Training Neural Networks

조인영

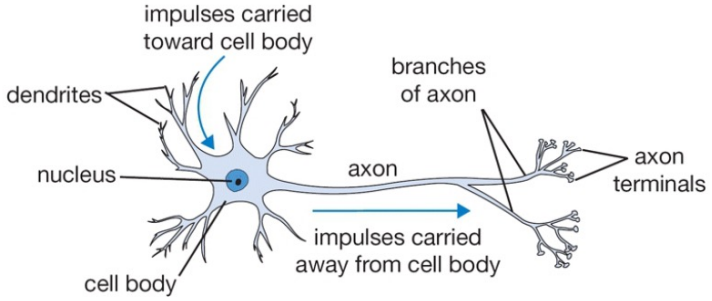
01

Modeling on neuron

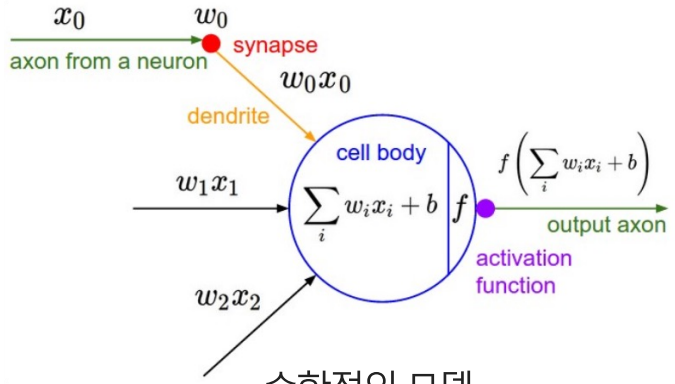
Biological motivation and connections

- 뇌의 기본 계산단위는 뉴런
- 뉴런-수상돌기(dendrites)로 부터 입력신호를 받음

  → 단일축삭(axon)을 따라 출력신호 생성
- 축삭은 분기되어 시냅스를 통해 다른 뉴런의 수상돌기에 연결
- Synapse- 다른 뉴런에 대한 영향의 강도를 제어
- 최종 합계가 특정 임계값을 초과하면 뉴런이 발화



생물학적 뉴런



수학적인 모델

01

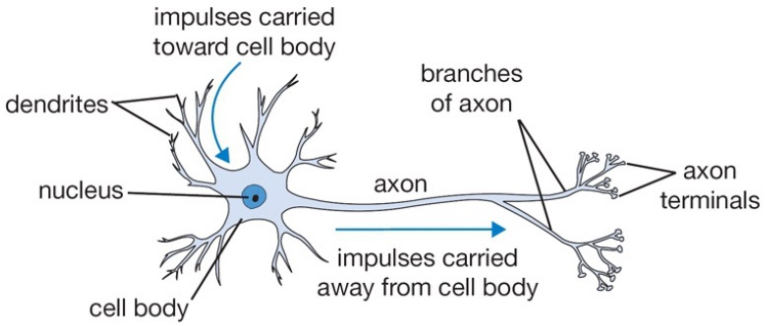
Modeling on neuron

Biological motivation and connections

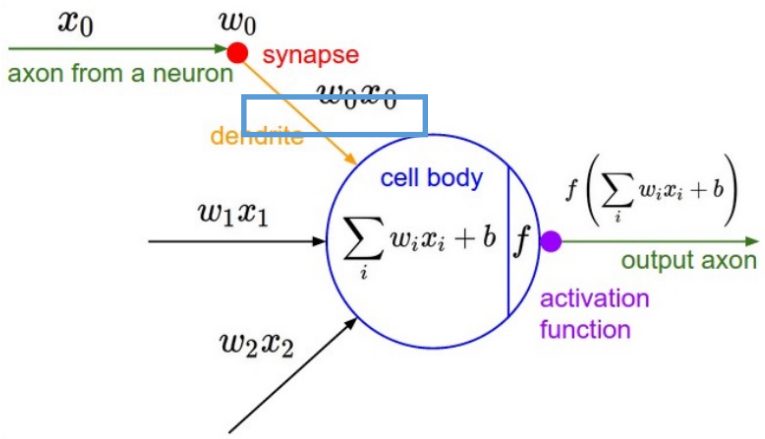
- 활성화 함수- 입력 신호의 총합을 출력 신호로 변환하는 함수.

  → 뉴런의 발사 주기(firing rate)를 모델링함

- Sigmoid function-0과 1범위로 squashes, firing rate으로 좋은 해석이 있음



생물학적 뉴런



수학적인 모델

01

Modeling on neuron

Biological motivation and connections

단일 뉴런에서 신호를 앞으로 전달하는 과정(forward-propagating)

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

입력과 weight에 대해 내적을 연산하고 bias를 더함

Sigmoid함수를 적용해 firing rate를 구함

Coarse model: 실제 뉴런의 수상 돌기에서는 weight 여러 개 ,non linear dynamical system임

02

Modeling on neuron

Single neuron as a linear classifier

- 뉴런의 특정 선형 영역: like(activation near one), dislike(activation near zero)
- 뉴런 출력에 적절한 loss function을 적용함으로써, 단일 뉴런을 선형분류기로 바꿀 수 있음

[Binary Softmax Classifier]

- σ(∑iwixi+b)를 한 클래스의 확률인 P(yi=1|xi;w)로 해석 가능
- Cross-entroy loss를 공식화할 수 있음
- 0.5를 중심으로 class 2개를 나눌 수 있음

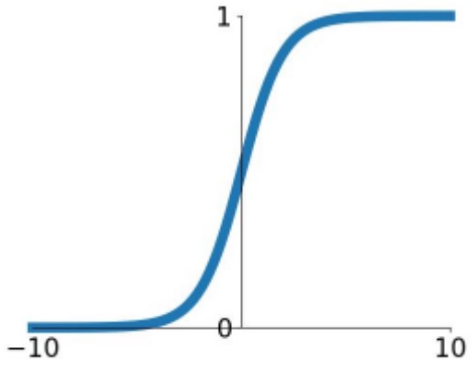
[Binary SVM Classifier]

- 뉴런의 출력에 max-margin hinge loss 사용

Regularization interpretation – 위와 같은 regularization loss는 점진적 망각으로 해석될 수 있음
--> 파라미터 업데이트 이후 마다 모든 weight가 0을 향해 줄어들기 때문

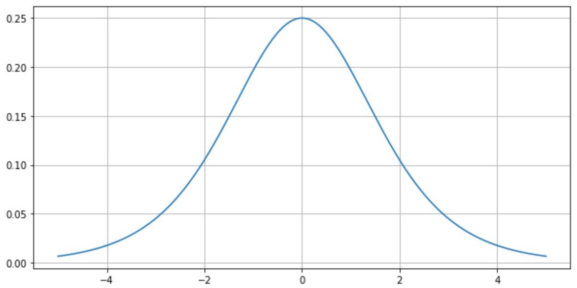
Modeling on neuron

Activation functions(sigmoid)



Sigmoid

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



Sigmoid함수 미분 그래프

[특징]

- 0고 1사이의 값으로 표현됨
- Firing rate를 적절히 해석

[단점]

- 뉴런의 값이 극단적으로 높거나 낮아지는 부분들의 기울기가 0으로 수렴 (sigmoids saturate and kill gradients)

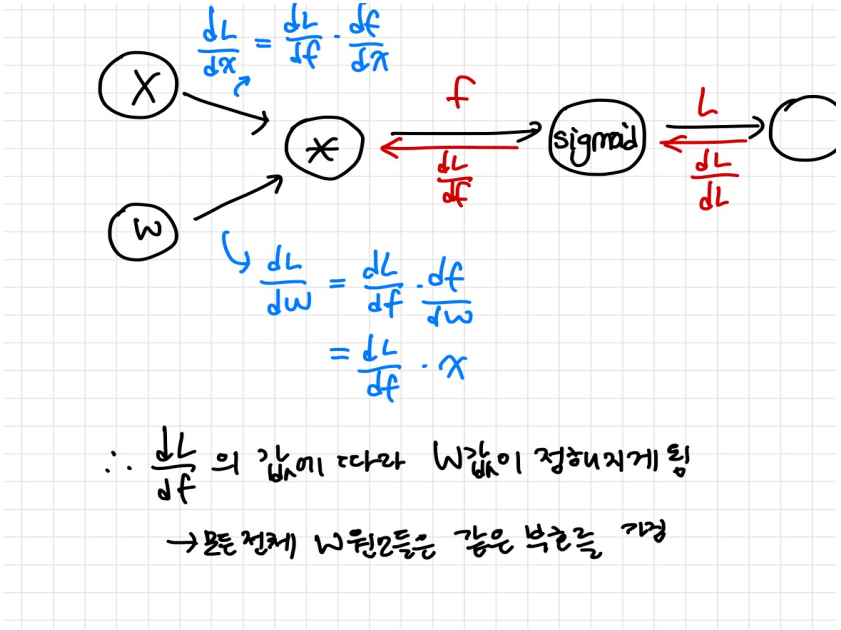
→ 기울기 0을 사용해 역전파를 진행할 경우 뉴런의 값이 비정상적으로 작아지고 네트워크가 흐르지 않을 것임

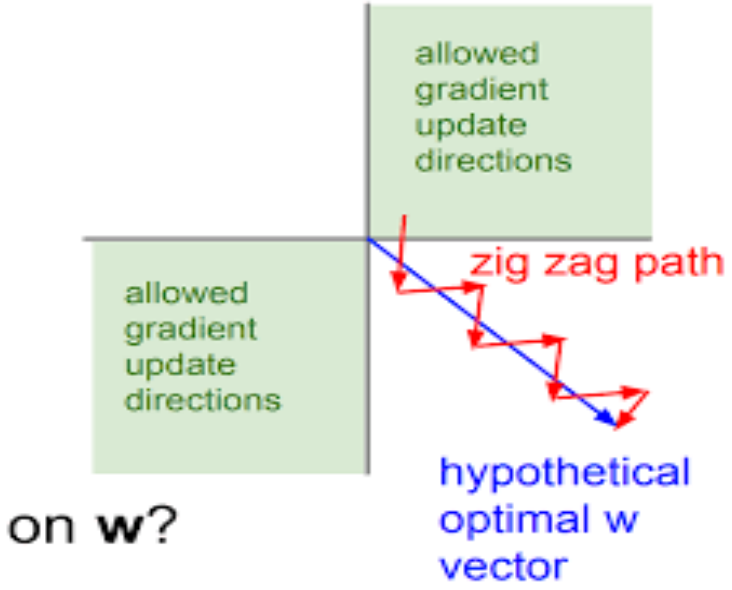
- Exp()함수 사용시 비용이 큼
- 출력값이 zero-centered 하지 않음

Modeling on neuron

Activation functions(sigmoid)

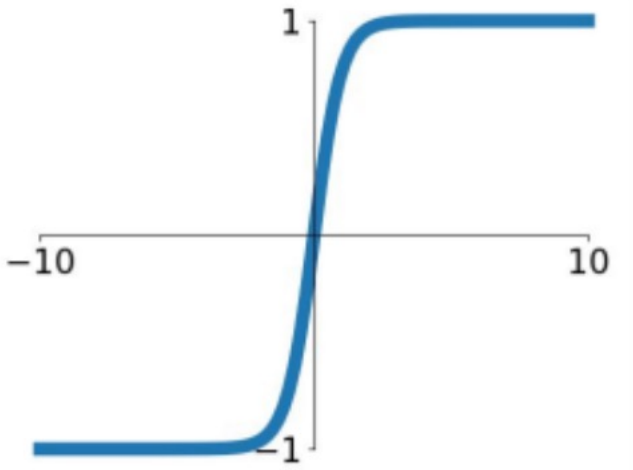
가정: 뉴런에 대한 input이 모두 양수인 경우(sigmoid함수의 결과 값은 항상 양수)
→ f=Wx+b, df/dw=x
→ 역전파를 거치면서 weight w의 gradient는 전부 같은 방향으로 업데이트 됨
→ Weight를 업데이트 할 때 zig-zagging dynamics를 가져올 수 있음
→ 비효율적으로 가중치가 업데이트 되는 것임





Modeling on neuron

Activation functions(tanh)



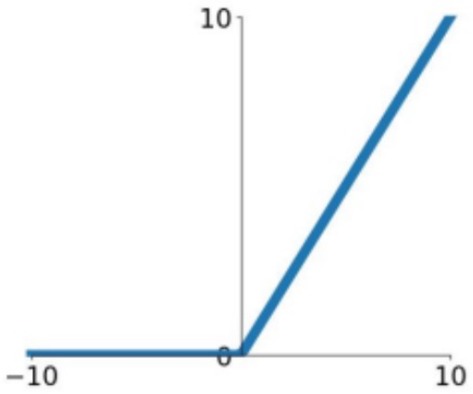
tanh(x)

[특징]

- Zero-centered 하지 못하다는 단점 보완
- -1에서 1값을 가짐
- Saturate되는 부분에서 gradient가 0이 됨

Modeling on neuron

Activation function(ReLU)

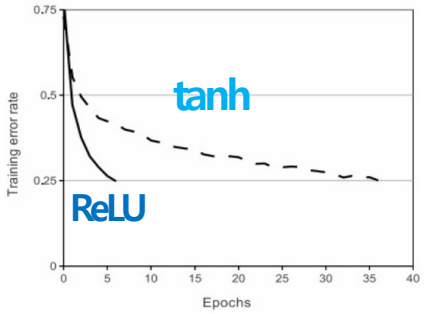


ReLU

$$f(x) = \max(0,x)$$

[특징]

- 양수 부분에서 saturate하지 않음(기울기 소실 문제 발생 x)
- 계산비용이 적음
- Sigmoid, tanh보다 SGD를 훨씬 빨리 수렴
- 생물학적 원리와 어느정도 상응



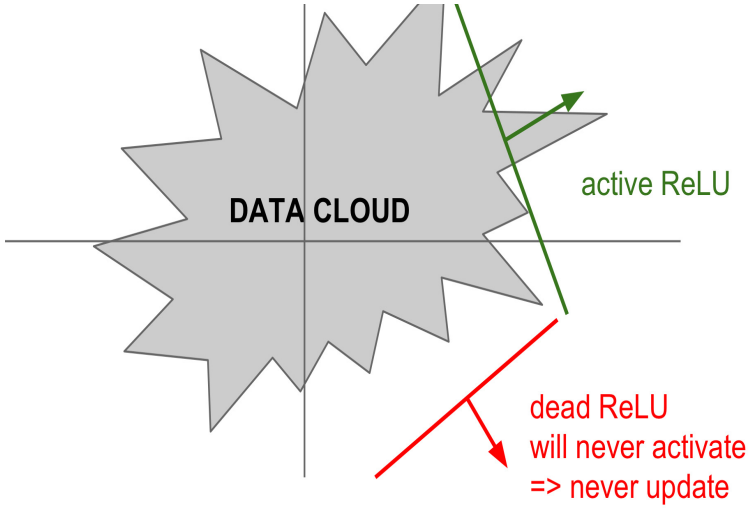
[단점]

- Zero-centered 해결 못함
- 여전히 음수 부분에서는 saturate함
- Dead ReLU

Modeling on neuron

Activation functions(ReLU)

- RELU가 data cloud로부터 지나치게 멀어져 있을 때 발생
- 뉴런이 활성화되지 않아 전혀 업데이트가 되지 않는 문제 발생



[발생 이유]

- 초기화를 잘못한 경우

→ 초기 가중치가 data cloud로 부터 멀리 떨어져 있으면 활성화되지 않아 업데이트가 되지 않음

- Learning rate를 높게 설정할 경우에 많으면 40%의 네트워크가 죽을 수 있음.

→ 학습속도 적절히 설정)

$$W_{ij}^{*} = W_{ij} - a\left(\frac{\partial E}{\partial W_{ij}}\right)$$

New Weight / Old Weight / Learning Rate / Derivative of Error with respect to Weight

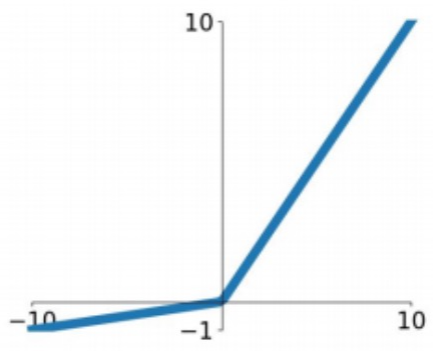
[해결]

- Positive bias

→ 0.01정도를 주어 초기화로 인한 Dead ReLU현상을 막아보려는 것

Modeling on neuron

Activation functions(Leaky ReLU)



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

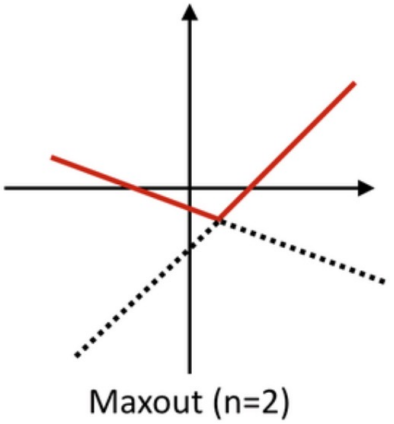
[특징]

- 기존의 ReLU 함수에 약간의 변형
- f(x)=1(x<0)(αx)+1(x>=0)(x)
- 음수영역의 saturate 현상을 보완

→뉴런이 죽지 않음

Modeling on neuron

Activation functions(Maxout)



- ReLU와 Leaky ReLU를 일반화한 버전
- Maxout - max(wT1x+b1,wT2x+b2) 를 수행
- ReLU의 모든 이점을 누리지만 dying ReLU라는 단점이 없음
- 하지만 모든 뉴런에 대해 파라미터 수를 두 배로 증가시켜, 파라미터 수를 크게 증가시킴

어떤 뉴런을 사용해야 되는지?

→ RELU 함수

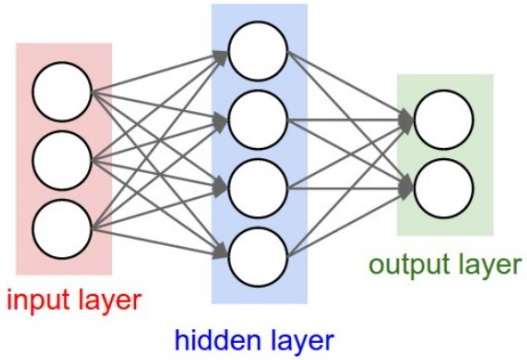
→ Learning rate와 네트워크 안에서 'dead'하는 비율을 잘 관찰해야함

→ Leaky ReLU, Maxout도 시도해보는 것도 좋음(sigmoid는 절대 사용x)

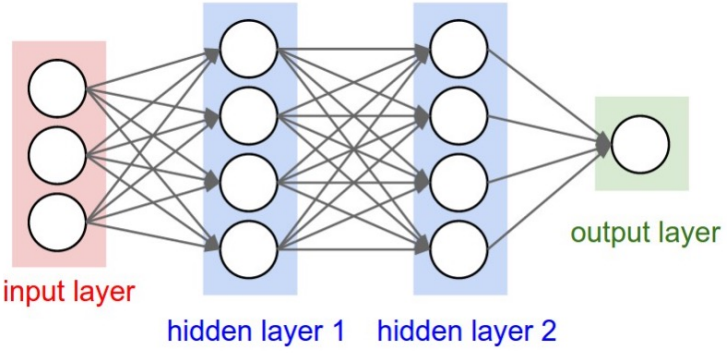
Neural Network architectures

Layer-wise organization

- Neural networks는 acyclic graph(비순환 그래프)에 연결된 뉴런의 집합체로 모델링
- 어떤 뉴런의 출력은 다른 뉴런의 입력으로 가능
- Layer 형태로 연결되어 있음
- Fully connected layer- 두 인접한 layer 사이의 뉴런이 전부 쌍으로 연결



2개의 layer



3개의 layer

Neural Network architectures

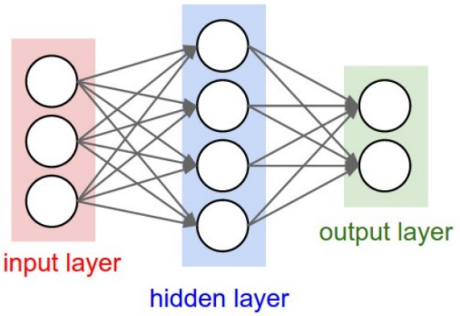
Layer-wise organization

[Output layer]

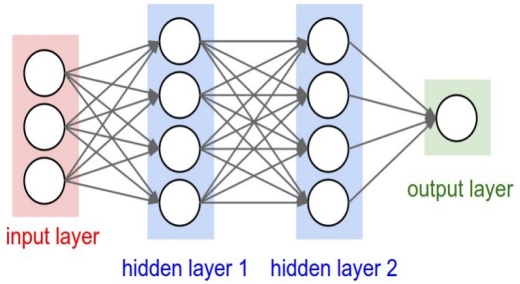
- 다른 모든 레이어와는 다르게, 출력 레이어에는 activation function이 없음
- 활성화 함수로써 선형의 항등함수를 가지는 것으로도 해석 가능
- 분류의 경우 클래스 점수를 표현 하기 위해 사용됨
- 회귀의 경우 실수 값을 가지는 것이 목표임

[Sizing neural networks]

- 크기 측정 기준: 뉴런의 수, 파라미터 수



Weight : [3x4]+[4x2]=20
Bias : 4+2=6
Total : 26개의 파라미터



Weight : [3x4]+[4x4]+[4x1]=32
Bias : 4+4+1=9
Total : 41개의 파라미터

Neural Network architectures

Example feed-forward computation

Input: [3x1] 벡터

W1:[4x3], b1:[4x1],
W2:[4x4]. b2:[4x1]
W3:[1x4]

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoi
d)
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
(4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

np.dot(W1,x)은 그 레이어의 모든 뉴런에 대한 activation을 계산함

마지막 layer는 활성화 함수를 거치지 않음

모든 데이터는 병렬적으로 구해짐

04

Neural Network architectures

Representational power

- Fully connected layer들로 이루어진 Neural Networks는 weight를 통해 파라미터화된 함수들을 정의한다고 볼 수 있음
- 어떠한 연속함수 f(x)가 주어지더라도 ,적어도 하나의 hidden layer를 가지고 비선형성을 포함하는 Neural Networks는 일반적인 근사치인 g(x)로 표현 가능, $|f(x)-g(x)|<\epsilon$
  → Neural Network는 어떤 연속함수라도 근사적으로 표현 가능

만약 하나의 hidden layer만으로도 함수를 근사하는데 충분하다면, 왜 더 많고 깊은 레이어를 쌓으려고 하는지?

→ Two layer Neural Network가 수학적으로 일반적인 근사치를 만족하지만 실전에서는 비교적 부정확하고 활용하기 어렵기 때문

→ 여러 hidden layer를 가진 더 깊은 네트워크는 단일 레이어 네트워크와 표현력이 동일하지만, 더 잘 동작한다는 사실이 있음

Neural Network architectures

Representational power

- 실전에서는 3 layer neural network가 2 layer net을 능가하는 경우는 많음,
- 하지만 더 깊은 layer인 3,4,5 가 더 좋은 성능을 발휘하는 경우는 거의 없음
- 이는 Convolutional Networks와 극명한 대조를 이루는 점 (깊이가 중요)
- 이미지가 계층 구조를 포함하기 때문

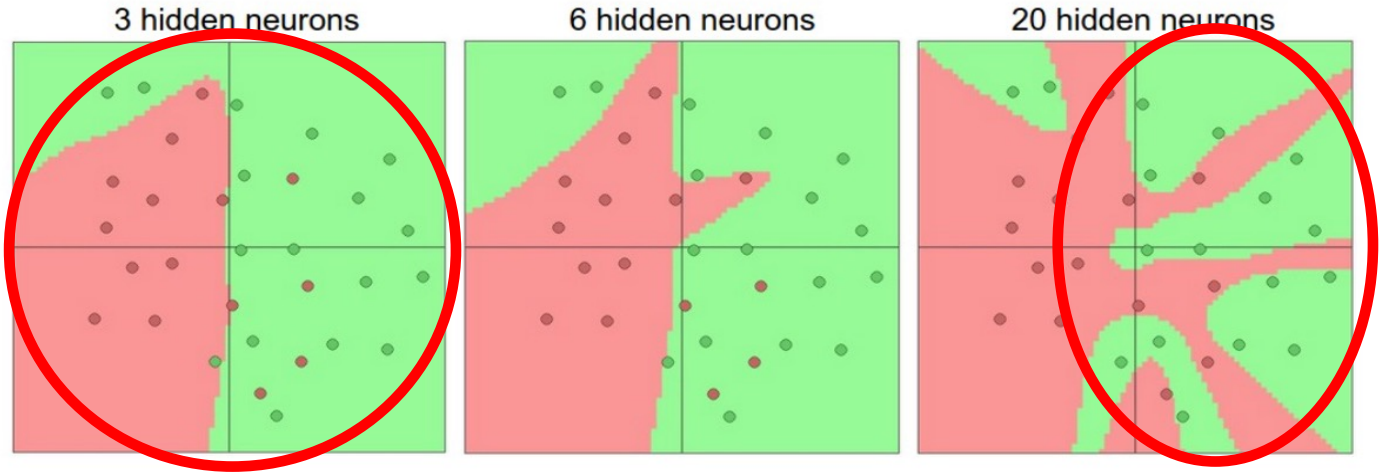
→ 이미지와 같은 데이터 도메인 에서는 여러 개의 layer를 처리하는 것이 직관적으로 타당함

Neural Network architectures

Setting number of layers and their sizes

Hidden layer의 크기 or 개수 ?

- Layer의 크기와 개수를 늘림--> 네트워크 용량(capacity)증가
- 뉴런들이 다양한 함수를 표현하기 위해 협동하기 때문에 표현 가능한 함수의 공간이 커짐
- Overfitting가능성 →모델의 용량이 너무 높아 예상되는 일반적인 관계를 넘어 데이터의 noise까지 맞춰버릴 때 발생



공간을 여러 개로 해체

2개의 공간으로 분할—>몇몇 점을 outlier로 처리 → generalization

Neural Network architectures

Setting number of layers and their sizes

[Overfitting 방지]

- L2 regularization
- Dropout
- Input noise

***overfitting을 제어하기 위해 뉴런의 수를 조절하는 것보단 다른 방법을 사용해주는 것이 좋음**

- 작은 네트워크의 loss function의 극솟값들은 비교적 적고, 여러 극솟값 중 하나의 값으로 수렴하기 쉬움

→ 작은 네트워크를 학습시킬 때 최종 loss는 상당히 큰 분산을 보임

→ 운이 좋은 경우 좋은 위치에 수렴, but 극솟값에 갇히는 좋지 않은 경우 발생 가능성이 있음

- 큰 네트워크는 훨씬 더 많은 극솟값을 가지고 실제 Loss 측면에서 더 좋은 극솟값을 보임

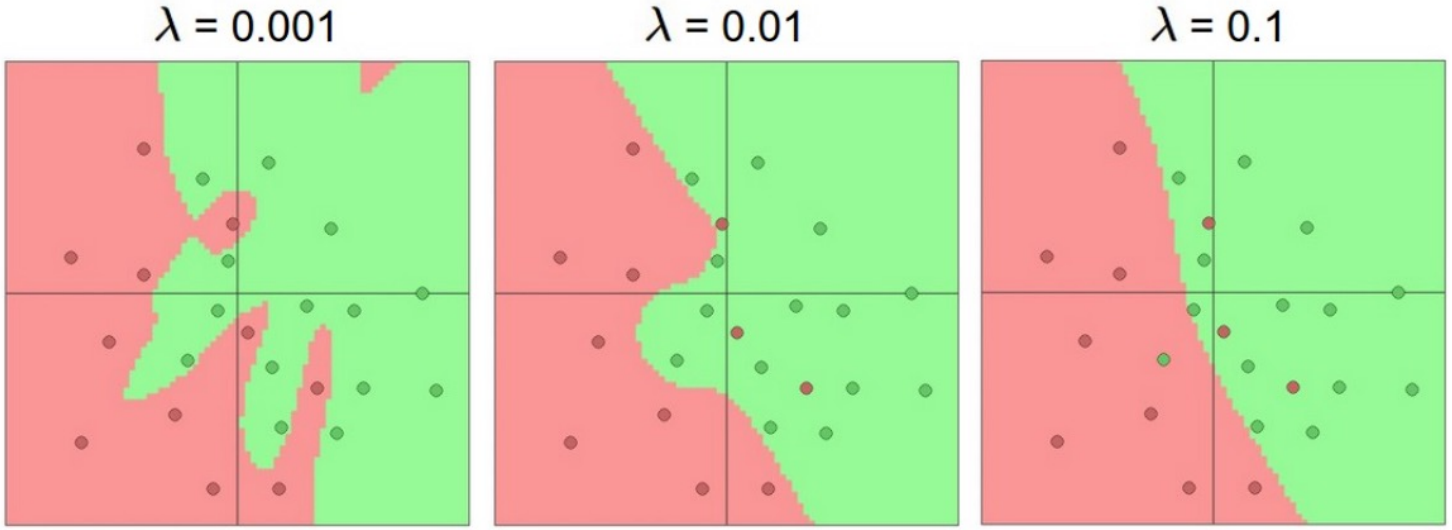
→ 학습시킬 때 다양한 해로 수렴하겠지만, 최종 loss 분산은 훨씬 작을 것이다.

→ 모든 해가 비슷한 정도로 양호할 것임

Neural Network architectures

Setting number of layers and their sizes

동일한 20개의 hidden neurons



Regularization strength이 높을수록 결정 영역이 더 부드럽게 만들어짐

→ Overfitting때문에 작은 네트워크를 사용하는 것은 바람직하지 않음

→ 연산량이 감당할 수 있을 만큼 최대한 큰 네트워크를 사용하고,

오버피팅을 제어하기 위해서는 다른 regularization기법들을 사용함