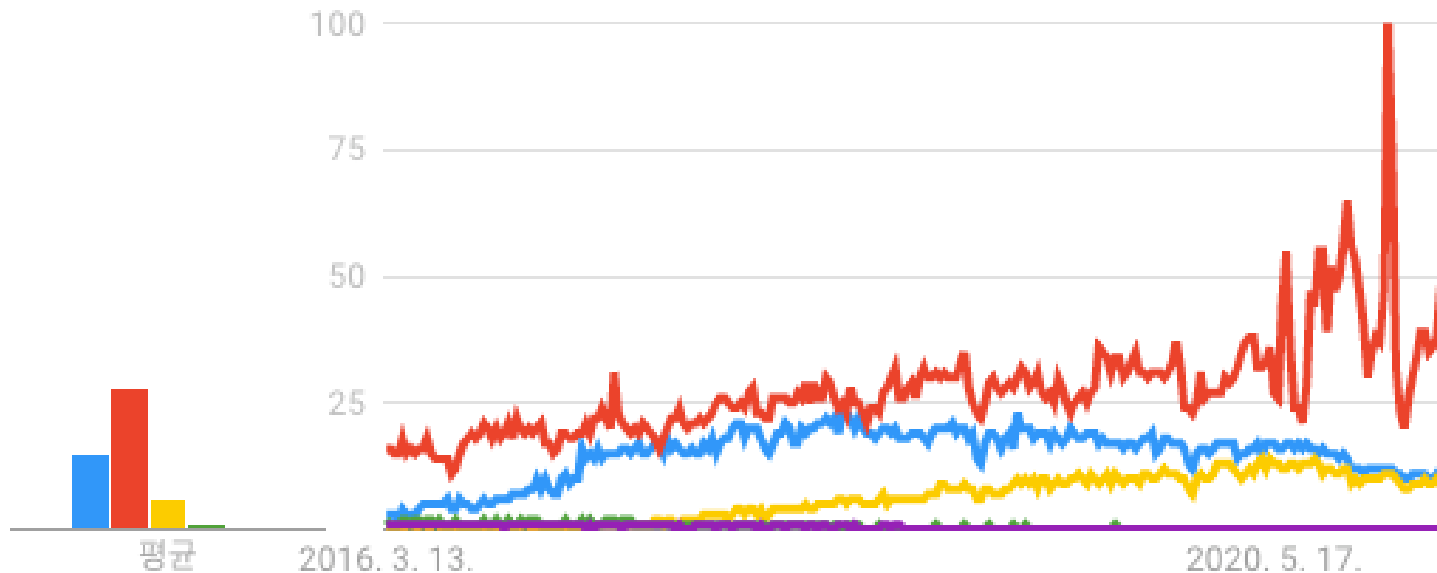
케라스

■ 딥러닝 프레임워크 관심도 비교

시간 흐름에 따른 관심도 변화

Google Trends

● tensorflow ● keras ● pytorch ● Theano ● cntk

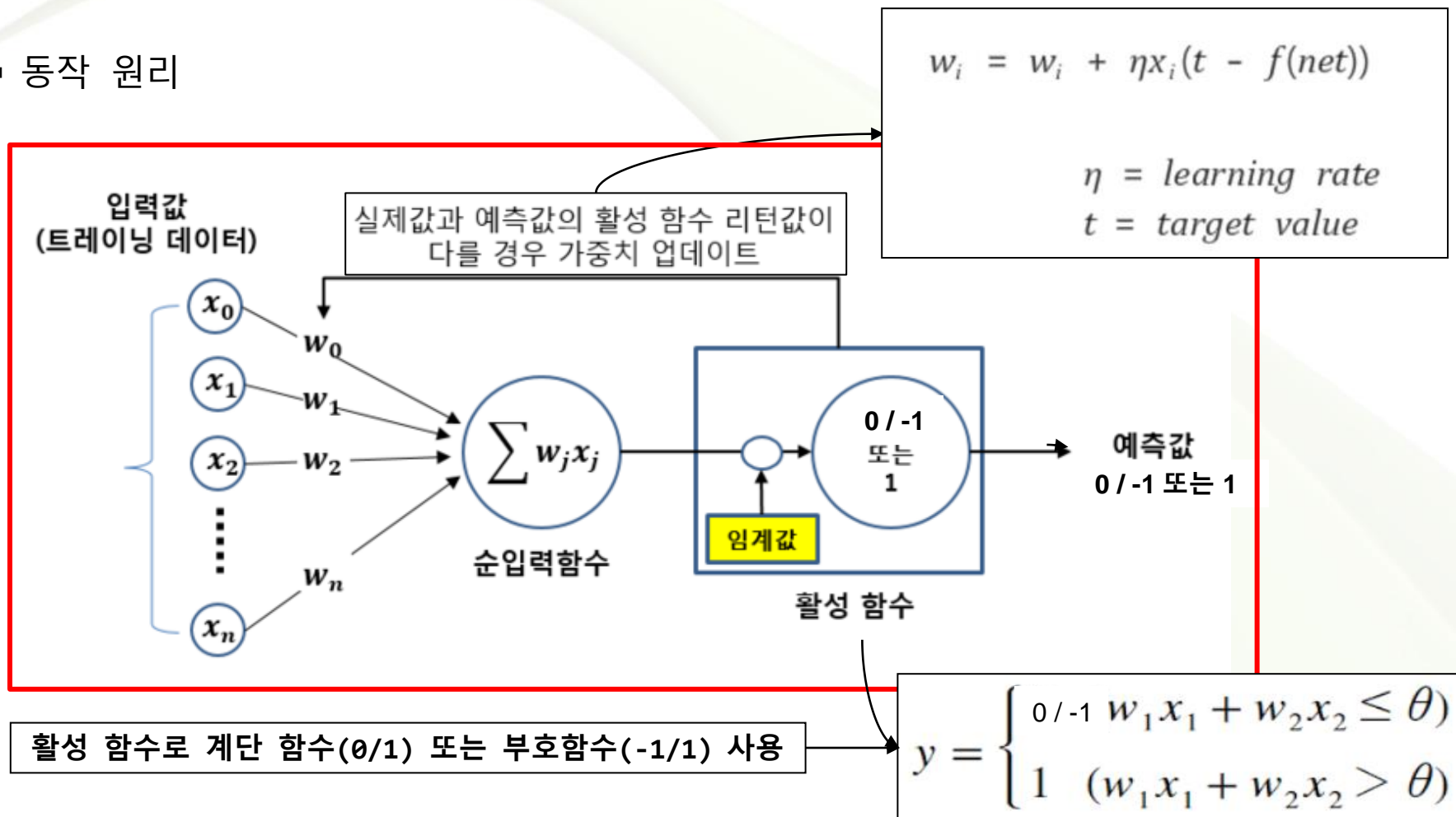


The background features a large, flowing green wave that starts from the left, peaks in the upper middle, and then descends towards the bottom right. The wave has a gradient from a lighter green to a darker green. A solid dark green horizontal bar is at the very bottom of the image.

신경망 요소

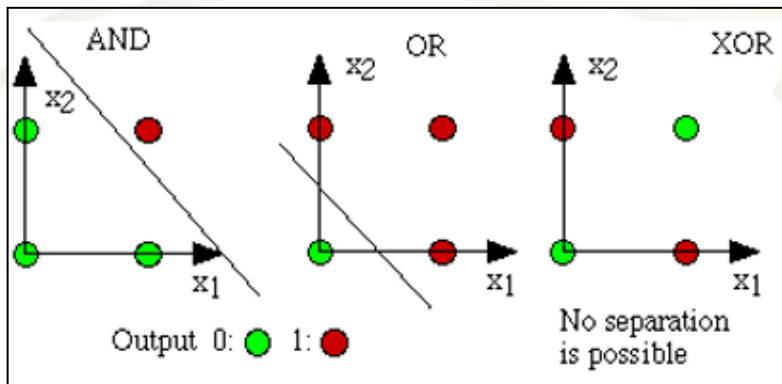
단층 퍼셉트론(Perceptron)

- 프랑크 로젠블라트(Frank Rosenblatt)가 1957년에 고안한 신경망의 기원이 되는 알고리즘
- 동작 원리

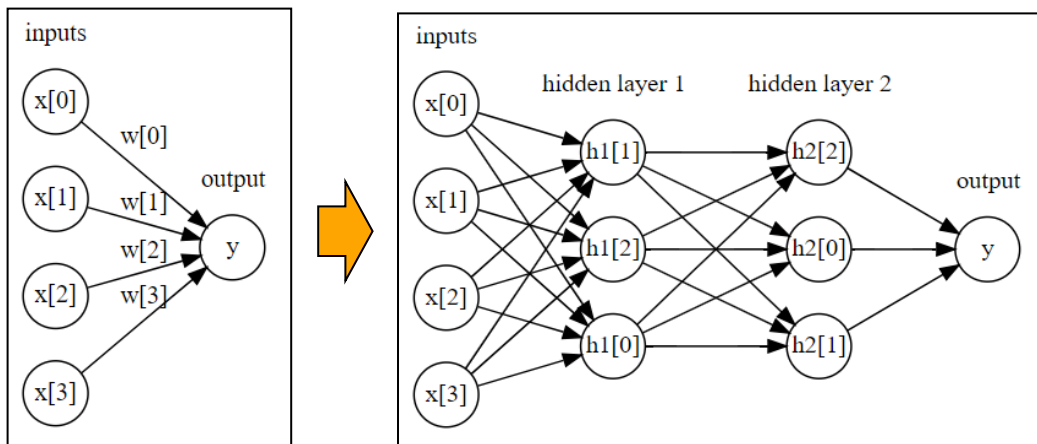


다층 퍼셉트론

- 단층 퍼셉트론으로는 XOR 문제를 해결할 수 없음 → 다층 퍼셉트론 필요

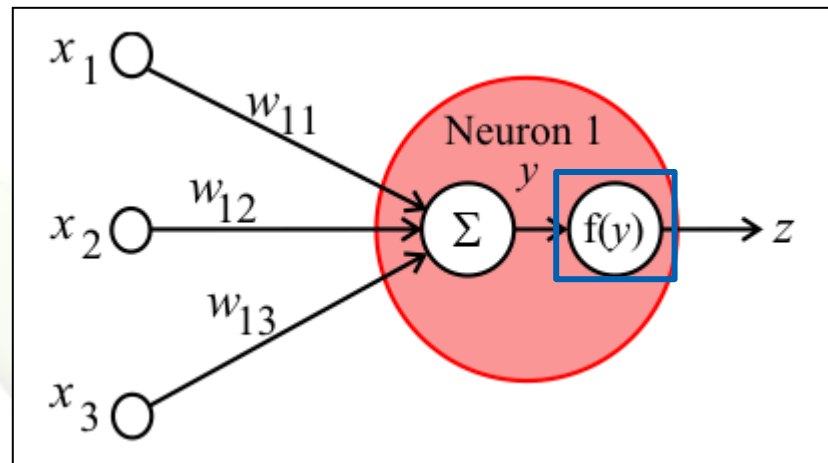


- 입력층과 출력층 사이에 하나 이상의 은닉층이 존재하는 신경망



활성화 함수 (Activation Function)

- 입력 신호의 총합을 출력 신호로 변환하는 함수
 - » 신호의 총합으로 활성화 여부를 결정
 - » 계단 함수에서 다른 함수로 변경하는 것이 신경망 세계로 나아가는 열쇠
 - » 다층 신경망 구현을 위해 반드시 비선형 활성화 함수 사용

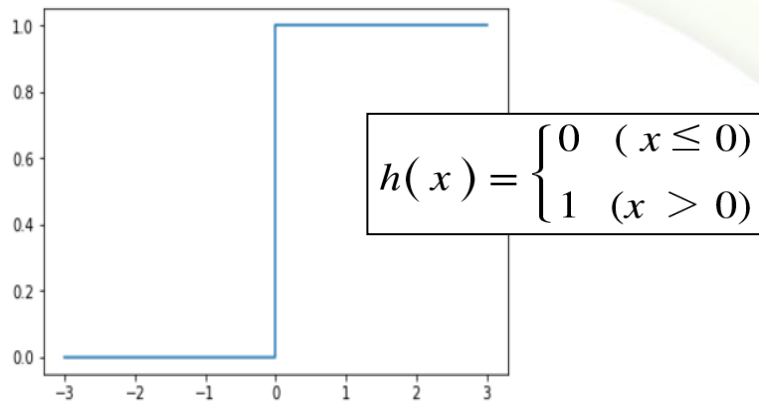


- 종류
 - » 계단 함수
 - » Sigmoid 함수
 - » ReLU 함수
 - » 하이퍼볼릭 탄젠트

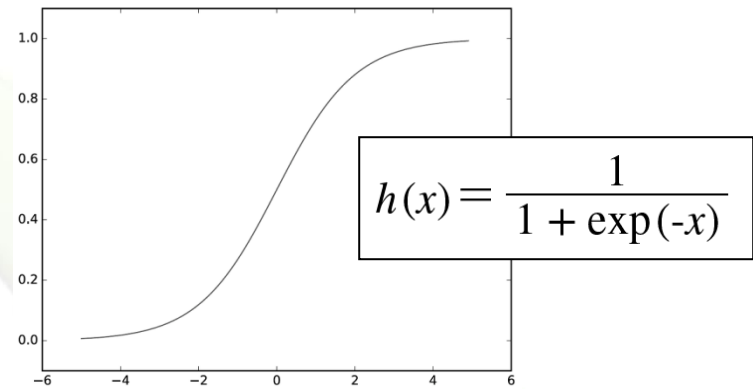
비선형 함수를 사용하지 않을 경우
활성화 함수가 $f(x) = cx$ 라면
 $f(f(f(x))) = c * c * cx$ 이므로
 $c * 3x$ 인 은닉층이 없는 단층 신경망으로 치환 가능

활성화 함수 종류

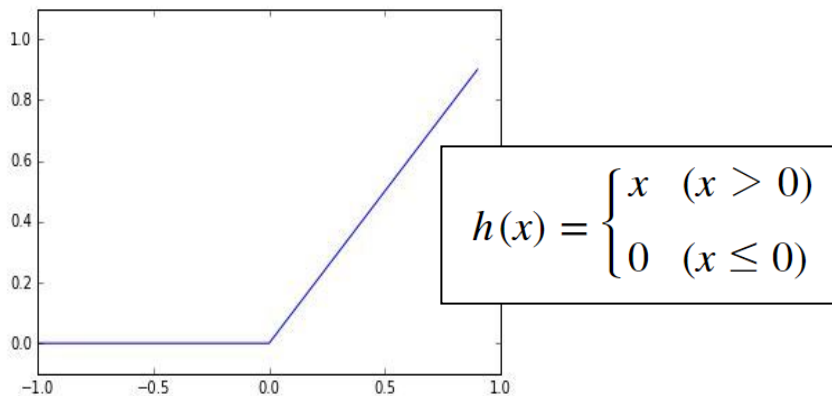
■ 계단 함수



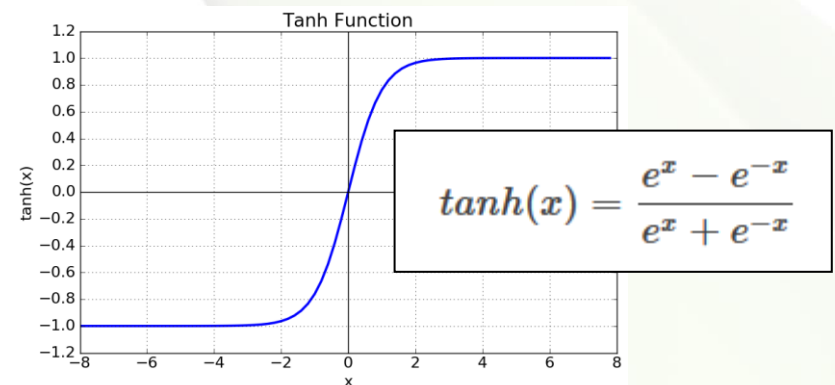
■ Sigmoid 함수



■ ReLU 함수



■ 하이퍼볼릭 탄젠트 함수



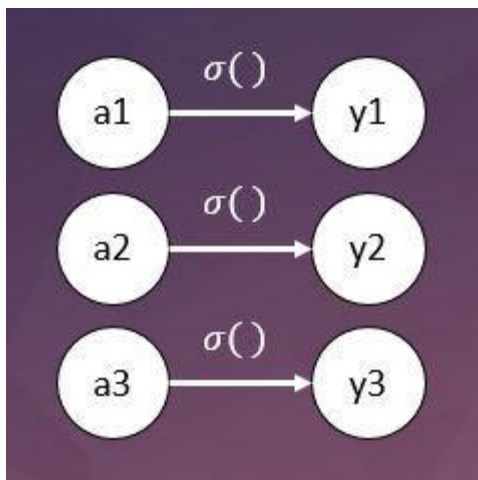
출력층

■ 분류와 회귀

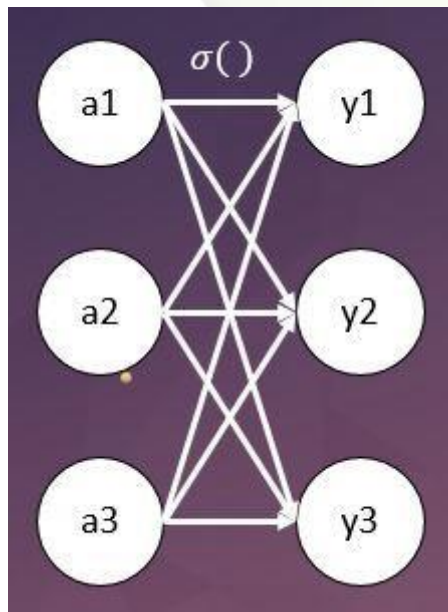
- » 신경망은 분류와 회귀 모두에 적용 가능
- » 분류는 데이터가 어느 범부에 속하는지를 판단하는 문제
- » 회귀는 입력 데이터로 결과 수치를 예측하는 문제
- » 일반적으로 분류에는 소프트맥스 함수를 회귀에는 항등 함수를 출력 함수로 사용

■ 항등 함수

- » 입력 값을 그대로 출력



■ 소프트 맥스 함수



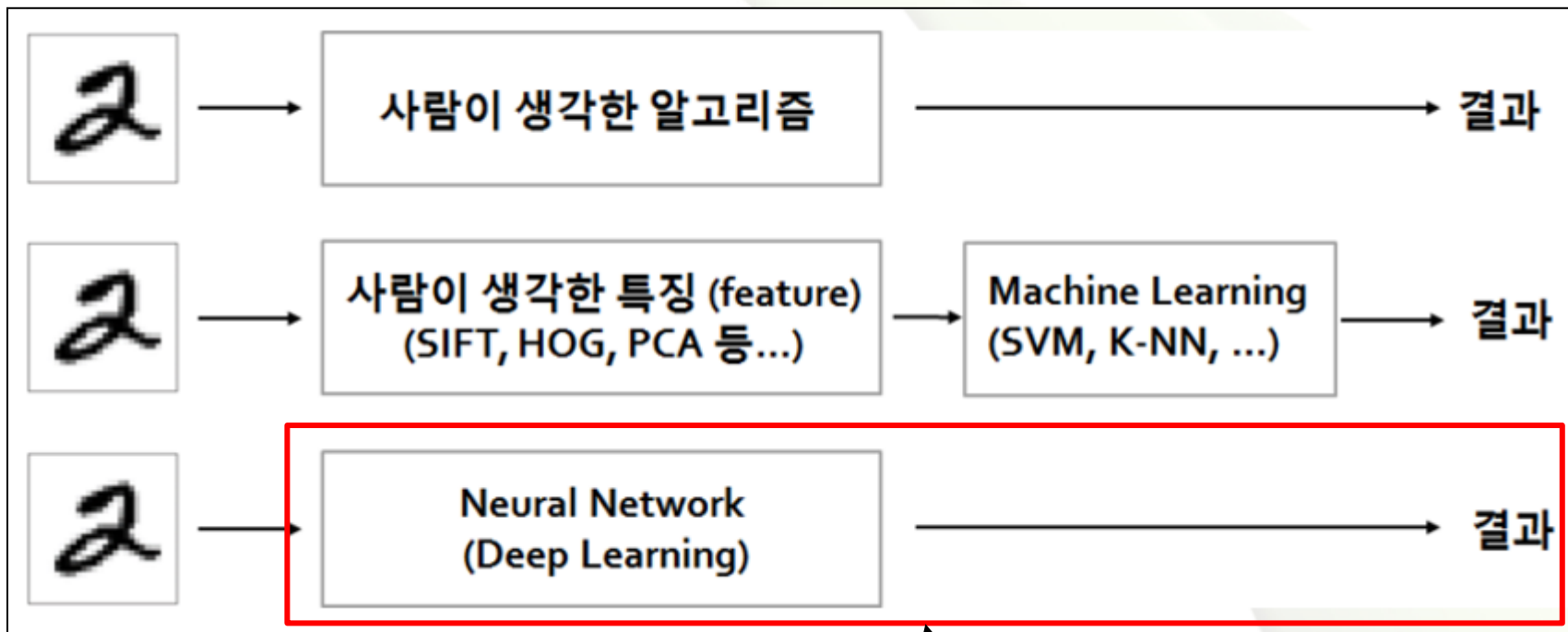
$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^n \exp(a_i)}$$

출력층

- 소프트맥스 함수의 특징
 - » 소프트맥스 함수의 출력은 $0 \sim 1$ 사이의 값
 - » 출력 총합은 1
 - » 그러므로 출력 값을 확률로 해석 가능
 - » 소프트맥스 함수를 적용해도 가장 큰 출력을 내는 뉴런의 위치는 변경되지 않기 때문에 신경망으로 분류할 때는 출력층의 소프트맥스 함수 생략 가능 (지수 함수 비용을 줄이는 효과)
- 출력층의 뉴런 수는 풀려는 문제에 맞게 결정
 - » 분류에서는 분류하고 싶은 클래스 수로 설정

신경망 학습

- 훈련 데이터로부터 가중치 매개변수의 최적 값을 자동으로 획득하는 과정
- 데이터 주도 학습
 - » 수집된 데이터로부터 특징과 규칙을 찾아내는 역할을 기계가 담당



종단간 기계학습 (end-to-end machine learning)

손실 함수

- 신경망은 어떤 "지표"를 기준으로 최적의 매개변수 값을 탐색
 - » 신경망 학습에서 사용하는 지표는 손실 함수(loss function)
 - » 일반적으로 평균 제곱 오차와 교차 엔트로피 오차를 손실 함수로 사용
- 한 건의 데이터에 대한 손실 함수 계산
 - » 평균 제곱 오차

$$E = \frac{1}{2} \sum_k (y_k - t_k)^2$$

- » 교차 엔트로피 오차

$$E = - \sum_k t_k \log y_k$$

미니 배치 학습

- 전체 데이터를 대상으로 손실함수를 계산할 경우 데이터 양이 많아지면 연산 비용이 크게 증가함
 - » 연산 비용을 효과적으로 관리하기 위해 미니 배치 사용
- 미니 배치
 - » 전체 데이터 중 일부를 추출해서 손실함수 값을 계산하고 이 값을 전체 데이터의 손실함수 값의 근사값으로 사용
 - » 이런 학습 방법이 미니 배치 학습
- 미니 배치 학습에서 손실 함수 값 계산

$$E = -\frac{1}{N} \sum_n \sum_k t_{nk} \log y_{nk}$$

손실 함수 활용

- 신경망 학습은 최적의 매개변수(가중치와 편향)을 탐색할 때 손실 함수의 값을 가능한 작게 만드는 매개변수의 값 추적
- 이 때 매개변수의 미분(기울기)을 계산하고 이 미분 값을 단서로 매개변수의 값을 서서히 갱신하는 과정 반복
- 정확도를 지표로 삼을 경우 미분 값이 대부분의 장소에서 0이 되어 매개변수를 갱신 불가능
 - » 정확도는 매개변수의 미세한 변화에는 영향 받지 않고 불연속적인 값으로 표현됨
 - » 예 : 1000개에서 320개를 정확하게 예측한 경우와 324개를 정확하게 예측한 경우 모두 32%

미분

- 수치 미분

- » 미분의 원리를 계산식으로 구현해서 도출
- » 증분 값의 크기를 대개 10^{-4} 으로 하고 중심 차분 방식 사용

- 해석적 미분

- » 수학적 해석 기반의 미분
- » 오차를 포함하지 않는 진정한 미분 값 도출

- 편미분

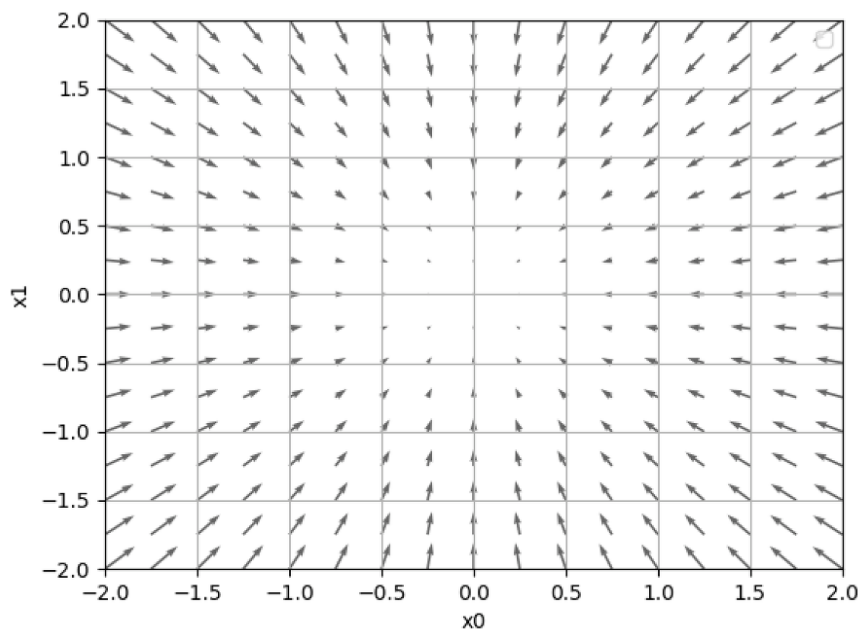
- » 변수가 여러 개인 함수에 대한 미분
- » 변수가 하나인 미분과 마찬가지로 특정 장소의 기울기 의미
- » 여러 변수 중에서 목표 변수 하나에 초점을 맞추고 다른 변수는 값 고정

기울기 계산

- 기울기

- » 모든 변수의 편미분을 동시에 계산해서 벡터로 정리한 것

- 이론적으로 기울기는 현재 위치에서 가장 낮은 장소를 가리키는 방향 표시 → 각 장소에서 (손실)함수의 출력 값을 가장 많이 감소시키는 방향

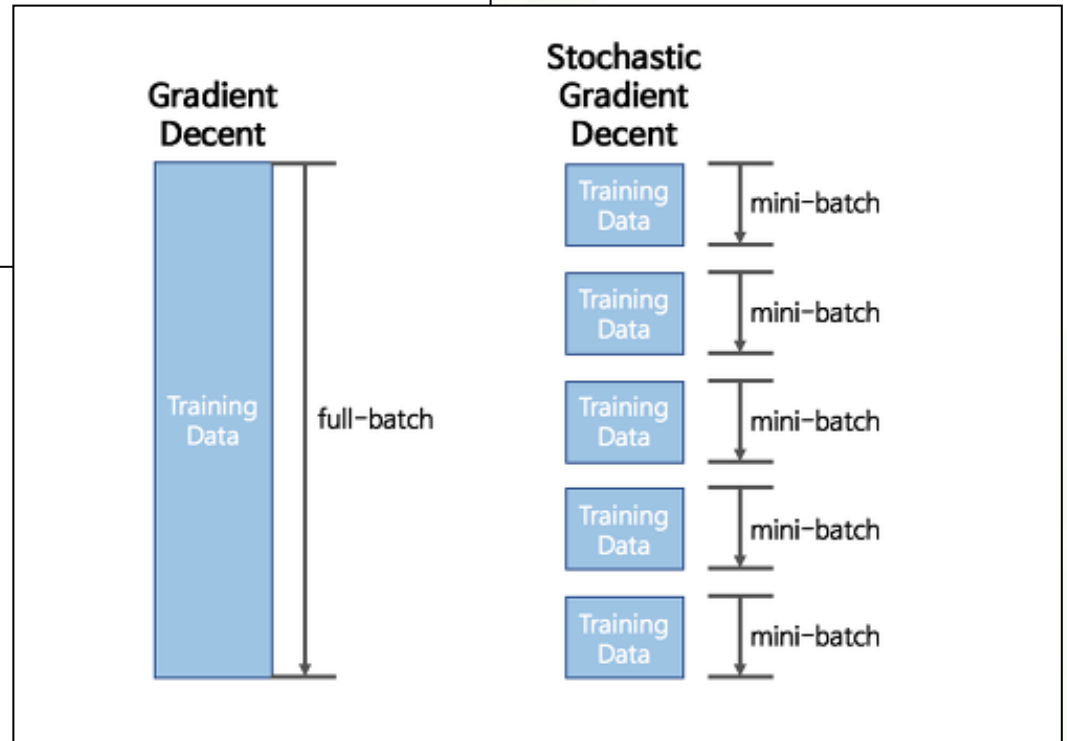
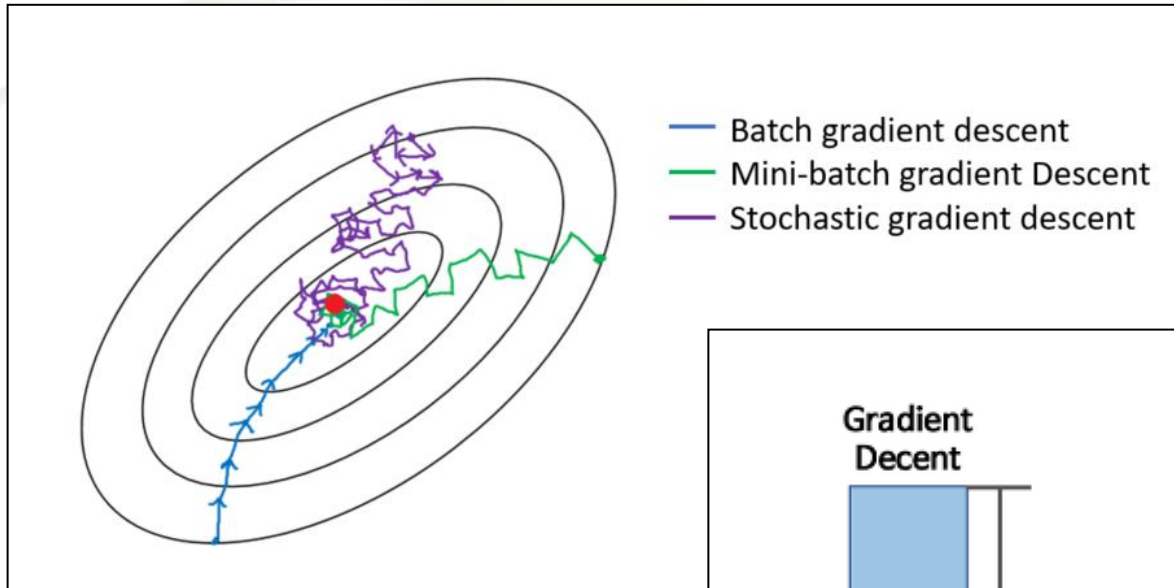


경사하강법

- 신경망은 학습을 통해 최적의 매개 변수를 추적
 - » 최적은 손실 함수가 최소값이 될 때의 매개변수 값
- 경사하강법 (Gradient Descent Method)
 - » 기울기를 잘 활용해서 (손실)함수의 최소값을 찾는 방법
 - » 과정
 - a. 특정 위치에서 기울기 계산
 - b. 특정 위치에서 기울어진 방향으로 일정 거리만큼 이동
 - c. 이동한 위치에서 다시 기울기 계산
 - d. 기울어진 방향으로 일정 거리만큼 이동
 - e. 기울기가 0이 되거나 특정한 조건을 만족할 때까지 a ~ d 반복
- 학습률
 - » 한 번의 학습에서 기울기 방향으로 갱신하는 양
 - » 학습률이 지나치게 크거나 작으면 효과적인 학습 불가능 (발산 또는 최저점에 도달하기 전 학습 종료)

경사하강법

- 배치 경사 하강법 vs 확률적 경사 하강법 vs 미니 배치 경사 하강법



매개 변수 갱신

- 매개 변수 최적화
 - » 신경망 학습의 목적
 - » 손실 함수의 값을 가능한 낮게 만드는 매개변수 추적
 - » 기울기(미분) 방향으로 매개변수 값 갱신 과정 반복
- 최적화 기법 종류
 - » 확률적 경사 하강법 (Stochastic Gradient Descent)
 - » 모멘텀 (Momentum)
 - » AdaGrad (Adaptive Gradient)
 - » Adam (Adaptive Momentum)

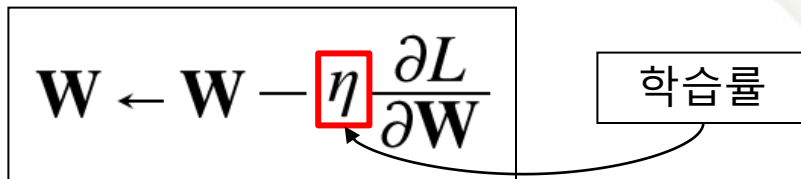
최적화 기법

■ 확률적 경사 하강법 (Stochastic Gradient Descent)

- » 손실함수의 기울기를 구해 기울어진 방향으로 학습률에 비례해서 매개변수 값 갱신 과정 반복
- » 방향에 따라 기울기가 달라지는 비등방성 함수에서는 효율성이 좋지 않음

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

학습률

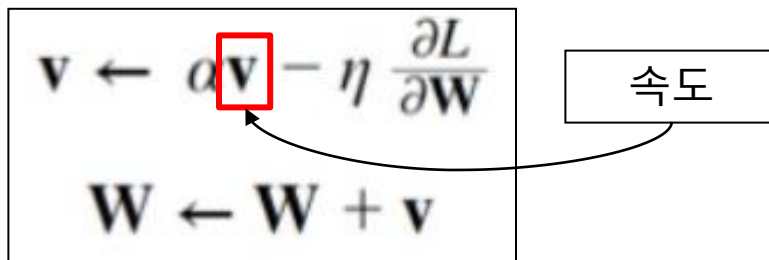


■ 모멘텀 (Momentum)

- » SGD에 가속도 개념 적용 → 변수 \mathbf{v}
- » α 값은 0.9 등의 값으로 설정

$$\mathbf{v} \leftarrow \alpha \mathbf{v} - \eta \frac{\partial L}{\partial \mathbf{W}}$$
$$\mathbf{W} \leftarrow \mathbf{W} + \mathbf{v}$$

속도



최적화 기법

▪ AdaGrad (Adaptive Gradient)

- » 개별 변수별로 학습률을 조금씩 조정하면서 학습 진행
- » 일반적으로 학습을 진행할수록 학습률 감소
 - 무한히 학습을 진행하면 학습률이 0이 되는 문제 발생 → 최신의 기울기를 크게 반영(지수이동평균)하는 RMSProp을 사용해서 해결

$$\begin{aligned} \mathbf{h} &\leftarrow \mathbf{h} + \frac{\partial L}{\partial \mathbf{W}} \odot \frac{\partial L}{\partial \mathbf{W}} \\ \mathbf{W} &\leftarrow \mathbf{W} - \eta \frac{1}{\sqrt{\mathbf{h}}} \frac{\partial L}{\partial \mathbf{W}} \end{aligned}$$

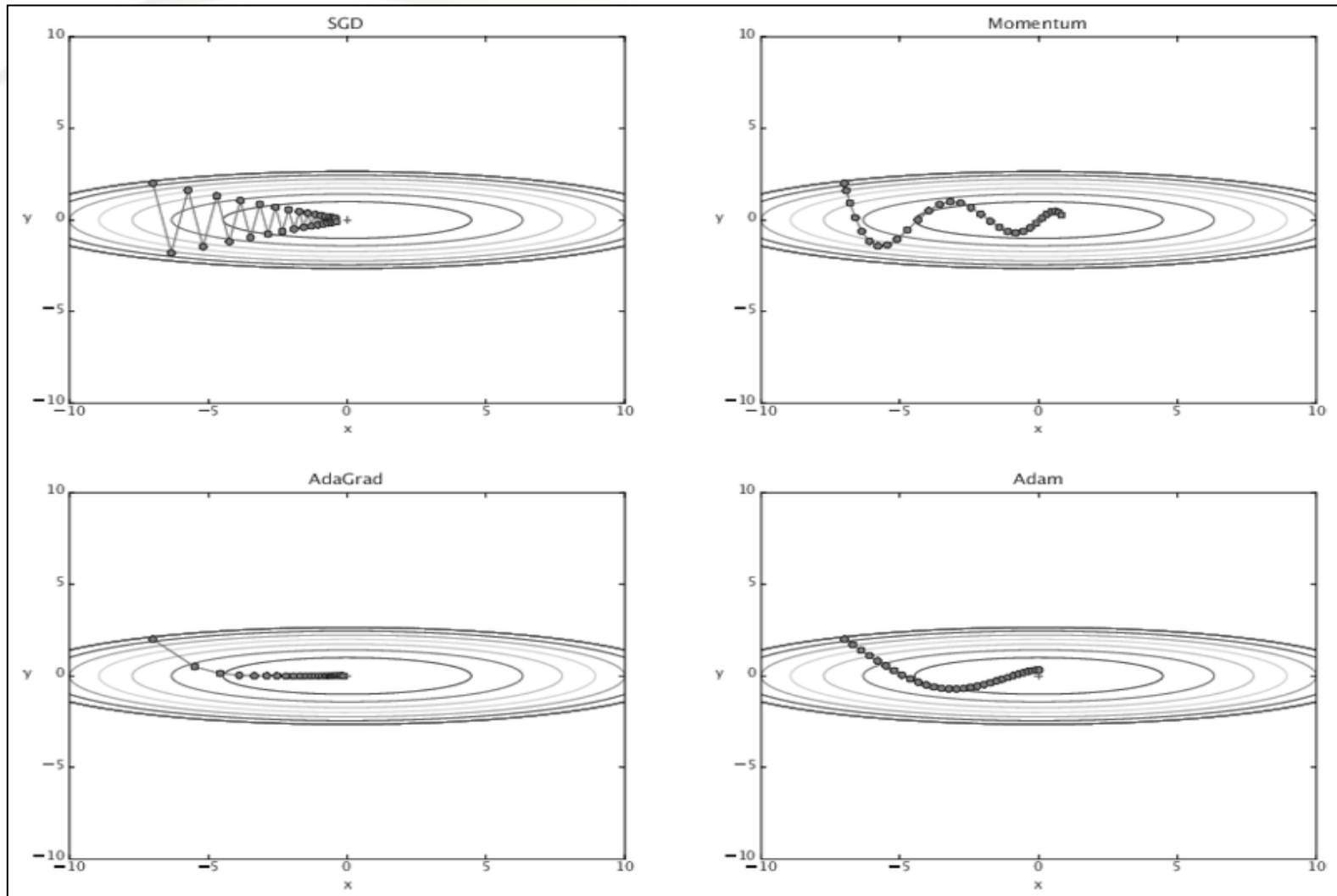
행렬의 원소별 곱

▪ Adam (Adaptive Moment Estimation)

- » 비교적 최근에 제안된 기법 (2015년)
- » 모멘텀과 AdaGrad (또는 RMSProp)을 융합한 방법 → 효율적 탐색 진행
- » 하이퍼파라미터 편향이 보정됨

최적화 기법

■ 각 기법 학습 패턴 시각화

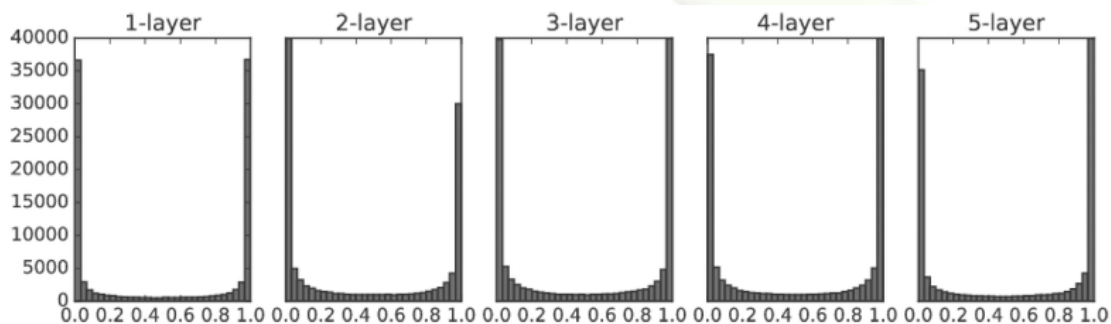


가중치 초기값

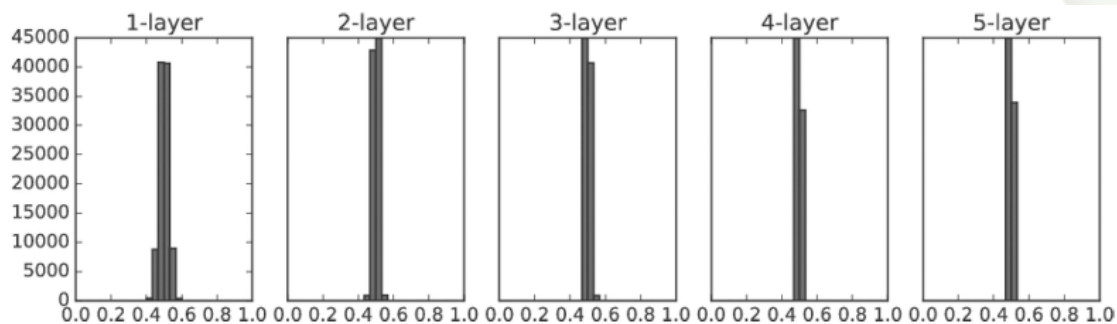
- 초기 가중치 설정이 신경망 학습의 성패에 영향을 미치는 사례가 많음
- 오차 역전파법에서 모든 가중치가 동일하게 갱신되기 때문에
 - » 초기 가중치를 0으로 설정하는 경우 또는 초기 가중치를 균일한 값으로 설정하는 경우 학습이 정상적으로 수행되지 않음
 - » 따라서 가중치 값을 무작위로 설정해야 함

가중치 초기값

- 초기값 설정에 따른 은닉층 활성화 값 분포
 - » 표준편차가 1인 정규분포로 초기화한 경우
 - 역전파의 기울기 값이 사라지는 기울기 소실문제 발생

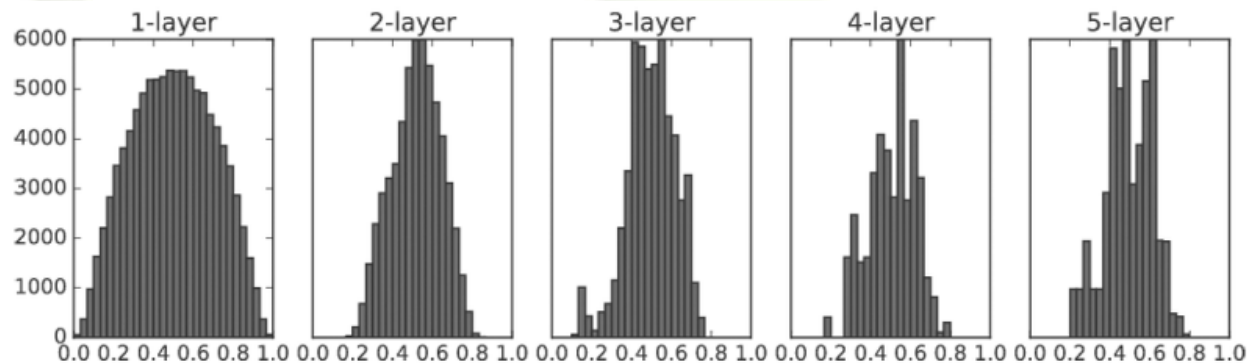


- » 표준편차가 0.01인 정규분포로 초기화한 경우
 - 다수의 뉴런이 거의 같은 값을 출력 → 뉴런을 여러 개 둔 효과 상실

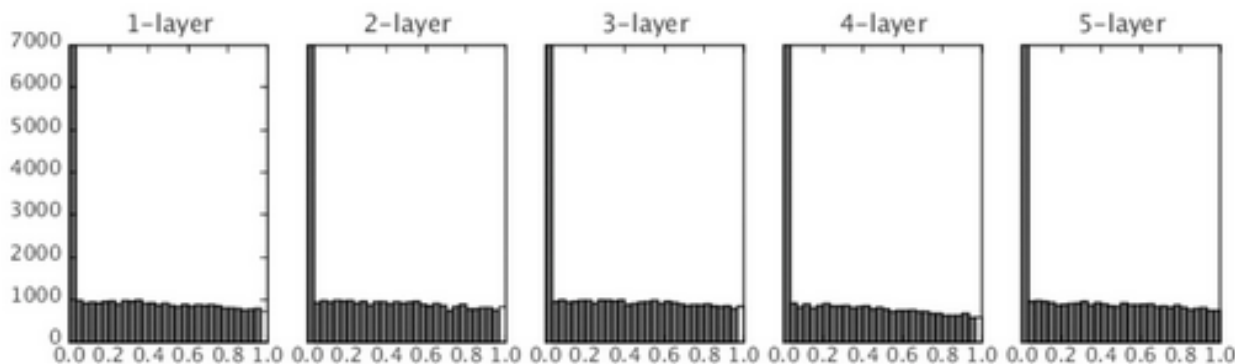


가중치 초기값

- Sigmoid, tanh 활성화함수를 사용하는 경우 Xavier 초기값을 사용하면 은닉층의 활성화 값의 분포가 좋아짐

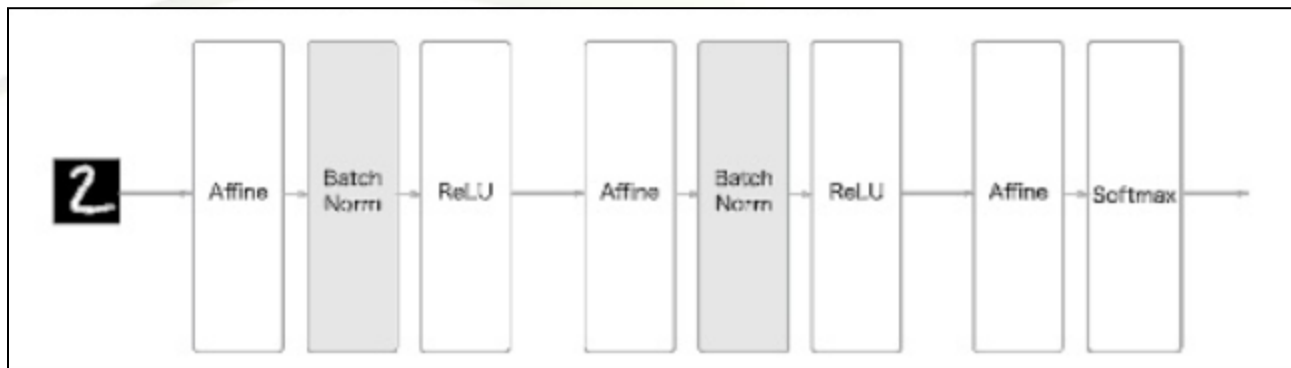


- ReLU를 활성화함수로 사용하는 경우 He 초기값을 사용하는 것이 좋음



배치 정규화

- 신경망의 각 층에서 활성화 값이 적당히 분포되도록 조정하는 기법



- » 미니 배치를 단위로 정규화
- » 평균 0 분산 1이 되도록 정규화

- 특징
 - » 학습 속도 개선
 - » 초기값에 대한 낮은 의존도
 - » 오버피팅 억제 (드롭아웃 등의 필요성 감소)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots x_m\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

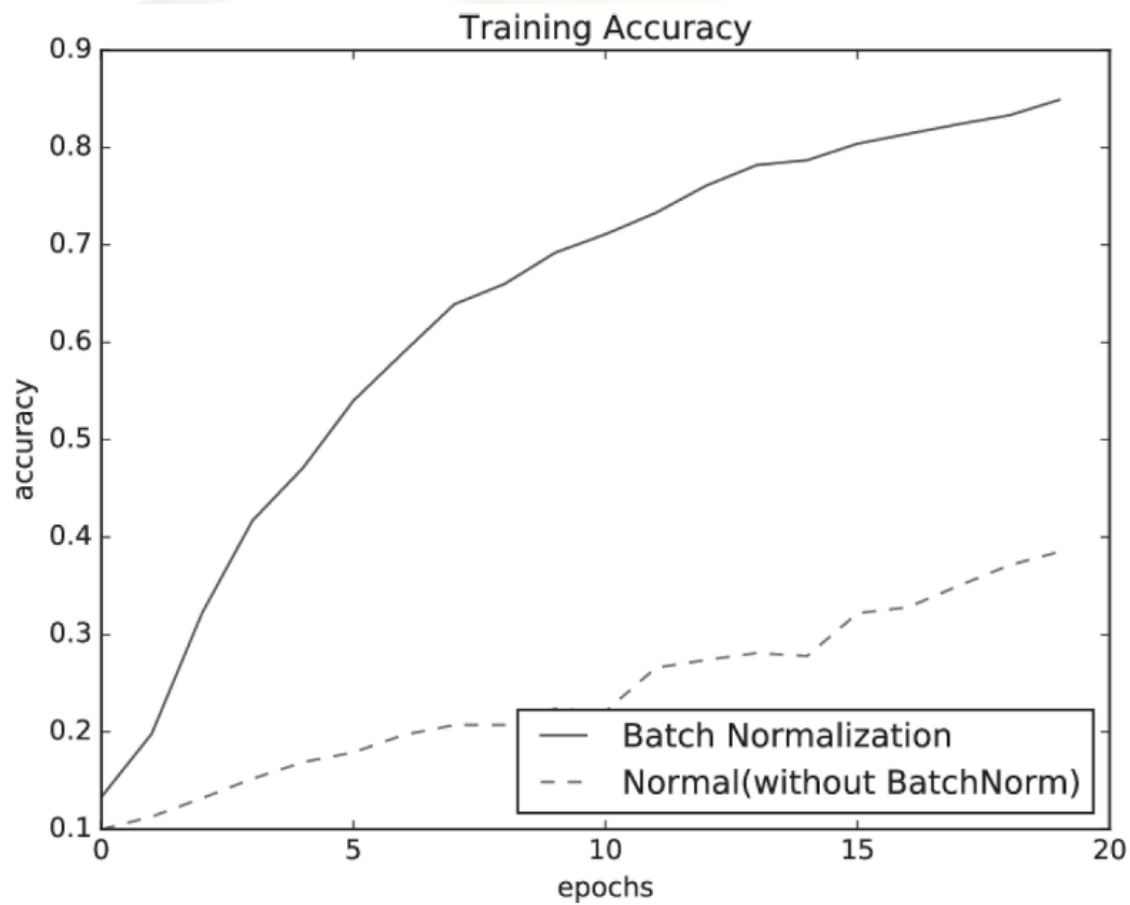
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

배치 정규화

- 손글씨 숫자 인식 학습에서 배치 정규화 효과



과적합 (오버피팅)

- 모델이 훈련 데이터에 대해서는 매우 높은 정확도를 보이지만 테스트 데이터에 대해서는 상대적으로 낮은 정확도를 나타내는 경우

- 오버피팅 억제 기법

- » 가중치 감소

- 큰 가중치에 대해 상응하는 패널티를 부과해서 오버피팅 억제
- L1(가중치 절대값 사용), L2(가중치 제곱항을 사용) 등의 규제 사용

- » 드롭아웃

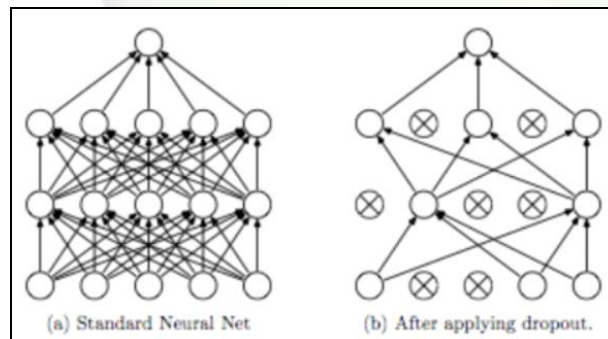
- 뉴런을 임의로 삭제하면서 학습하는 기법
- 훈련시에 은닉층의 뉴런을 무작위로 삭제

L1 Regularization

$$\text{Cost} = \sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2 + \lambda \sum_{j=0}^M |W_j|$$

L2 Regularization

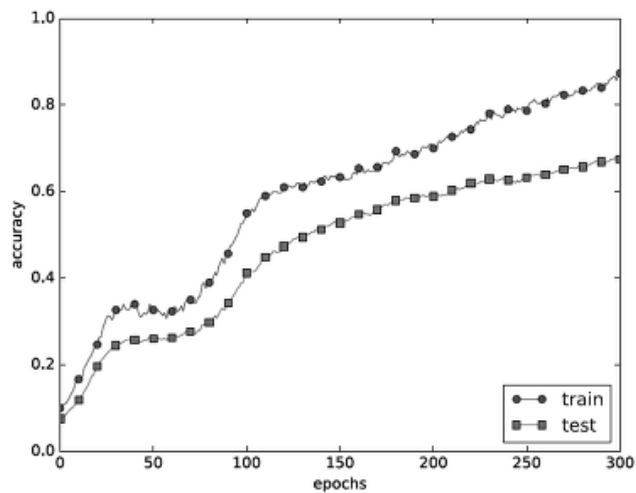
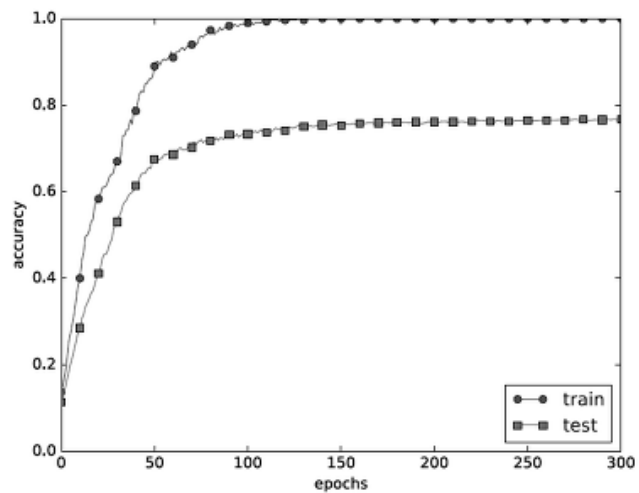
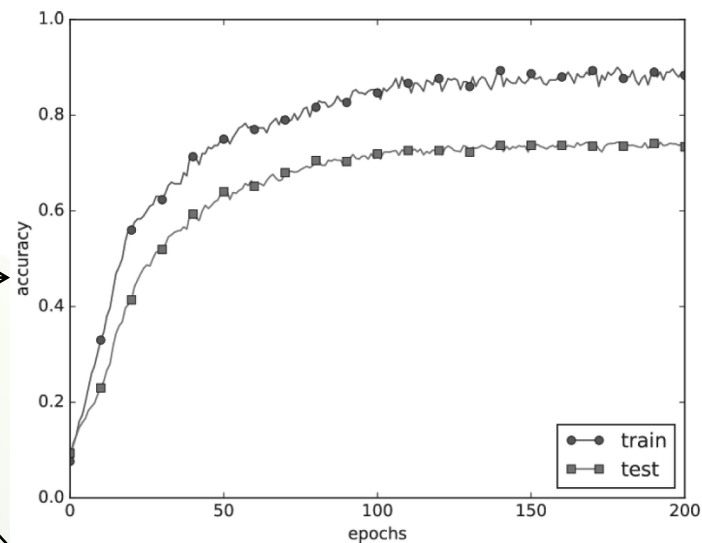
$$\text{Cost} = \underbrace{\sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^M W_j^2}_{\text{Regularization Term}}$$



과적합 (오버피팅)

■ 과적합 억제 기법 구현

- » 가중치 감소 (규제 적용)
- » 드롭아웃



하이퍼 파라미터 값 탐색

- 검증 데이터 사용

- » 하이퍼 파라미터 탐색을 위해 테스트 데이터를 사용할 경우 모델이 테스트 데이터에 오버피팅되는 결과 도출
- » 훈련 데이터의 일부를 검증 데이터로 사용해서 하이퍼 파라미터 값 탐색

- 최적화

- » 하이퍼 파라미터의 최적 값이 존재하는 범위를 조금씩 줄여가는 과정
- » 무작위로 선택한 로그 스케일 범위에서 하이퍼 파라미터 검증
- » 그리드 서치와 같은 규칙적인 탐색보다는 무작위 샘플링이 더 좋은 결과 도출