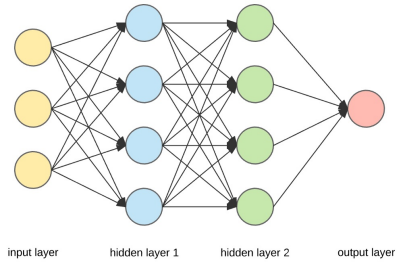
13 Neural Network

< 뉴런 덩어리 >

[input → hidden layer → output] 구조의 함수 근사 모델



13.1 Overview of NN Architectures

- image/text 분류
- voice 인식
- 예측 모델링 ....

* Architecture :

$$x \xrightarrow{W^{(1)},\sigma} h_1 \xrightarrow{W^{(2)},\sigma} h_2 \xrightarrow{\cdots W^{(L)}} \hat{y}$$

(수학적...)

- 각 layer : $h^{(l)} = \sigma(W^{(l)} h^{(l-1)} + b^{(l)})$
- σ : non-linear function
- $W^{(l)}$: weight matrix, parameter
- $b^{(l)}$: bias vector

입력공간 $\mathbb{R}^d$을 출력공간 $\mathbb{R}^K$에 mapping하는 가중치 벡터 θ로 매개변수화된 함수. 입력을 층별로 출력으로 변환.

층 $l+1$ 의 값은 이전 층 l 의 값의 함수

↳ linear / Nonlinear

↝ layer를 여러개 쌓으면 복잡한 nonlinear function 근사가능하게됨

- 학습은 loss function ↓ : backpropagation + GD

- Activation function

e.g)

sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$
tanh	$\frac{e^x - e^{-x}}{e^x + e^{-x}}$
ReLU	$\max(0, x)$
Leaky ReLU	$\max(0.01x, x)$

구조 변화

- MLP (Fully connected) : 모든 node 연결
- CNN : spatial structure (공간구조) 유지 (image)
- RNN : Recurrent; 순서 정보 (시계열, text)
- ResNet, Transformer 더 깊은 attention 도입...

MLP (Fully connected) / CNN (공간

때문이라...

CNN : Convolutional NN

image data에 주로 사용.

∵ 이미지의 spatial structure (공간구조) 활용 가능

Fully connected layer 사용

- 32 x 32 x 3 이미지 → 3072개 node
- 다음 layer가 1000개 node 가지면? → 3072000개

↓ parameter 너무 많

$$Z_{i,j} = \sum X_{i+m, j+n} \cdot K_{m,n}$$

X : 입력 image
K : 필터
Z : 출력 feature map

작은 kernel이 입력 이미지를 일정간격으로 슬라이딩하며 local feature를 추출

* RNN : Recurrent NN

순서가 중요한 data (시계열/text/audio)

구조) 이전의 hidden state를 현재로 전달.

$h_t = \sigma(W_{hh} h_{t-1} + W_{xh} x_t + b)$ ...

output) $y_t = W_{hy} h_t$

hidden state가 순서를 따라 time step마다 반복

Prob) 긴 문장 등. long-term dependency 기억 어려움

→ vanishing/exploding gradient

sol) LSTM, GRU.....

NN : 고정된 vector을 입력하고 (모든) node를 완전 연결 & 공간/순서 무시

13.1 Overview of NN Architectures

- Linear Layer (Affine Transformation + Activation)

$z^{(\ell+1)} = \phi(A_\ell z^{(\ell)} + b_\ell)$. A^ℓ: weights matrix

b^ℓ: bias vector

eg) 2-layer NN:

$f_\theta(x) = \sigma(A_1 \cdot \mathrm{ReLU}(A_0 x + b_0))$

- Convolutional layer:

is organized in a tensor

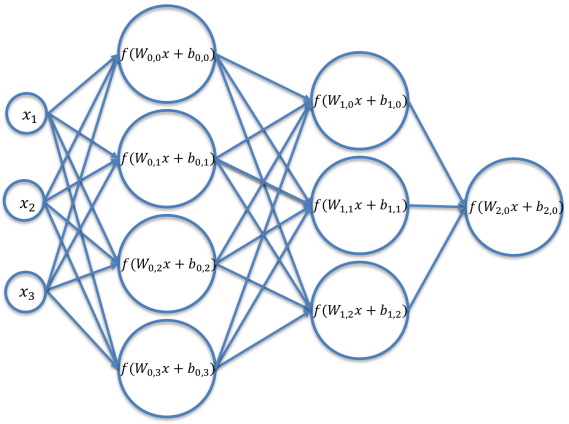
image dimension

$w_\ell \times h_\ell \times c_\ell \rightsquigarrow w_{\ell+1} \times h_{\ell+1} \times c_{\ell+1}$

width height channel

- Attention layer:

act on a sequence of n data vectors of dimension d.



13.2 Optimization of Neural Network

loss:

$$\hat{R}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \mathrm{loss}(y_i, f_\theta(x_i))$$

- Classification이면 cross-entropy,
  regression이면 MSE 사용

목표 = $\hat{R}(\theta)$ 로스를 줄이자.

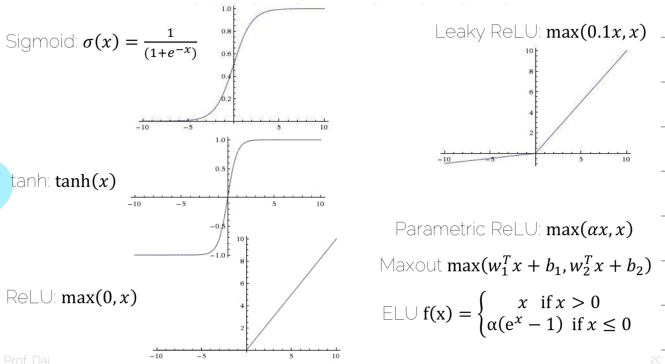
* Problem:

Neural Network의 최적화는 non-linear & non-convex 라서 여러개의 local minima가 존재 가능

→ 해결방법

Gradient Descent : $\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta \hat{R}(\theta^{(t)})$

cf) 보통 SGD 또는 Adam 등 변형된 방법 사용



* 실전에서는 convex가 아니지만 NN구조와 초기화가 잘 설계되면 global minima에 수렴 가능 (empirical risk가 0이 되는 경우)

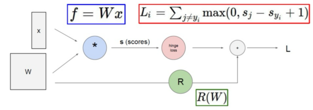
13.3 Automatic Differentiation & Backpropagation

$\nabla_\theta \hat{R}(\theta)$ 를 최소화할때 (minimizing empirical risk) 4가지 방법 있음

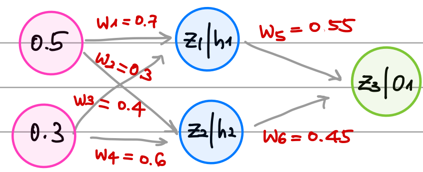
방법	설명	한계
by hand	수학적 직접유도 (기울기 직접계산 등)	비현실적
Symbolic diff	Maple, Mathematica 같은 도구 패키지.	수식 커짐
Numberic diff	미분 $\frac{f(x+h)-f(x-h)}{2h}$ 으로 계산	부정확
Auto diff	연산 그래프 기반 (PyTorch, Tensorflow, JAX)	정확·효율

2.2 Backpropagation

- Calculate gradients numerically if function becomes too complex for analytical solutions
- Analytically solutions would at least partly be possible, but can't e.g. be parallelized
- Possible problems and solution:
  - Solution for multiple outputs in a compute node: take average
  - Solution for loops: No loops in computational graphs
  - Values of forward pass required for backward pass: Cache results of forward pass to reach faster runtime in backward pass
- Computational graph for Linear activation with hinge loss and regularization:



* Backpropagation



1) feedforward

$z_1 = x_1 w_1 + x_2 w_2 = 0.5 \cdot 0.7 + 0.3 \cdot 0.4 = 0.47$

$h_1 = \text{sigmoid}(z_1) = \frac{1}{1+e^{-0.47}} = 0.615$

$z_2 = x_1 w_2 + x_2 w_4 = 0.5 \cdot 0.3 + 0.3 \cdot 0.6 = 0.33$

$h_2 = \text{sigmoid}(z_2) = \frac{1}{1+e^{-0.33}} = 0.582$

$z_3 = h_1 \cdot w_5 + h_2 \cdot w_6 = 0.615 \cdot 0.55 + 0.582 \cdot 0.45 = 0.6$

$O_1 = \text{sigmoid}(z_3) = \frac{1}{1+e^{-0.6}} = 0.645$

$S(x) = \frac{1}{1+e^{-x}} \xrightarrow{df\ldots} \frac{e^{-x}}{(1+e^{-x})^2} \approx S(x)\left(1 - S(x)\right)$

2) $L = \frac{1}{n}\sum_i^n (y_i - \hat{y}_i)^2 \Rightarrow n=1$: 가정 $L = \frac{1}{1}(1 - 0.645)^2 = 0.126$

3) Chain rule 로 계산

$L = \frac{1}{n}\sum_i^n (y_i - \hat{y}_i)^2 \rightsquigarrow n=1: \; L = (y - O_1)^2$

$\frac{\partial L}{\partial O_1} = -2(y - O_1) = -2(1 - 0.645) = -0.71$

보통 Sigmoid 의 형태 : $O_1 = \text{sigmoid}(z_3)$

$\frac{\partial O_1}{\partial z_3} = S(z_3)(1 - S(z_3)) = O_1(1 - O_1) = 0.229$

$\frac{\partial L}{\partial w_5} = \frac{\partial z_3}{\partial w_5} \cdot \frac{\partial O_1}{\partial z_3} \cdot \frac{\partial L}{\partial O_1}$ (우세 역 나)

$z_3 = h_1 w_5 + h_2 \cdot w_6$

$\frac{\partial z_3}{\partial w_5} = h_1 = 0.615$

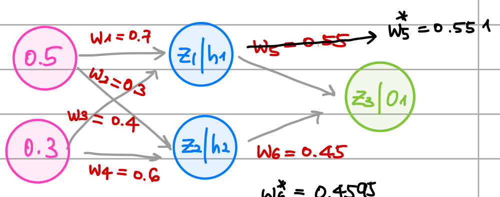
$\rightarrow \frac{\partial L}{\partial w_5} = \frac{\partial L}{\partial O_1} \frac{\partial O_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5} = -0.71 \cdot 0.229 \cdot 0.615 = -0.01$

SGD: $\theta^{(t+1)} = \theta^t - \eta \cdot \nabla \hat{R}(\theta)$ ($\nabla \hat{R}(\theta)$ // $\frac{\partial L}{\partial w_5}$)

$w^* = w^5 - \eta \cdot \frac{\partial L}{\partial w_5}$

$= 0.55 - 0.1 \cdot (-0.01) = 0.551$ (0.1: 임의가정)

$w_1^* = 0.7010$
$w_2^* = 0.3009$
$w_3^* = 0.4011$
$w_4^* = 0.6005$



* NN의 오차가 줄었는지 확인.

$z_1 = x_1 w_1 + x_2 w_3 = 0.5 \cdot 0.7010 + 0.3 \cdot 0.4011 = 0.4708$

$h_1 = \frac{1}{1+e^{-0.4708}} = 0.6156$

$z_2 = x_1 w_2 + x_2 \cdot w_4 = 0.5 \cdot 0.3009 + 0.3 \cdot 0.6005 = 0.3306$

$h_2 = \frac{1}{1+e^{-0.3306}} = 0.5819$

$z_3 = h_1 \cdot w_5 + h_2 \cdot w_6 = 0.6156 \cdot 0.551 + 0.5819 \cdot 0.4595 = 0.6066$

$O_1 = \frac{1}{1+e^{-0.6066}} = 0.6472$

$L = \frac{1}{n}\sum_i^n (y_i - \hat{y}_i)^2 \rightsquigarrow n=1: \; L = (y - O_1)^2$

$= (1 - 0.6472)^2 = 0.1245$

13.4 Vanishing Gradients

↳ 아주 깊은 Network 에서 gradient 가 지나치게 작거나 커짐 → 학습이 안 됨 < Vanishing problem / Exploding problem

- Example : $loss(\theta_1, \dots \theta_L) = (y - x \cdot \theta_1 \cdot \theta_2 \cdot \theta_3 \cdots \theta_L)^2$

  L이 커질수록 (깊어질수록) $y - x\prod_i \theta_i$ 형태 → 0 또는 무한대

- Solution – gradient의 분포를 조절해서 학습을 안정화 시키자

1) Normalization layers

- Batch Norm : 각 배치마다 mean = 0, var = 1 Regularization (정규화)
- LayerNorm : Layer 단위로 정규화
- Channel Normalization : 각 채널마다 독립적으로 정규화

$$CN(z) = \frac{z - mean(z)}{\sqrt{var(z) + \varepsilon}}\gamma + \beta$$

γ, β 는 trainable 하거나 고정된 채널별 scale or shift parameter

ε 는 numerical stability 사용

- If $CN(\theta) = \frac{\theta}{|\theta|}$ 이면 → $loss(\theta_1, \cdots \theta_L) = \left(y - x \cdot \theta_1 \frac{\theta_2}{|\theta_1|} \cdots \frac{\theta_L}{|\theta_L|}\right)^2$

  벡터 θ를 norm 1 로 정규화

  γ 는 $1 \cdots 1$

- Normalization makes the deeper layers obsolete

  $\theta_2 \sim \theta_L$ 까지를 규제해버자

2) Residual Connections (Skip Connection)

각 layer들을 identity map 에 더 근사시키자. = 입력을 다음층에도 직접 더하자

$z_{l+1} = ReLU(A_l z_l + b_l)$ 에서 $\boxed{z_{l+1} = z_l + ReLU(A_l z_l + b_l)}$

- ResNet 에서 사용, 이미지 복원의 U-Net 구조에서도 사용

$$loss(\theta_1, \theta_2, \cdots, \theta_L) = \left(y - x\theta_1(1+\theta_2)\cdots(1+\theta_L)\right)^2$$

$\theta_2, \cdots, \theta_L$ 이 1 에 비해 $(1+\theta_i)$ term 에서 작기때문에 gradient 가 너무 작거나 너무 크지 않으며 vanishing problem 해소

3) Good initialization

적절한 초기화(initialization)만으로도 vanishing problem 해결가능

e.g) 우리 loss( ) 에서 $\theta_2, \cdots, \theta_L$ 을 1근방값으로 초기화하면 residual 구조와 유사 ∴ 해소