6강: 신경망 파트1 - 아키텍쳐 설계

인트로

- 신경망 점수 함수의 예: $s_2 = W_2 \max(0, W_1 x)$ (x: 입력 벡터)
- 가령 $x \in \mathbb{R}^{3072}$이고 W_1은 100x3072 행렬, W_2는 10x100 행렬일 수 있음
- $\max(0, x)$ 함수(ReLU 함수)는 일반적으로 사용하는 비선형 함수
- 비선형 함수를 제외했을 때의 함수 $W_2 W_1 x$ 는 단지 선형 함수에 불과함
- 비선형성이 있어야 선형함수가 못하는 것들을 할 수 있음
- W_1, W_2는 확률적 경사하강법으로 학습됨

인트로

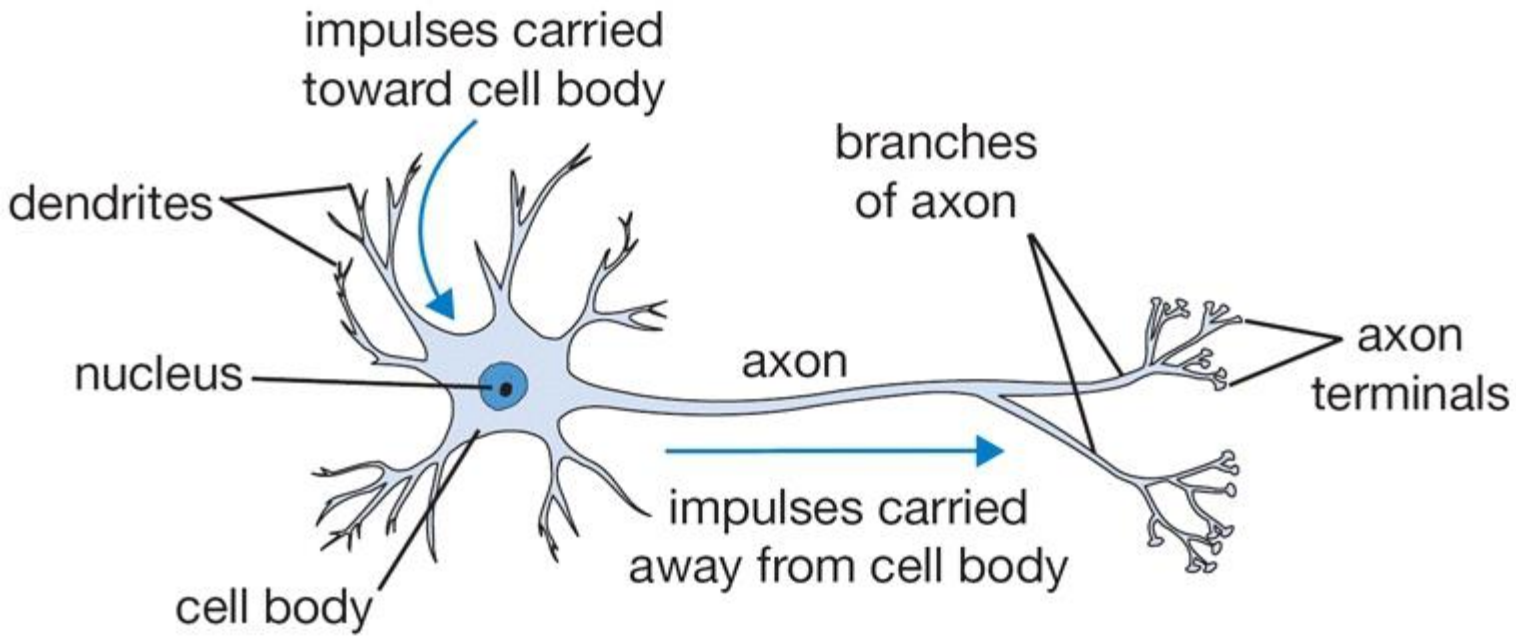
- 세 층 신경망(three-layer neural network) 예시
  - $s = W_3 \max(0, W_2 \max(0, W_1 x))$
  - 크기: 3072 -> ? -> ? -> 10
  - 히든 벡터(hidden vector)의 차원은 하이퍼파라미터

생물학적 영감

- 신경망 분야는 원래 생물학적 신경시스템 모델링이 목표였지만 이후로는 기계 학습 작업에서 좋은 성능을 달성하는 문제로 발전함
- 뉴런
  - 뇌의 기본 계산 단위
  - 약 860억개의 뉴런이 있음
  - 대략 10^{14}~10^{15} 개의 시냅스로 연결됨

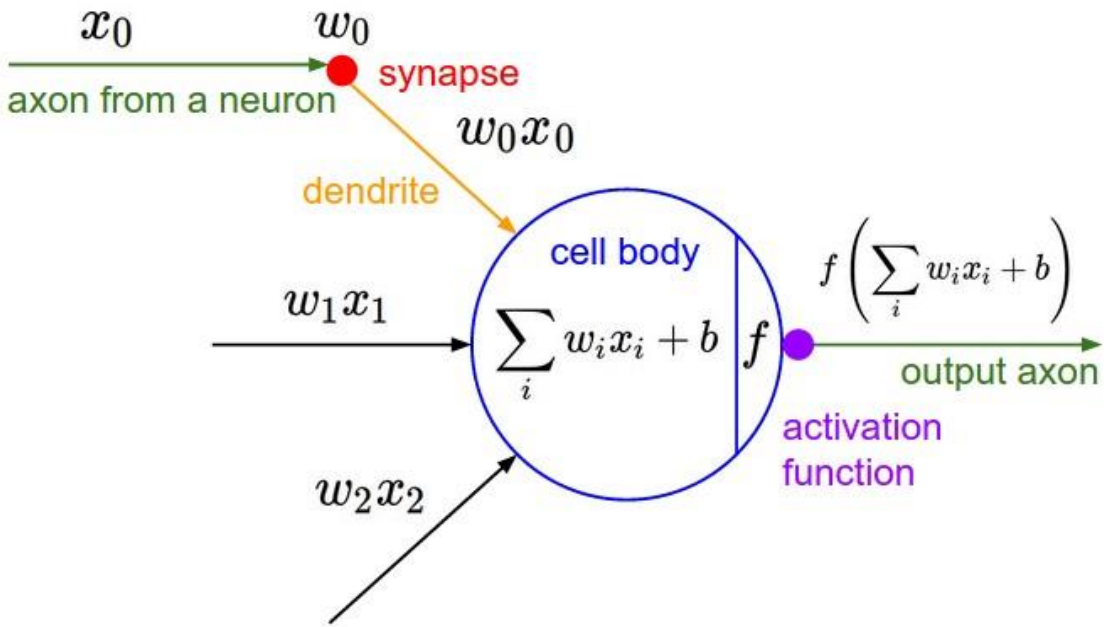
생물학적 영감

- 각 뉴런은 수상돌기로부터 입력 신호를 받아 axon을 통해 출력 신호를 생성함
- axon 신호는 결국 나뉘어지고 시냅스를 통해 다른 뉴런의 수상돌기에 연결됨



생물학적 영감

- axon 신호는 시냅스 강도에 따라 다른 뉴런의 수상돌기와 곱셈적으로 상호 작용함
- 시냅스 강도 w는 학습 가능하며, 이는 한 뉴런이 다른 뉴런에 미치는 영향 강도를 제어함
- 수상돌기로부터 받은 신호들은 합산되며, 이 합계가 특정 임계값 이상이면 뉴런은 활성화되어 스파이크를 보냄
- 스파이크의 빈도만이 정보를 전달한다고 가정하며, 이는 활성화 함수 f로 모델링됨



생물학적 영감

- 역사적으로 활성화 함수의 일반적인 선택은 시그모이드 함수

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

$$\sigma(\sum_i w_i x_i + b)$$

```
class Neuron(object):
  # ...
  def forward(self, inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

생물학적 영감

- 생물학적 뉴런에 대한 이 모델은 매우 대략적인 모델
  - 다양한 종류의 뉴런이 존재함
  - 뉴런의 수상돌기는 복잡한 비선형 계산을 수행함
  - 시냅스는 하나의 가중치가 아닌 복잡한 비선형 동적 시스템
  - 많은 경우 스파이크의 정확한 시점이 중요하므로 스파이크 빈도로 근사하는 것이 유효하지 않을 수 있음

하나의 뉴런을 하나의 분류기로

- 하나의 뉴런을 하나의 분류기로 대응시킬 수 있음
- 뉴런의 출력에 적절한 손실 함수를 사용하면 단일 뉴런을 선형 분류기로 바꿀 수 있음

이진 소프트맥스 분류기

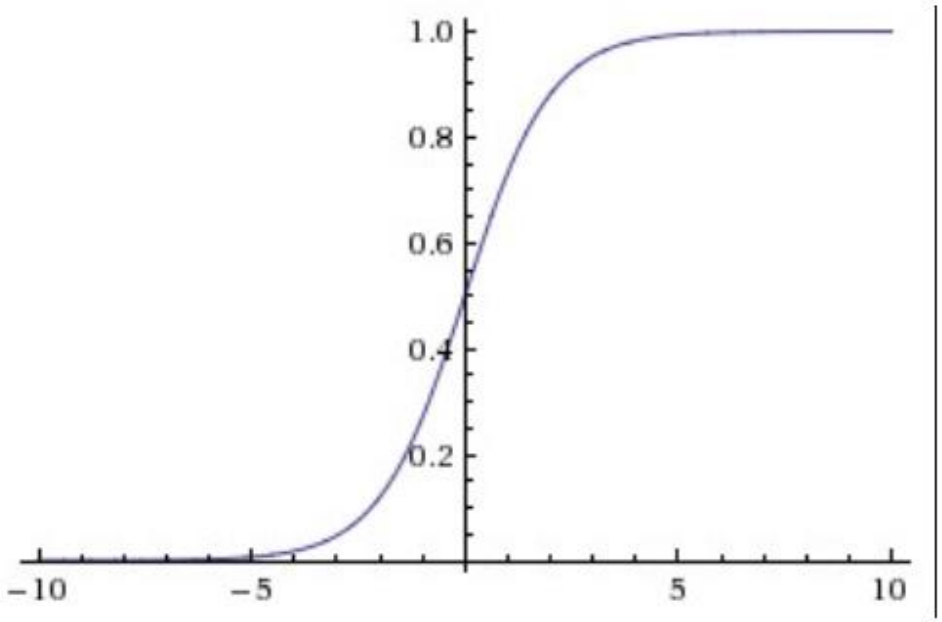
- 뉴런의 출력인 $\sigma(\sum w_i x_i + b)$를 클래스 중 하나일 확률 $P(y_i = 1 \mid x_i; W)$로 해석 가능
- 다른 클래스일 확률은 $P(y_i = 0 \mid x_i; W) = 1 - P(y_i = 1 \mid x_i; W)$
- 크로스 엔트로피 손실 사용 가능
- 이를 최적화하면 이진 소프트맥스 분류기 (혹은 로지스틱 회귀)

이진 SVM 분류기

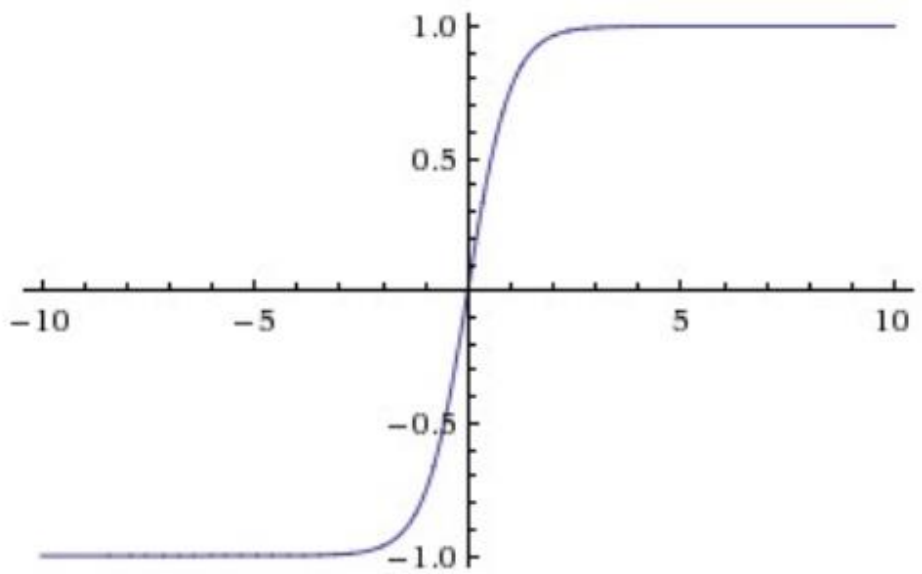
- 최대 마진 힌지 손실(max-margin hinge loss)을 뉴런의 출력에 연결하고 이를 이진 SVM 분류기로 훈련시킬 수 있음

활성화 함수(activation function)

- 모든 활성화 함수(activation function)은 단일 숫자를 받아 고정된 수학적 연산을 수행함



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



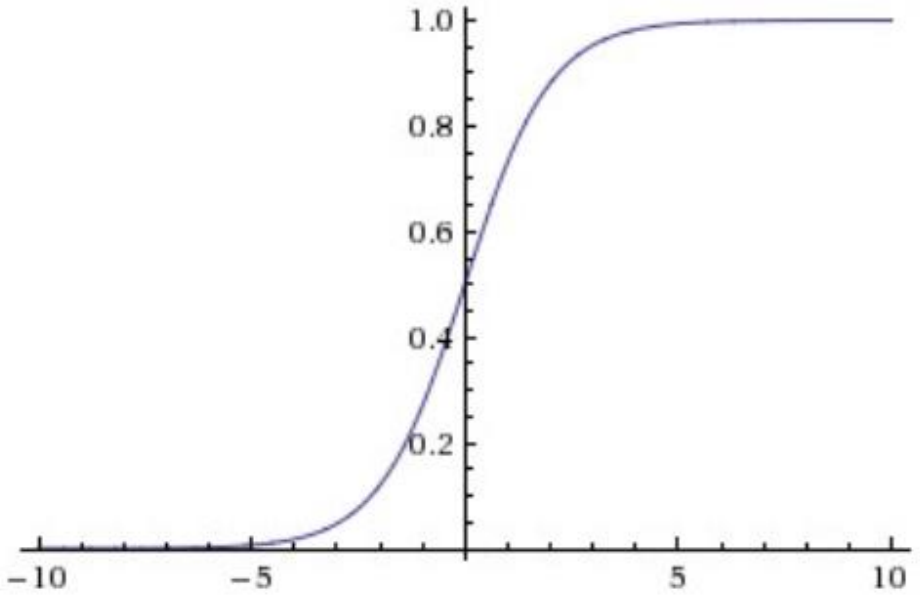
tanh(x)

시그모이드(Sigmoid)

- 시그모이드(Sigmoid) 함수

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

- 뉴런의 발화율을 잘 설명하는 것처럼 보여 과거에는 자주 사용됨
- 두 가지 큰 단점으로 최근에는 거의 사용되지 않음

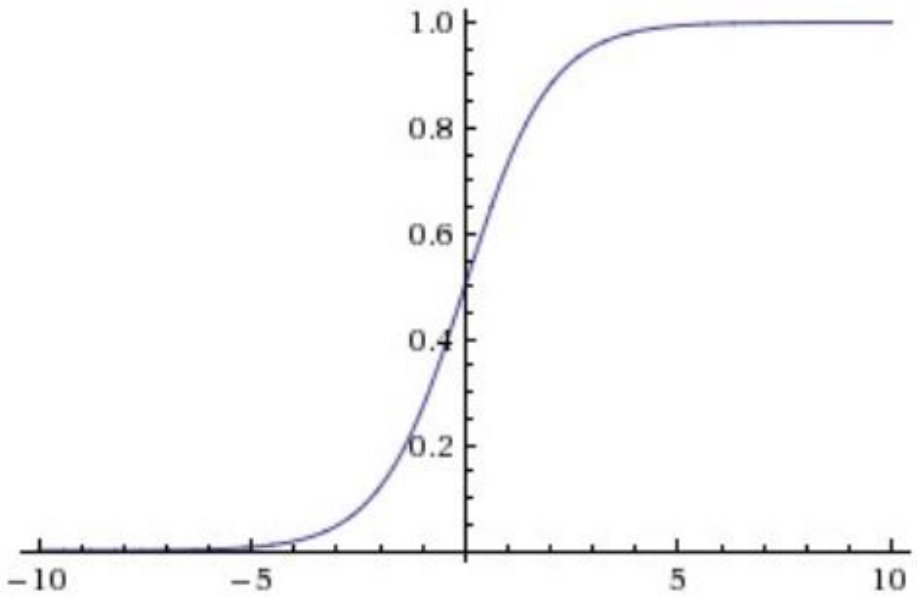


시그모이드 함수 단점1

- 시그모이드 함수는 포화되어 그래디언트를 소멸시킴
- 뉴런의 출력이 0 또는 1의 꼬리 쪽에서 포화되면 기울기는 거의 0
- 역전파 과정에서 그래디언트가 죽게 되어 학습이 제대로 안 됨

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

- 시그모이드 뉴런의 가중치는 포화되지 않도록 특별한 주의가 필요

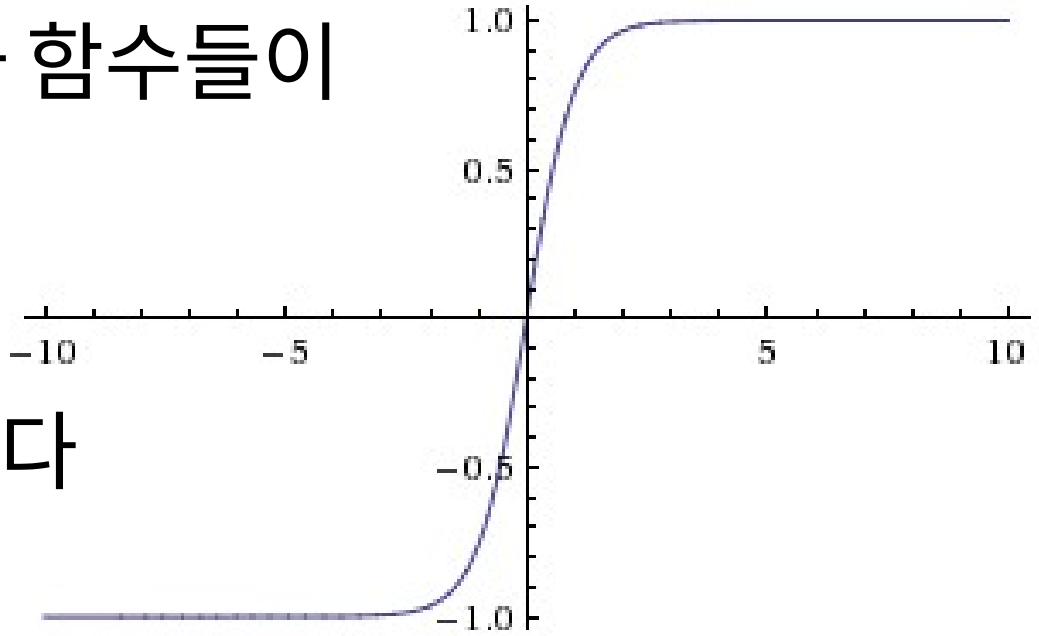


시그모이드 함수 단점2

- 시그모이드 함수 출력은 zero-centered가 아님
- 신경망의 뒤쪽 층에서 처리되는 뉴런들은 zero-centered가 아닌 데이터를 받게 됨
- 이는 경사하강법 동작에 영향을 미침
- 가령 $f = w^T x + b$에서 $x > 0$인 경우, 즉, 입력이 항상 양수인 경우 역전파 동안 기울기는 모두 양수가 되거나, 모두 음수가 됨

tanh 함수

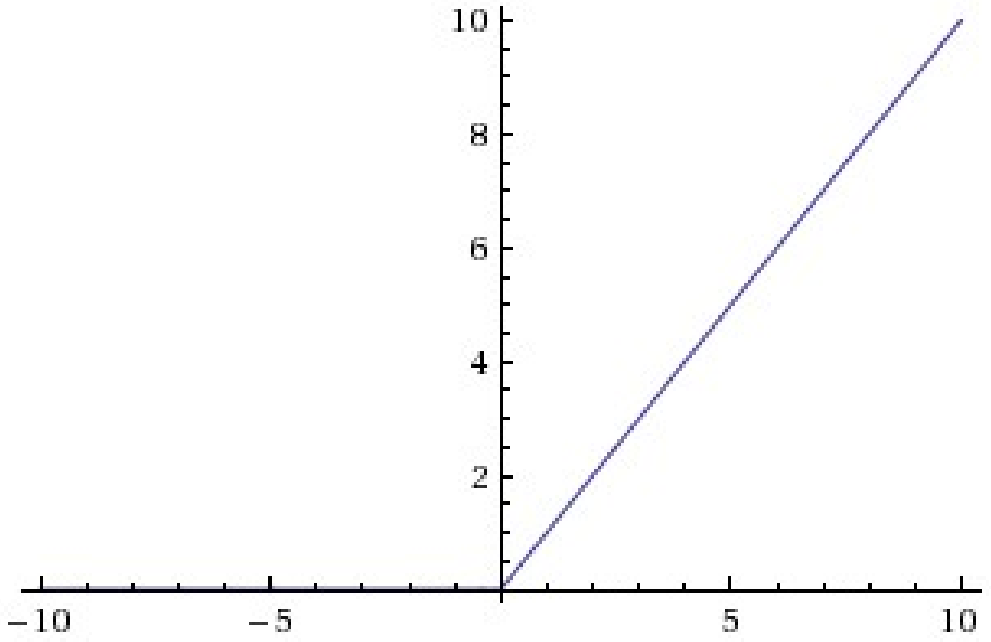
- 시그모이드 함수와 마찬가지로 tanh 출력은 포화된다는 단점 존재
- 이런 이유로 tanh보다는 다른 활성화 함수들이 더 사용됨
- tanh는 시그모이드 함수와 달리 그 출력이 zero-centered이므로 tanh보다 항상 선호됨



$$\tanh(x) = 2\sigma(2x) - 1$$

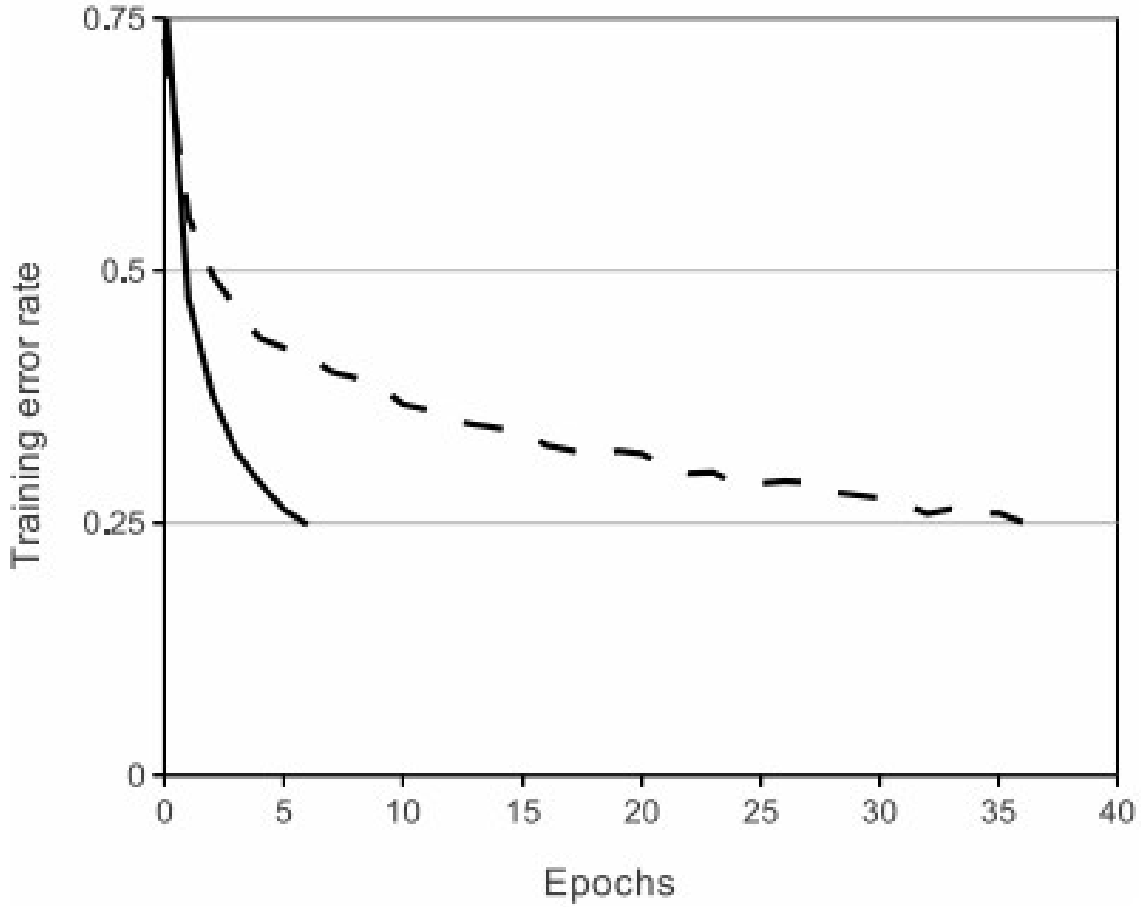
ReLU

- ReLU(Rectified Linear Unit) 함수는 대표적인 활성화 함수
- ReLU 함수는 $f(x) = \max(0.x)$를 계산함



ReLU 장점1

- 시그모이드/tanh 함수에 비해 확률적 경사 하강법의 수렴을 크게 가속화함 (Krizhevsky 등 논문: 6배 가속화)
- 선형이고 포화되지 않는 형태 때문
- 그림에서 실선, 점선은 각각 ReLU 함수 및 tanh 함수를 사용한 경우
- ReLU의 경우가 tanh 보다 훨씬 더 빨리 수렴

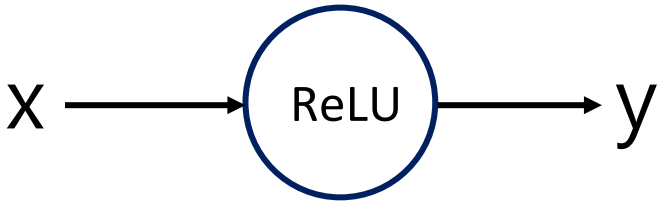


ReLU 장점2

- 시그모이드/tanh 함수는 지수함수 등 비용이 많이 드는 연산을 포함
- ReLU는 단순히 0에서 임계값을 설정하여 구현

ReLU 단점

- 훈련 중에 죽을 수 있음
- ReLU 유닛이 다시는 활성화되지 않도록 가중치가 업데이트될 수 있음
- 이 때부터, 그 ReLU 유닛을 통과하는 기울기는 계속 0
- ReLU 함수의 역전파
  - x>0이면 df/dx = df/dy
  - x<0이면 df/dx = 0



Leaky ReLU

- ReLU의 죽는 문제를 해결하기 위한 시도 중 하나는 Leaky ReLU
- 작은 상수 α에 대해 $f(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x)$ 계산
- Leaky ReLU의 성공 사례들이 보고됐지만 항상 일관되지는 않음

MaxOut

- 지금까지는 $f(w^T x + b)$ 형태를 사용함
- 다른 유형의 유닛인 Maxout이 제안됨
- 이는 ReLU와 Leaky ReLU 를 일반화함
- Maxout은 $\max(w_1^T x + b_1, w_2^T x + b_2)$를 계산함
- Maxout은 ReLU의 모든 이점을 누리면서도 죽는다는 단점이 없음
- 그러나 ReLU 대비 각 뉴런마다 파라미터를 2배로 사용하므로 총 파라미터의 개수가 매우 높아짐

활성화 함수 Q&A

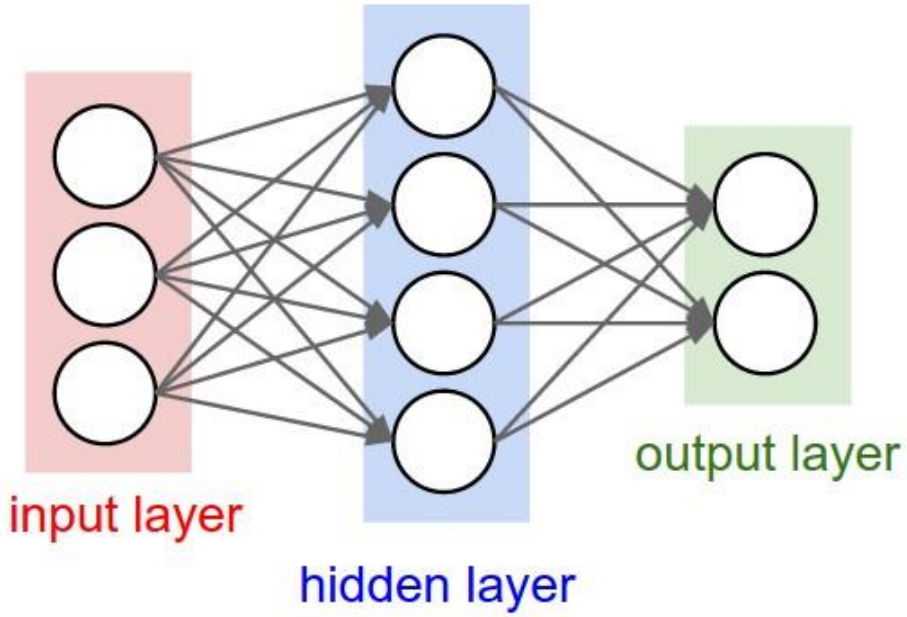
- Q1)
  - 여러 활성화 함수를 섞어 쓰는 것도 가능한지?
- A1)
  - 근본적으로 문제가 되지는 않지만, 동일한 네트워크에서 다른 유형의 활성화함수를 섞어 쓰는 것은 매우 드뭄

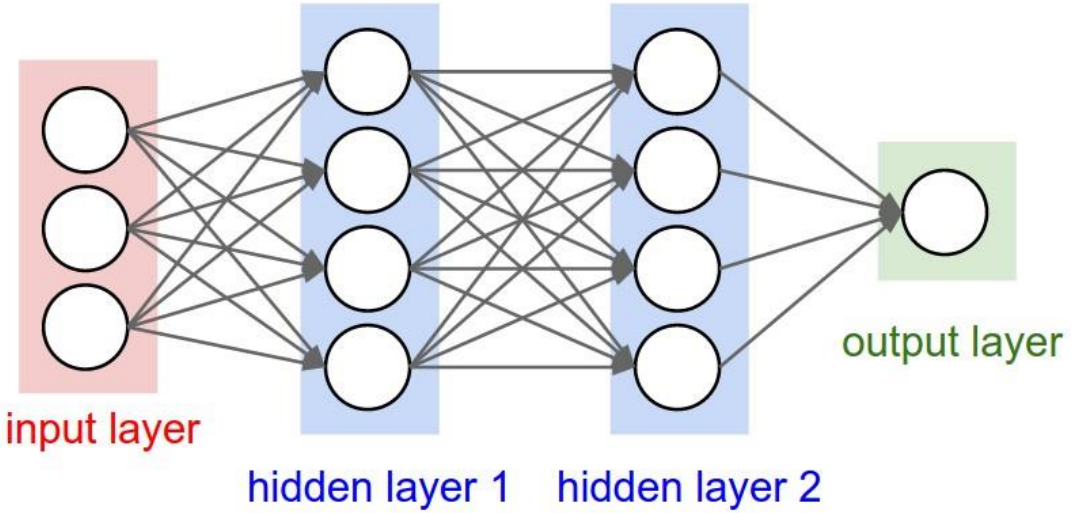
활성화 함수 Q&A

- Q2)
  - 여러 활성화 함수 중 어떤 걸 사용해야 하는지?
- A2)
  - ReLU 함수를 사용하되, 학습률에 주의하고 죽은 유닛의 비율을 모니터링 할 수 있음
  - 죽는 것이 걱정되면 Leaky ReLU나 Maxout 시도 가능
  - 시그모이드는 절대 사용하지 말 것
  - tanh를 시도해볼 수는 있지만 ReLU/Maxout보다는 성능이 떨어질 것이 예상됨

신경망 아키텍쳐

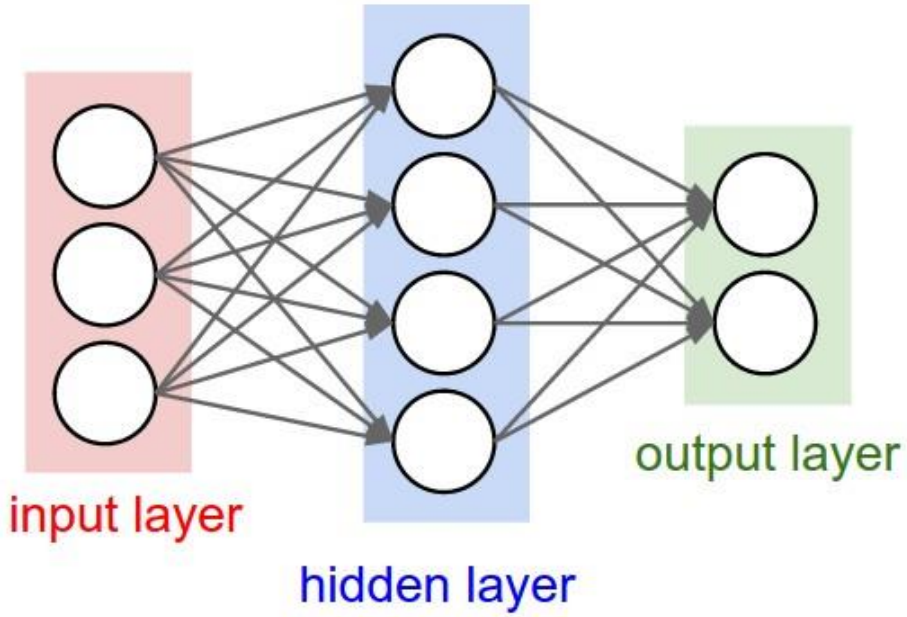
- 신경망은 비순환 그래프로 연결된 뉴런들의 모음으로 모델링됨
- 신경망 모델은 종종 명확한 뉴런 계층으로 구성됨
- 가장 흔한 계층은 완전연결 계층(fully-connected layer)
- 두 인접한 계층 사이 뉴런은 완전히 연결되며 각 계층 내 뉴런은 연결이 없음

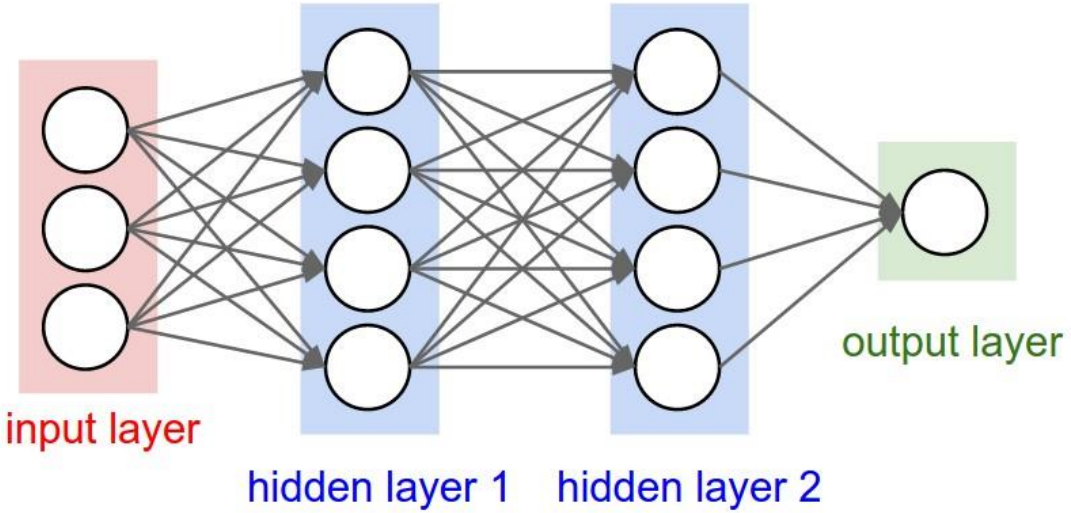




신경망 아키텍쳐

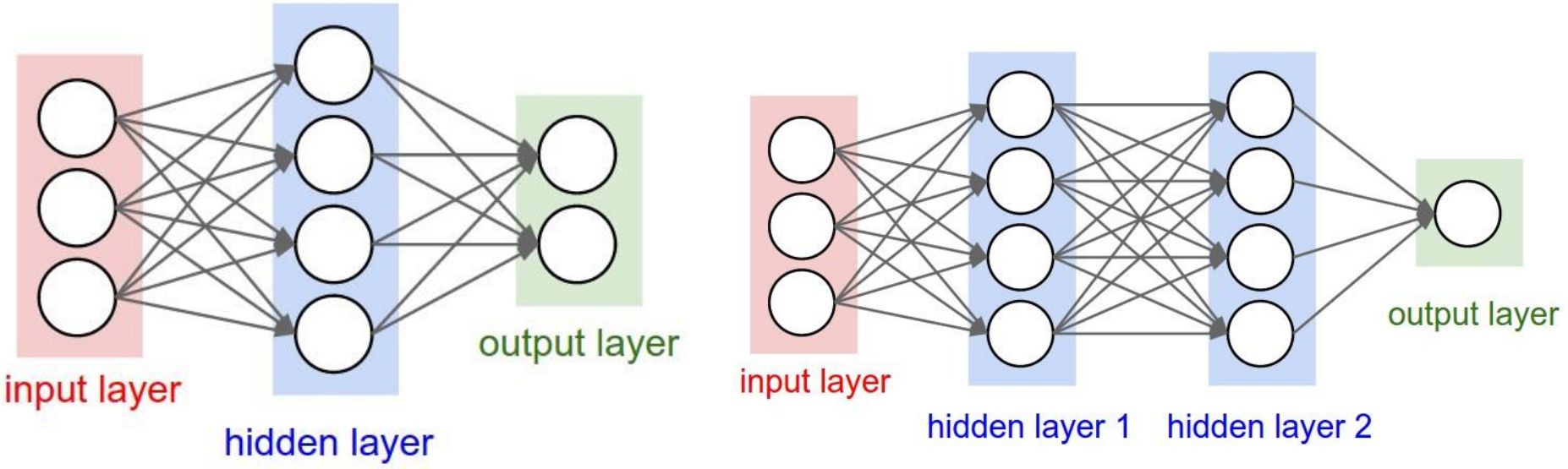
- 왼쪽 그림은 2계층 신경망이며 입력 계층(input layer), 은닉 계층(hidden layer), 출력 계층(output layer)을 가짐
- 오른쪽 그림은 3계층 신경망이며 입력 계층, 2개의 은닉 계층, 출력 계층을 가짐





이름 규칙

- N계층 신경망(N-layer neural network)이라 부를 때 입력 계층은 카운트하지 않음
- 단일 계층 신경망은 은닉 계층이 없는 신경망
- 신경망은 인공신경망(artificial neural networks, ANN) 혹은 다중 계층 퍼셉트론(multi-layer perceptron)으로 바꿔 불리기도 함

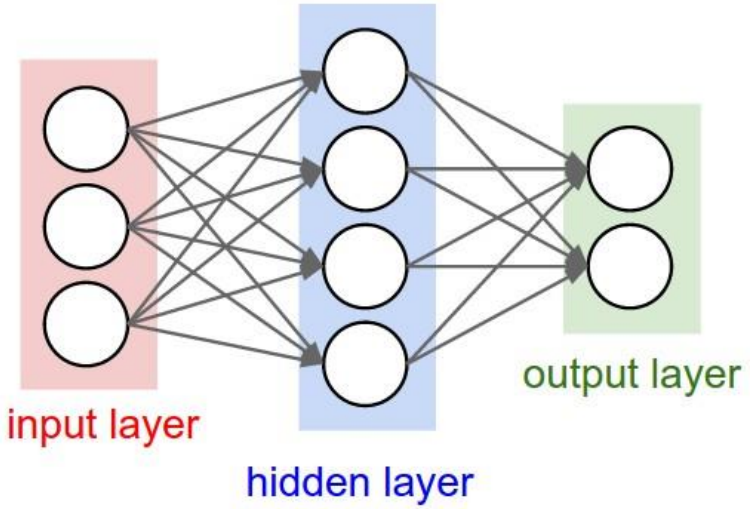


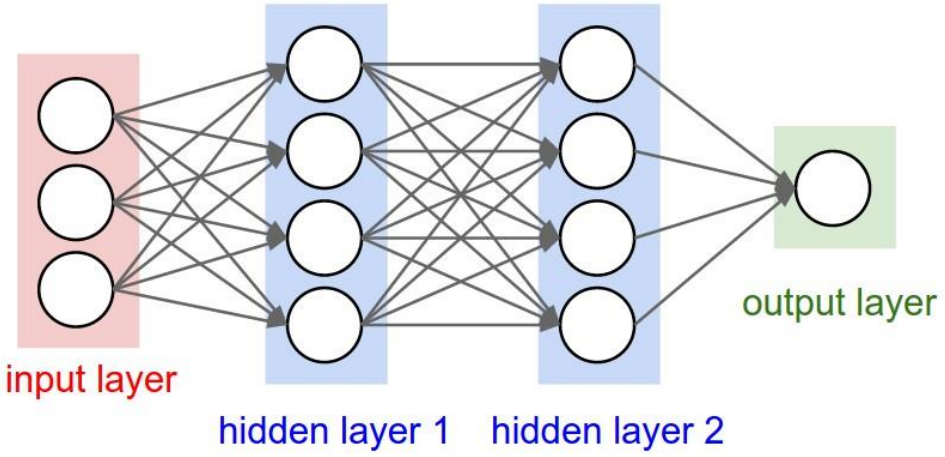
출력 계층

- 신경망의 다른 계층과는 달리 출력 계층의 뉴런은 보통 활성화 함수가 없음
- 마지막 출력 계층 결과가 일반적으로 클래스 점수를 나타내기 때문

신경망 크기 측정

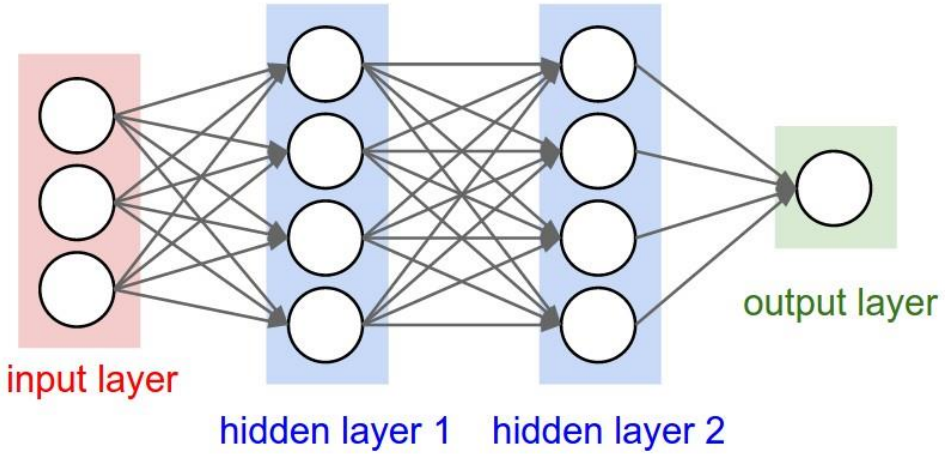
- 신경망의 크기를 측정하는 지표
  - (1) 뉴런의 수
  - (2) 파라미터의 수
- 왼쪽 신경망: 6개의 뉴런, 26개의 파라미터
- 오른쪽 신경망: 9개의 뉴런, 41개의 파라미터





신경망 연산(feed-forward) 예시

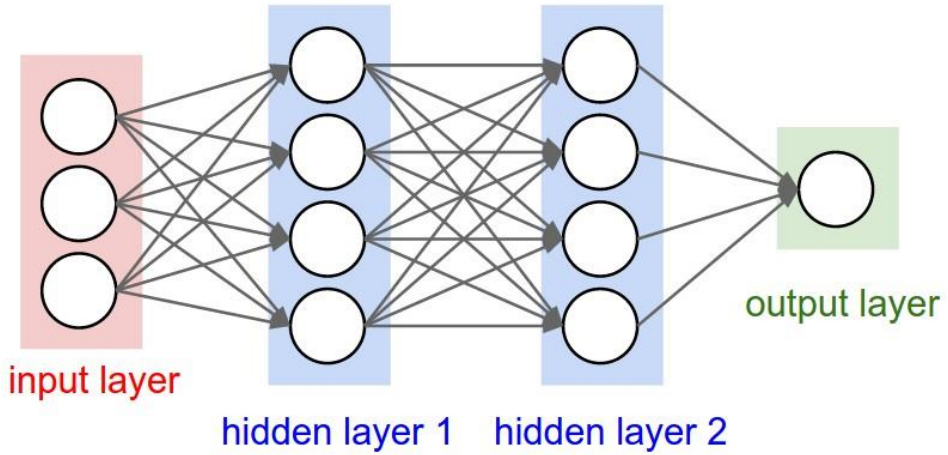
- 피드포워드(feed-forward): 데이터가 입력 계층에서 출력 계층까지 한 방향으로만 이동하는 신경망
- 행렬 곱셈과 활성화 함수 계산을 반복적으로 수행함
  - $h_1 = f(W_1 x + b_1)$
  - $h_2 = f(W_2 h_1 + b_2)$
  - $y = W_3 h_2 + b_3$



신경망 연산(feed-forward) 예시

- $h_1 = f(W_1 x + b_1)$
- $h_2 = f(W_2 h_1 + b_2)$
- $y = W_3 h_2 + b_3$

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```



신경망으로 모델링 가능한 함수

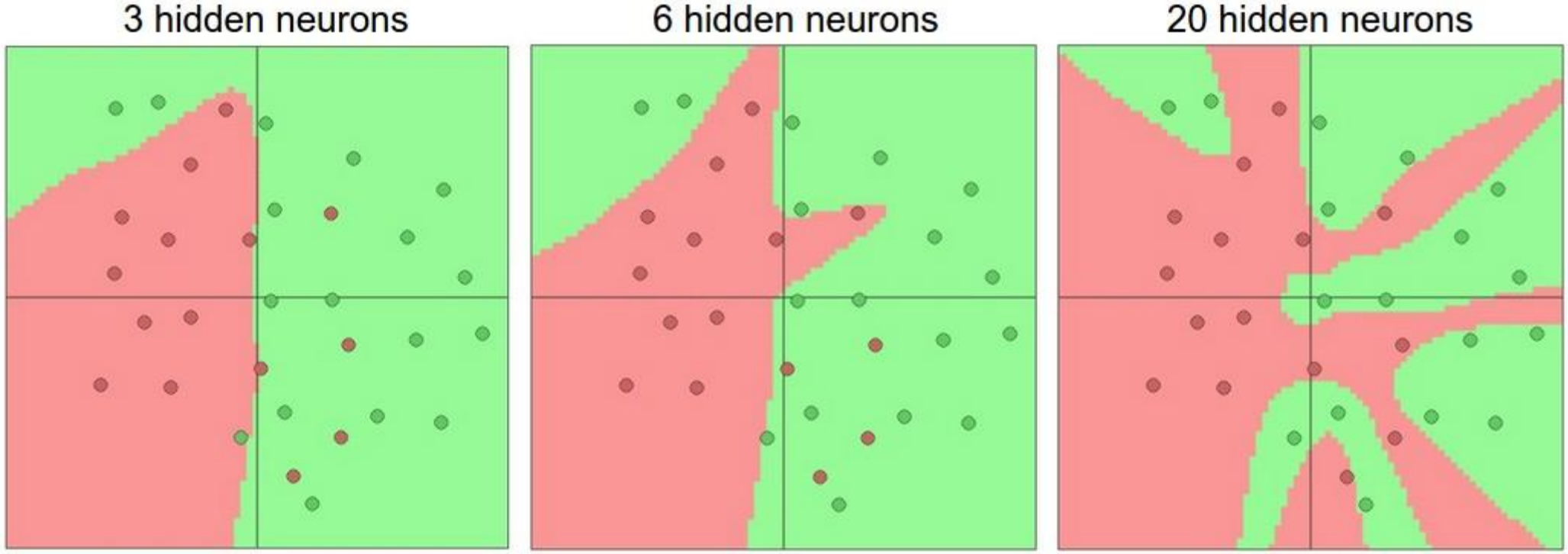
- 신경망은 하나의 파라미터화된 함수로 생각 가능
- 이 함수로 모델링 할 수 있는 함수들은?
- 적어도 하나의 은닉 계층을 가진 신경망은 어떤 연속함수든 잘 근사할 수 있음
- 단, 어떤 함수를 잘 근사하더라도 실제 데이터의 통계적 특성과 맞지 않으면 선호되지 않음
  - ex) $g(x) = \sum_i \ c_i \ 1(a_i < x < b_i)$

신경망 계층 수

- 종종 3계층 신경망이 2계층 신경망보다 성능이 좋음
- 그렇지만 4계층 이상으로 깊게 가는 것은 거의 도움이 되지 않음
- 단, 컨볼루션 신경망에서는 깊은 경우 잘 작동하는 경우가 많음
  - ex) 20, 32, 44, 56, 110
- 컨볼루션 신경망에 깊이가 중요한 이유는 이미지에 계층적 구조가 포함되어있기 때문
  - ex) 얼굴은 눈으로 이루어짐, 눈은 모서리로 이루어짐, …

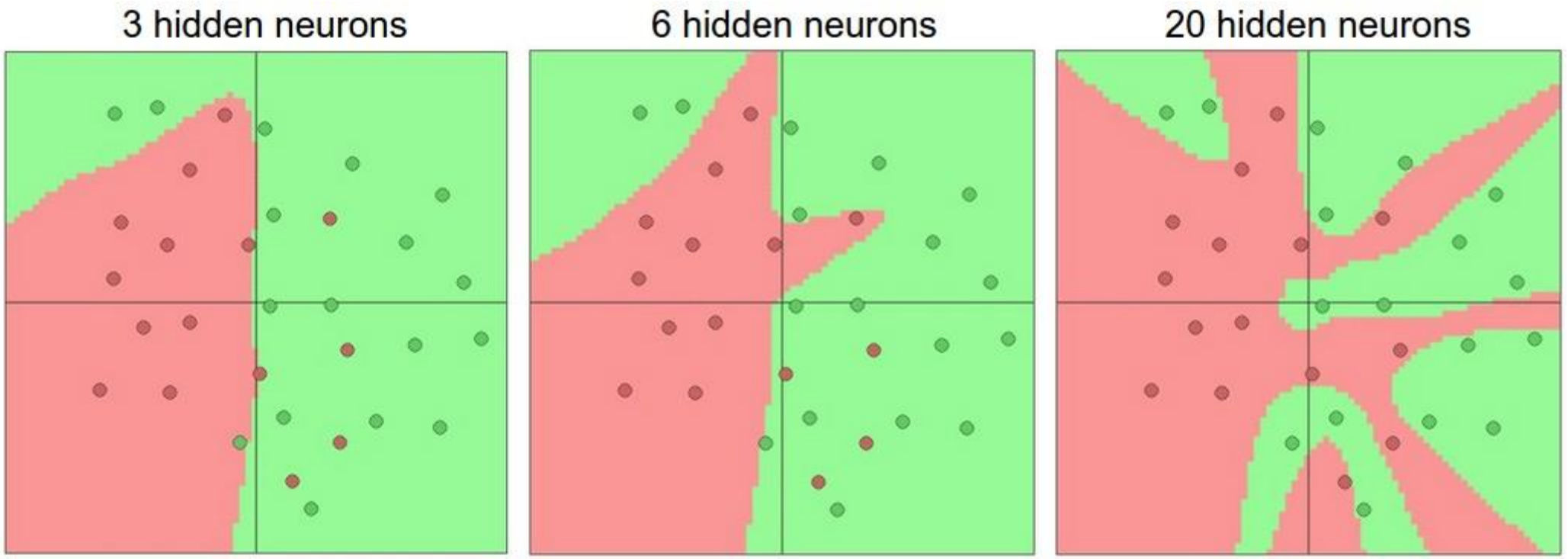
계층의 수와 크기 설정

- 계층의 크기와 수를 늘리면 신경망의 용량이 증가하면서 다양한 함수를 표현할 수 있음
- 가령, 2차원 이진 분류 문제에서, 하나의 은닉 계층을 가진 3개의 신경망을 독립적으로 훈련시키면 다음과 같은 분류기가 됨



계층의 수와 크기 설정

- 계층의 크기를 키우는 것의 장단점
  - 장점: 복잡한 데이터를 분류하는 것이 훈련 가능
  - 단점: 훈련 데이터에 과적합하기 쉬움



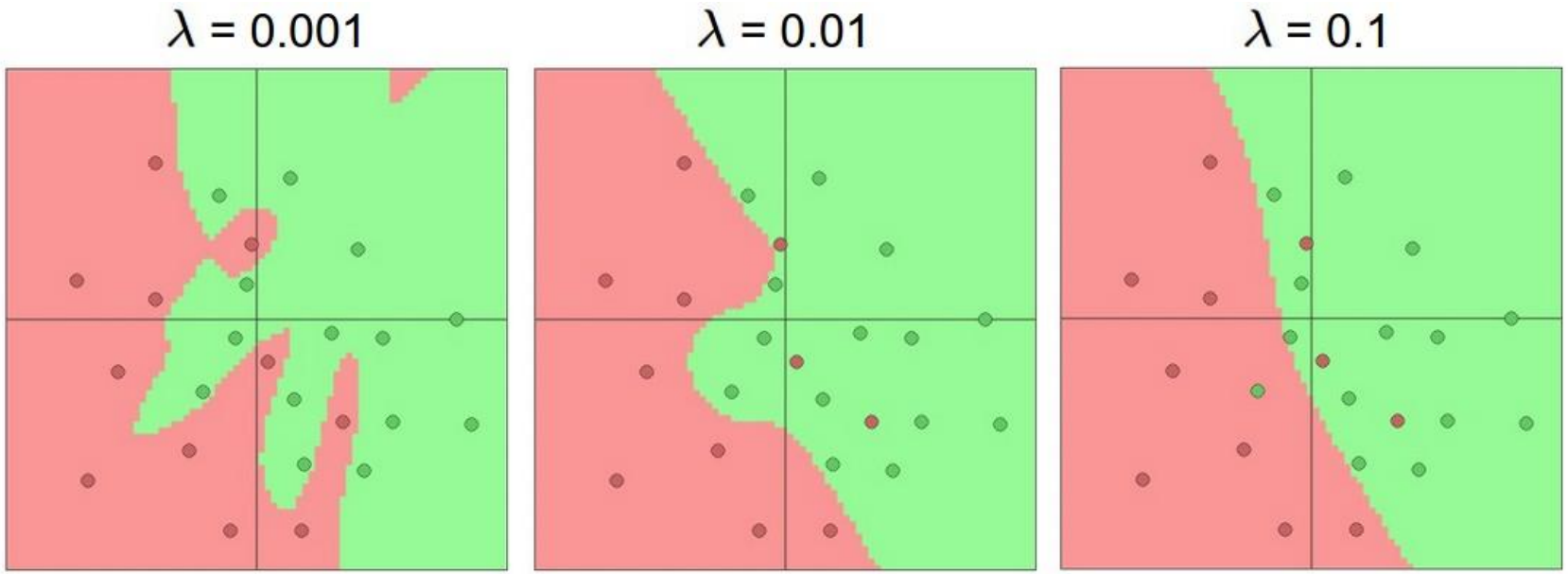
계층의 수와 크기 설정

- 과적합을 막기 위해 작은 신경망을 쓰는 것이 낫다고 할 수는 없음
- L2 정규화, 드롭아웃, 입력 노이즈 등과 같이 신경망에서 과적합을 방지하는 더 선호되는 다양한 방법들이 있음
- 이런 방법들을 사용하여 과적합을 제어하는 것이 뉴런의 수를 줄이는 것보다 항상 좋음

작은 신경망이 덜 선호되는 이유

- 작은 신경망은 경사하강법과 같은 지역적 방법으로 학습시키기 어렵기 때문에 덜 선호됨
- 작은 신경망은, 손실 함수들이 상대적으로 적은 지역적 최소값을 가지지만, 이들 중 많은 것들은 수렴하기 쉽고 높은 손실을 가짐
- 큰 신경망은 훨씬 더 많은 지역적인 최소값을 포함하지만, 이 최소값들은 훨씬 더 작은 손실을 가짐

작은 신경망이 덜 선호되는 이유



- 과적합을 두려워해서 작은 신경망을 사용하지는 말아야함
- 계산 용량이 허용되는 범위에서 큰 신경망을 사용하되, 정규화 기법 등으로 과적합을 제어