Lecture 3.
Model Optimization

Feedbacks,

열정적인 수업 감사드립니다.
코딩수업 바뀐방식이 더 좋은데 조금만더 천천히 진행해주셨으면 좋겠어요
Regularization이 이해가 잘 안되었습니다 순간적으로 집중을 못해서 놓쳤네요
수업 로드가 어제보다 적어 적당했습니다.
1회차보다 훨씬 어려웠지만 훨씬 좋았습니다.
good
1회차 수업에 비해 강의 난이도가 갑자기 오르긴 했지만, 아직까지는 괜찮은 듯 함.
코딩 실습에서 기본 제공코딩 또한 어떠한 기능을 하는 지 설명해주었으면 좋겠습니다.

추가적인 upload 하겠습니다.

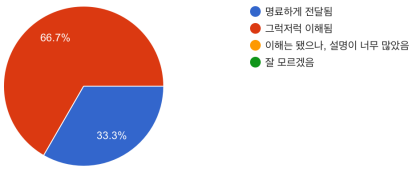
아마 2회차가 가장 어려울 것 같습니다....
수학적 내용은 천천히 이해해도 됩니다!

Loss에 대한 설명이 잘 전달되었나요?

응답 9개

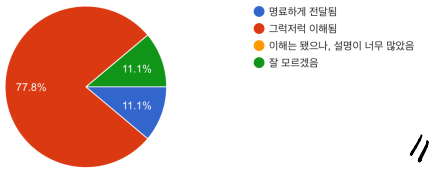
Loss Function에 대한 설명이 잘 전달되었나요?

응답 9개



Regularization에 대한 설명이 잘 전달되었나요?

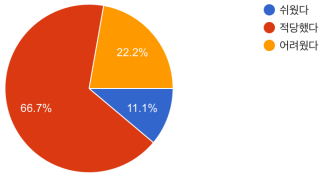
응답 9개



- Github 읽기 자료.
- Regularization & Validation 보충설명
- Questions via, Github Issue / Kakao Talk ...

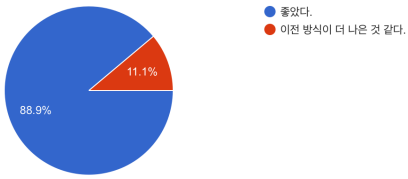
Linear Classifier 코딩실습의 난이도는 어땠나요?

응답 9개



코딩 실습 방식이 바뀐 것에 대해 어떻게 생각하시나요?

응답 9개



유지하되, 조금 천천히.

+ Github Repo Star

Review

Questions

1. **Train이 잘 되었는지 판단할 수치적 척도 필요**

   : define a **Loss Function** that quantifies our unhappiness with the scores across the training data

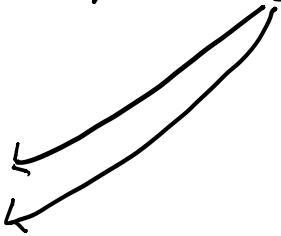
2. **Parameter를 update하는 algorithm 필요**

   : come up with a way of efficiently finding the parameters that minimize the **Loss Function**

Review

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda R(W)$$

1. Loss
2. **Loss Function**
   - Multiclass SVM Loss
   - Cross-Entropy Loss

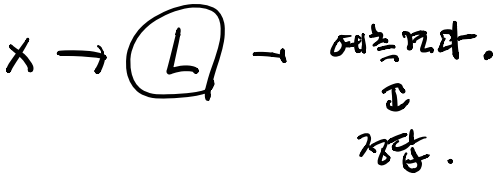


$x \rightarrow s$

3. **Regularization**
   - Why it is needed : to discourage large weight matrix
   - Overfitting

$L = 0 \quad |w'| \quad kw'$

Review



How is a Model Optimized / Updated?

Loss.

1. Training Set의 Data들을 Linear Classifier에 통과시켜서, 그 결과들을 정답과 비교
2. 1에서의 결과를 바탕으로 Parameter의 값들을 Update
3. 다시 1로

Loss.

Review

Questions

• **Train이 잘 되었는지 판단할 수치적 척도 필요**

: define a **Loss Function** that quantifies our unhappiness with the scores across the training data

• **Parameter를 update하는 algorithm 필요**

: come up with a way of efficiently finding the parameters that minimize the **Loss Function**

Optimization and Backpropagation

Loss Function은 Scalar Function

Thus, Loss Function의 input W, b에 대해

Gradient를 구해 주어 W, b에서 빼면,

Loss가 줄어드는 방향으로 학습하지 않을까?

21

Today's Contents

1. **Optimization**

2. **Backpropagation**

Optimization : Idea

(Currently) Incorrect Model → Correct Model 을 위해,

Parameter W와 b를 Update해줘야 함.

How…?

Optimization : Idea

① W = 0.5, b = 0.3 → L = 10

② W = 0.3, b = 0.1 → L = 5

IDEA 1. Random Search

1. Randomly set W and b
2. Select (W, b) that gives the minimum loss in training

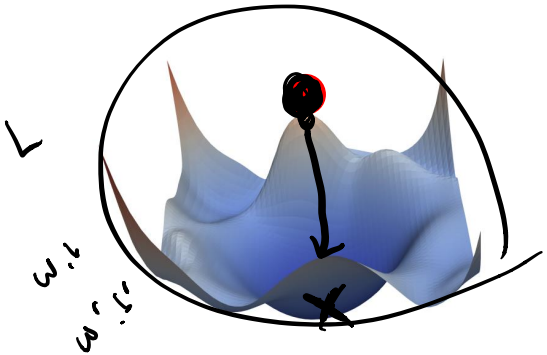
Obviously very poor ☹

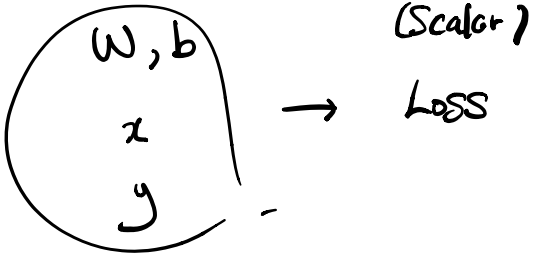
Optimization : Idea

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(Wx_i + b, y_i) + \lambda R(W)$$

IDEA 2. Gradient Descent

Loss : W, b, x, y (vector & matrix) → L (scalar)

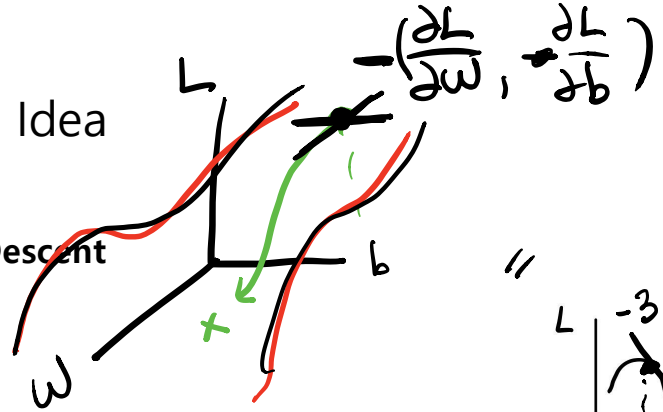




Loss의 Global Minimum으로 가려면 어떻게 해야할까?

Optimization : Idea

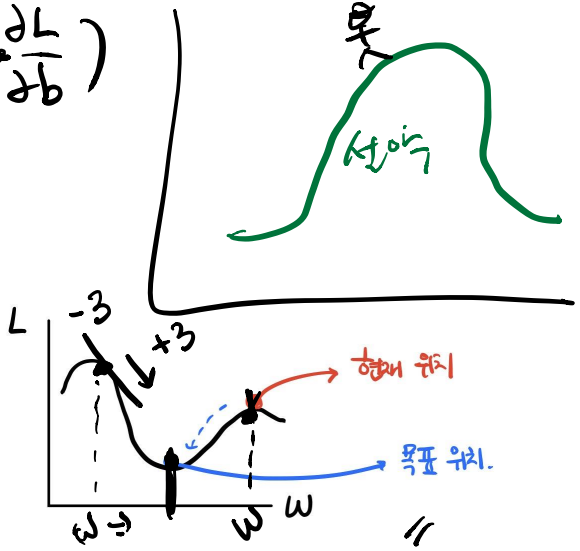
IDEA 2. Gradient Descent



$-\left(\frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}\right)$

W, b, x, y, L이 모두 Scalar라고 Simplify해 생각

$$W - \frac{\partial L}{\partial W} \times k$$

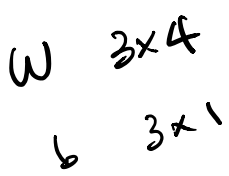


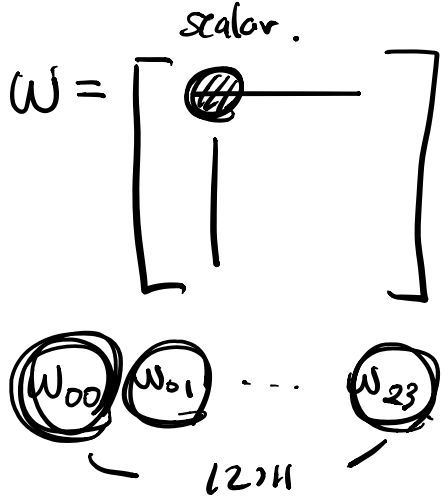
- 에서 • 로 가는 방법?
- 에서의 L의 기울기 방향으로 거쳐 됨.

i.e, $W - \frac{dL}{dW} \times k$

(k : some constant).

Optimization : Idea





IDEA 2. Gradient Descent

다시 Vector / Matrix일 때를 생각해 보면,

다차원 공간상의 L의 Minimum으로 가도록 W, b를 Update하려면,

We should calculate the GRADIENT ∇L_w

$$\frac{\partial L}{\partial W} = \begin{bmatrix} \frac{\partial L}{\partial w_{00}} & \frac{\partial L}{\partial w_{01}} & \cdots \end{bmatrix}$$

3x4

$$b = \begin{bmatrix} \\ \\ \\ \end{bmatrix} \quad b_0 \; b_1 \; b_3$$

3개

Optimization : Idea

IDEA 2. Gradient Descent

We can use vector and matrix notation to rewrite things a bit. Define the **gradient** of a scalar-valued function $f: X \subseteq \mathbf{R}^n \to \mathbf{R}$ to be the *vector*

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right).$$

Consequently,

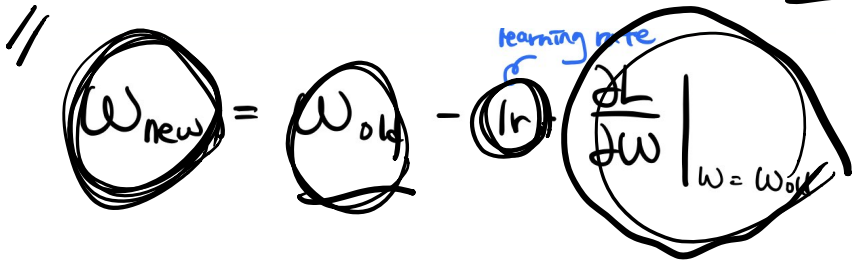
$$\nabla f(\mathbf{a}) = (f_{x_1}(\mathbf{a}), f_{x_2}(\mathbf{a}), \ldots, f_{x_n}(\mathbf{a})).$$

Optimization : Idea

W_{new} ← update 된 param.

IDEA 2. Gradient Descent

W_{old} ← 현재 param.



"lr"

Hyperparam.

"$\lambda \cdot R(W)$"

L1, L2

Optimization : Calculating Gradient

Gradient Descent

learning rate

$$W_{new} = W_{old} - lr \left(\frac{\partial L}{\partial W} \right)_{W = W_{old}}$$

$$b_{new} = b_{old} - lr \left(\frac{\partial L}{\partial b} \right)_{b = b_{old}}$$

then, how do we calculate the Gradient?

Optimization : Calculating "Gradient"

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$h = 0.000001$

IDEA 1. Numeric $\frac{\partial L}{\partial W} = \begin{bmatrix} -2.5 \\ \end{bmatrix}$

$$\frac{\partial L}{\partial w_{0,0}} = \lim_{h \to 0} \frac{L(w_{0,0}+h) - L(w_{0,0})}{h}$$

h에 10^{-4} 같은 작은 수 대입. $h=0.0001$

미분의 정의 활용

$$\frac{\partial L}{\partial w_{0,0}} = \lim_{h \to 0} \frac{L(w_{0,0}+h) - L(w_{0,0})}{h}$$

$h = 10^{-5}$

ex) $w_{0,0} = 0.34 \rightarrow L(w_{0,0}) = 1.25347$

$w_{0,0}+h = 0.3401 \rightarrow L(w_{0,0}+h) = 1.25322$

$$\frac{\partial L}{\partial w_{0,0}} \approx \frac{1.25322 - 1.25347}{0.0001} = -2.5$$

Optimization : Calculating Gradient

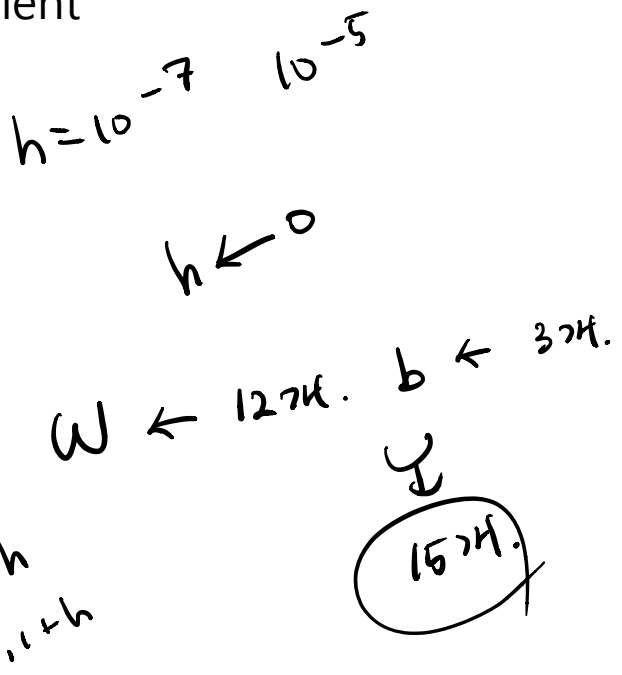
IDEA 1. Numeric

poor because,

1. 근사값만 얻을 수 있음

2. Time complexity

"thus, almost never used in practice"



Optimization : Calculating Gradient

IDEA 2. Analytic Gradient.

도함수를 직접 구해서 값을 대입해 미분값 구하기

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f'(x) \leftarrow x$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k \sum_l W_{kl}^2$$

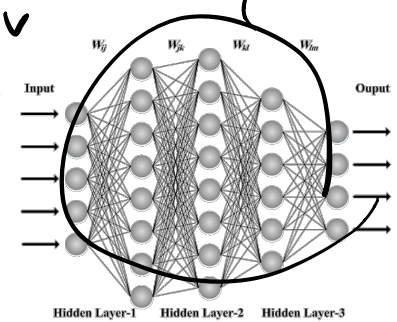
$$\sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

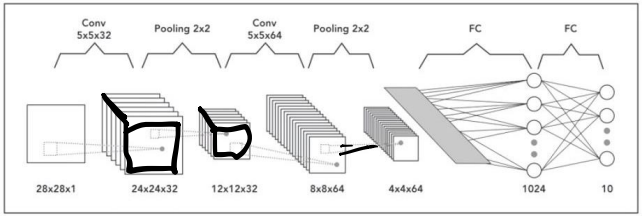
Optimization : Calculating Gradient

IDEA 2. Analytic

지금은 simple linear classifier만 다루고 있지만,

앞으로 다룰 model은





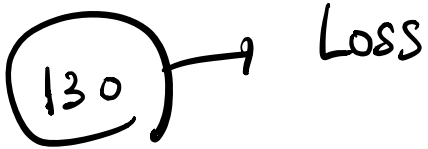
then, how should we derive the gradient?

Optimization : Calculating Gradient

Is Numeric Gradient **COMPLETELY** Useless?

No! Used for **Gradient Check**

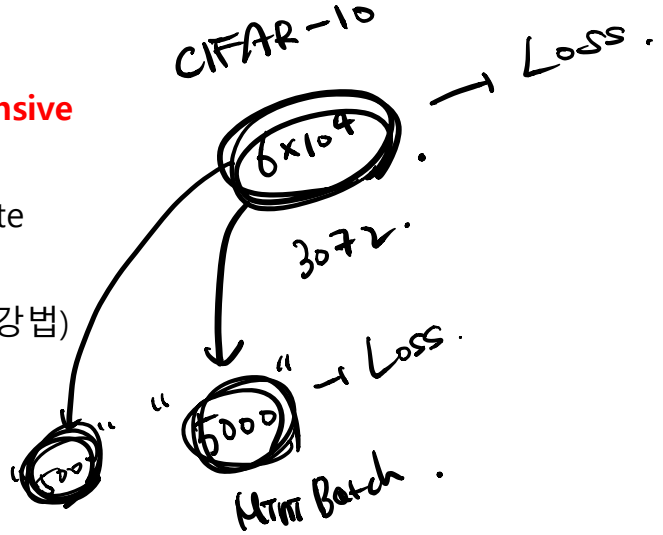
Optimization : Calculating Gradient



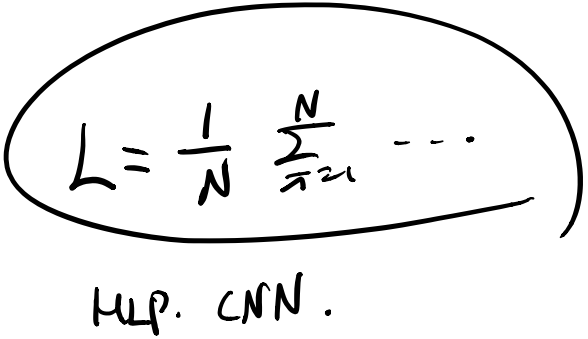
but, all dataset에 대해 Loss 계산...? Still Too Expensive

thus, 일부 Dataset(Batch)에 대해 Loss 계산 후 Update

= **SGD** (Stochastic Gradient Descent, 확률적 경사하강법)



"Backpropagation : Idea"



Analytic.

We compute the gradient via, "**Backpropagation**"

Backpropagation : Idea

$f(x) = e^{\cos x}$

$g(x) = e^{x}$, $h(x) = \cos x$ → $f = g(h(x))$

$L = h(f(g(\cdots)))$

1. 복잡한 함수는 간단한 함수/연산들의 합성함수로 나타낼 수 있다.
2. 간단한 함수들에 Chain Rule을 적용하여, 복잡한 함수의 미분계수를 구한다.

$$\frac{df}{dx} = h'(x) \times g'(h(x)) = \frac{df}{dh} \times \frac{dh}{dx} = -\sin x \times e^{x}$$

$$\frac{\partial L}{\partial w} = \frac{\partial h}{\partial \cdot} \times \frac{\partial \cdot}{\partial w}$$

Backpropagation : Computational Graph

복잡한 함수를 간단한 함수/연산들의 합성으로 나타낼 때,

"**Computational Graph**"를 활용 (Node : Operator, Leaf : Operand)

Backpropagation : Computational Graph

f(x, y, z) = (x + y) * z → (1, 2, 3)

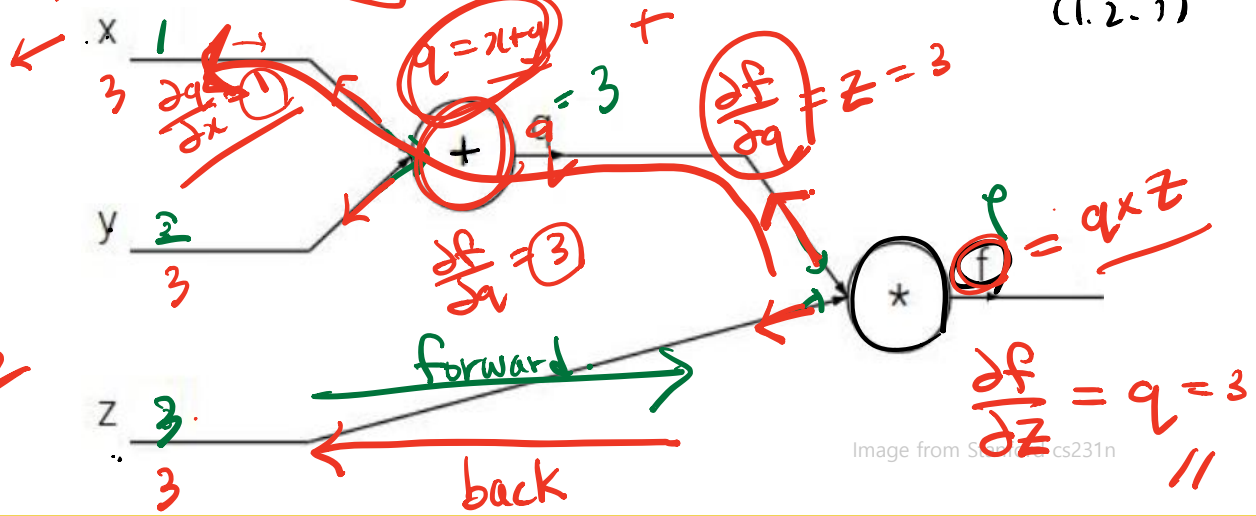
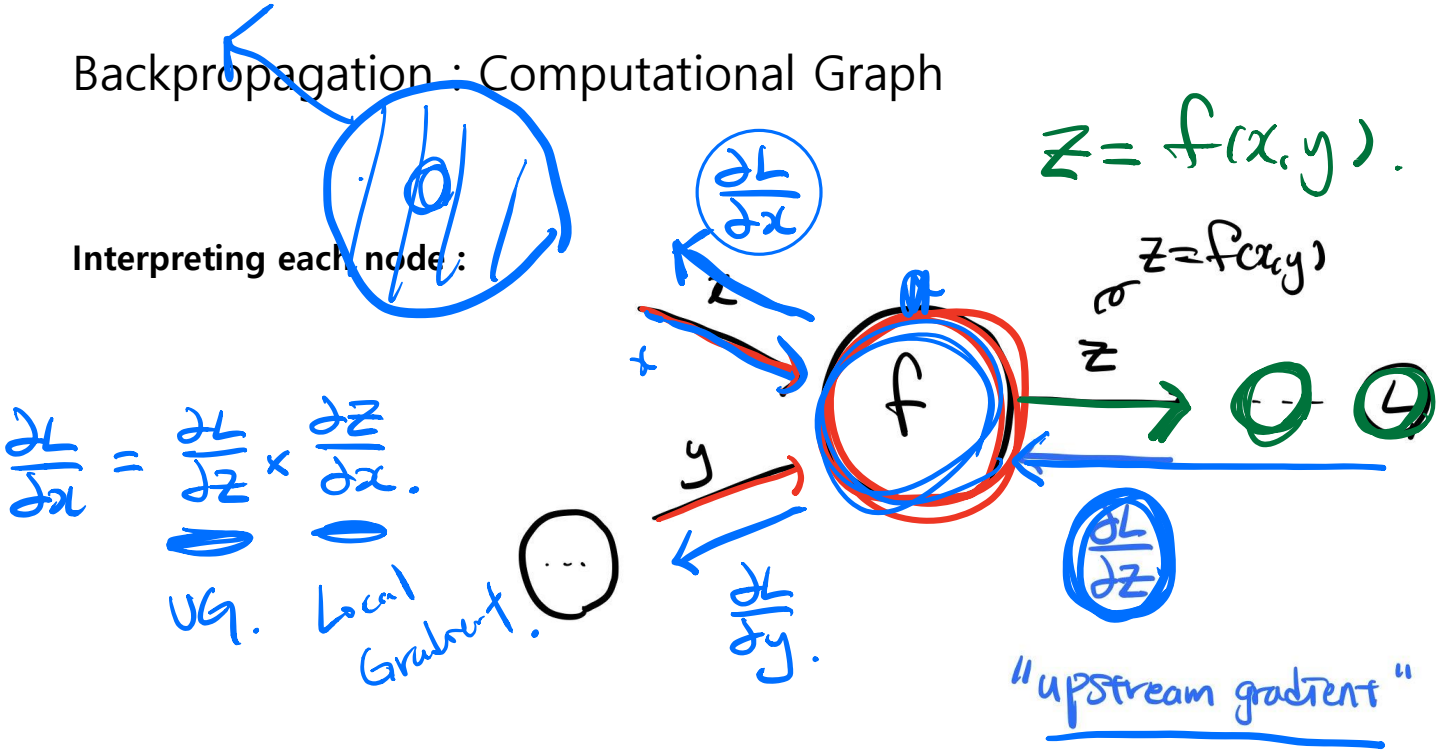


Image from Stanford cs231n

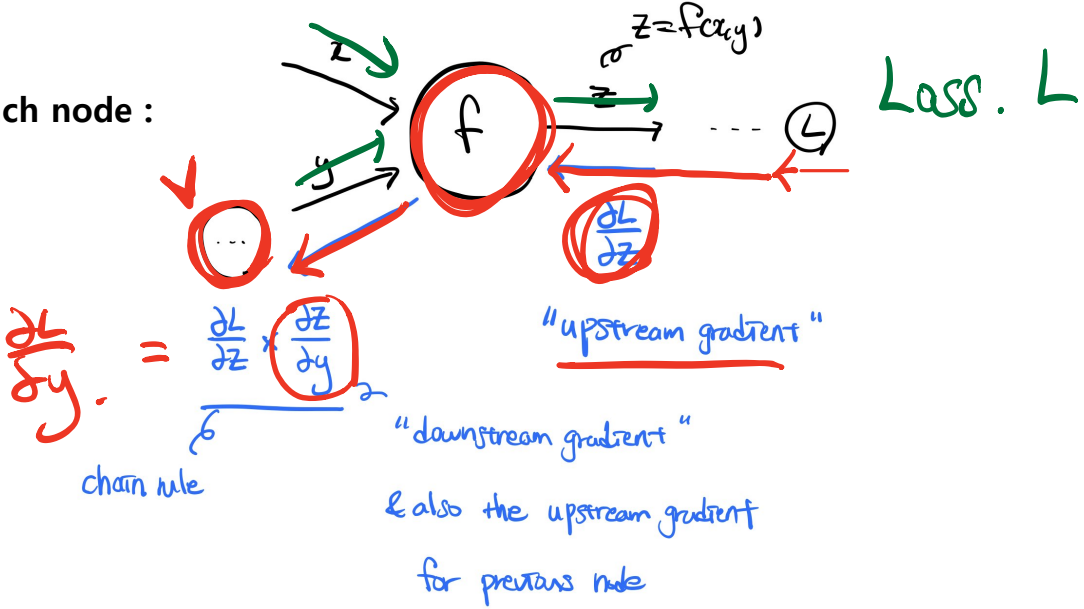
Backpropagation : Computational Graph

Interpreting each node :



Backpropagation : Computational Graph

Interpreting each node :



Backpropagation : Computational Graph

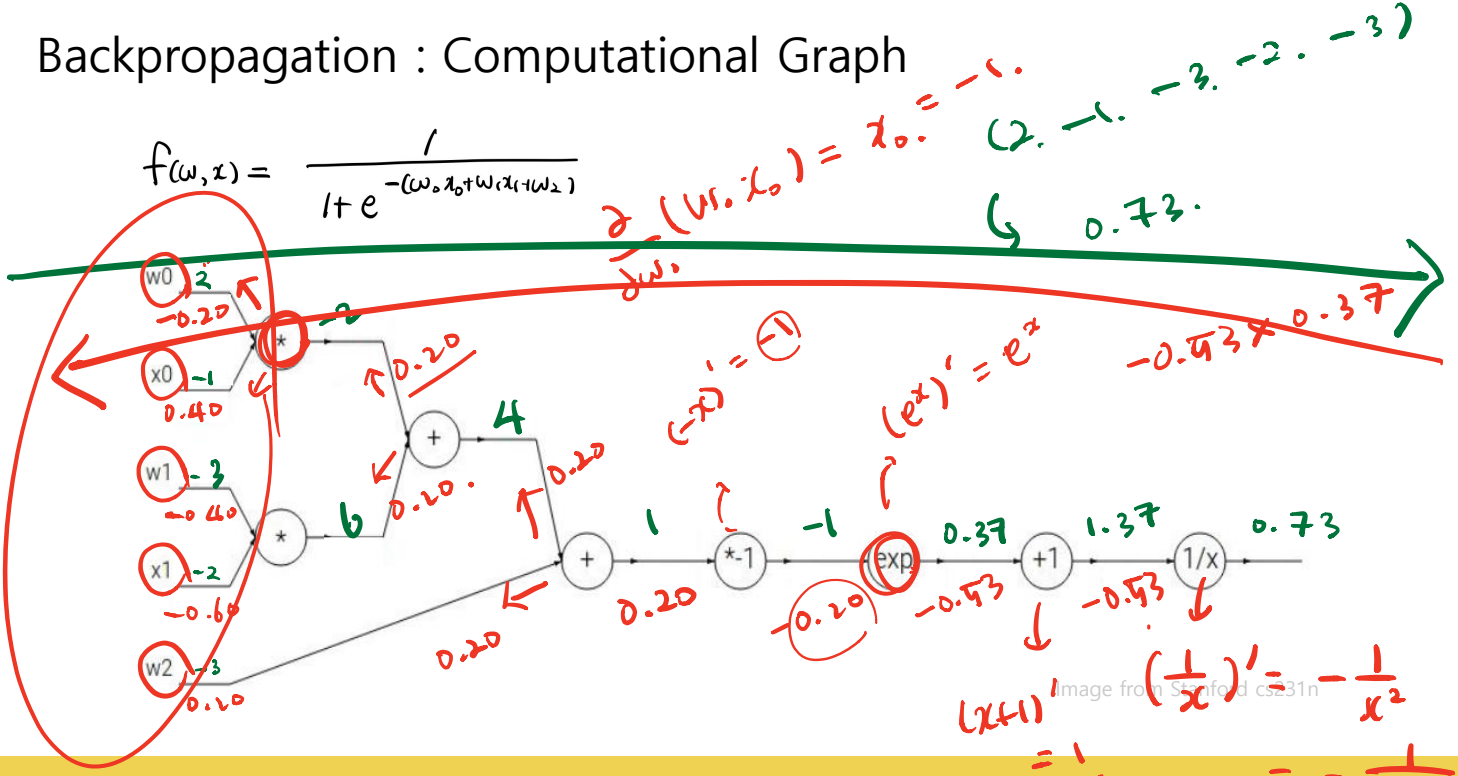
$(x+y)z$

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

Too simple…?

Try "$f(w,x) = \frac{1}{1+e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$"

Backpropagation : Computational Graph



Backpropagation : Computational Graph

Actually, inputs are not scalars

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W; b), y_i) + \lambda R(W)$$

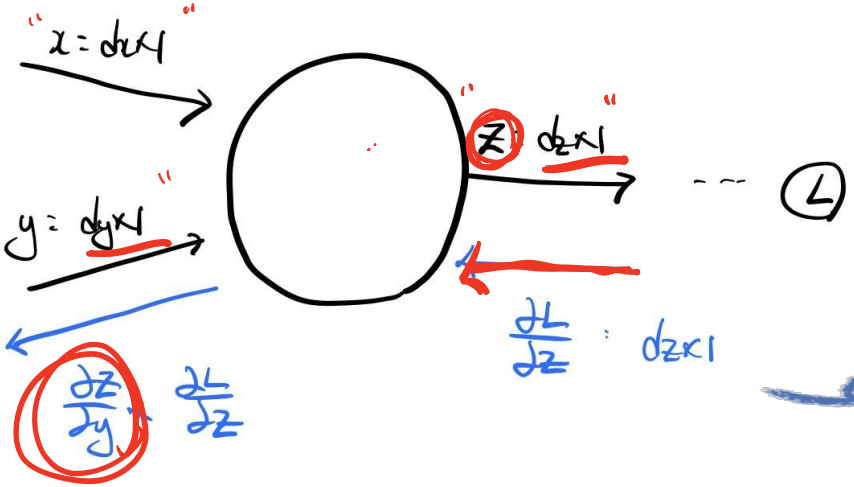
x : vector

W : matrix

b : vector

Backpropagation : Computational Graph

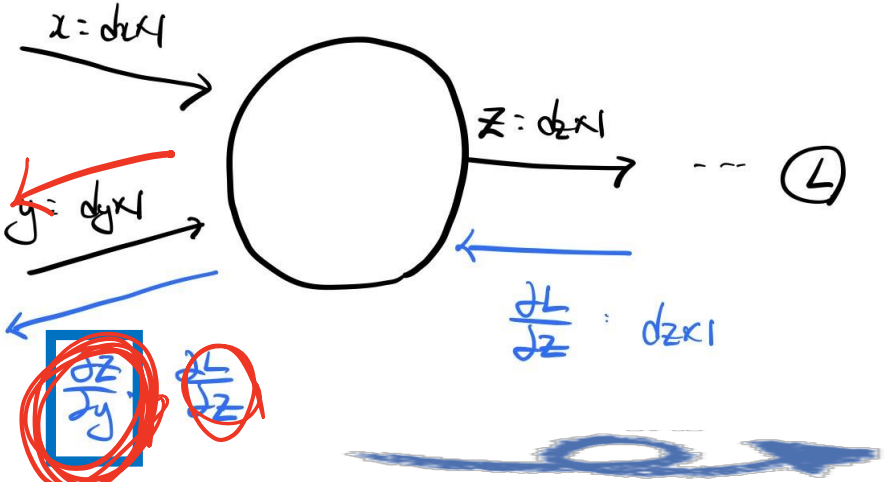
Vector in, Vector out



scalar L을 Z의 element들로 편미분한 vector

Backpropagation : Computational Graph

Vector in, Vector out



"Jacobian"

: element-wise derivative matrix

Backpropagation : Computational Graph

Vector in, Vector out

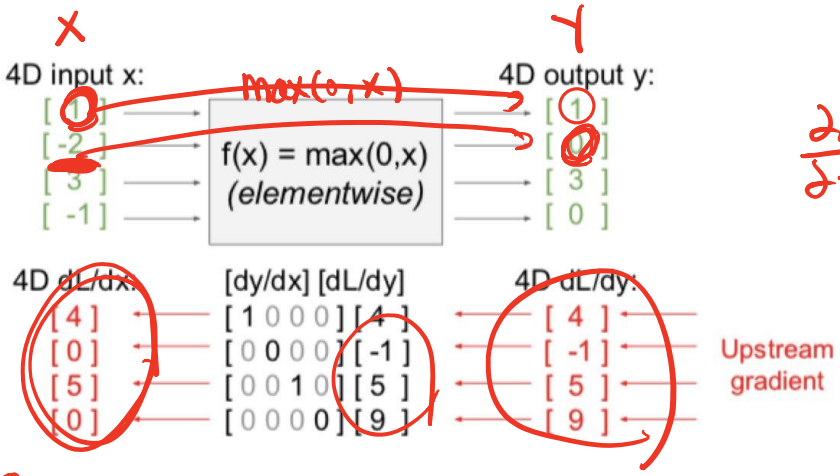
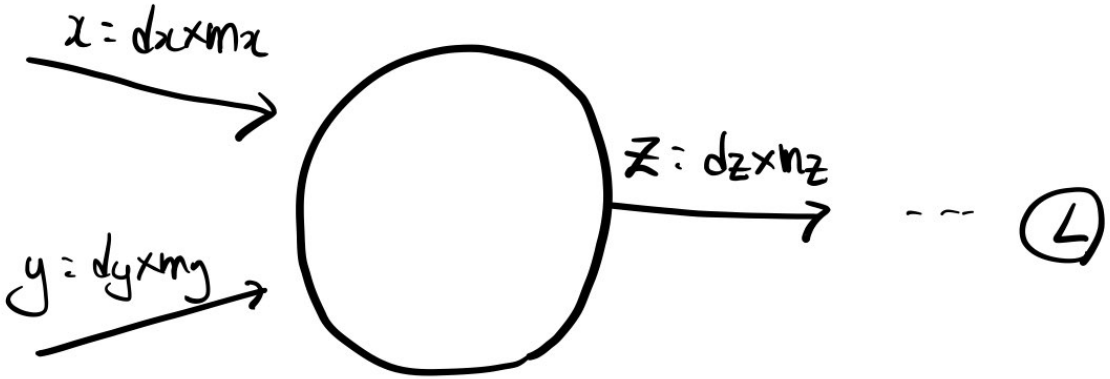


Image from Stanford cs231n

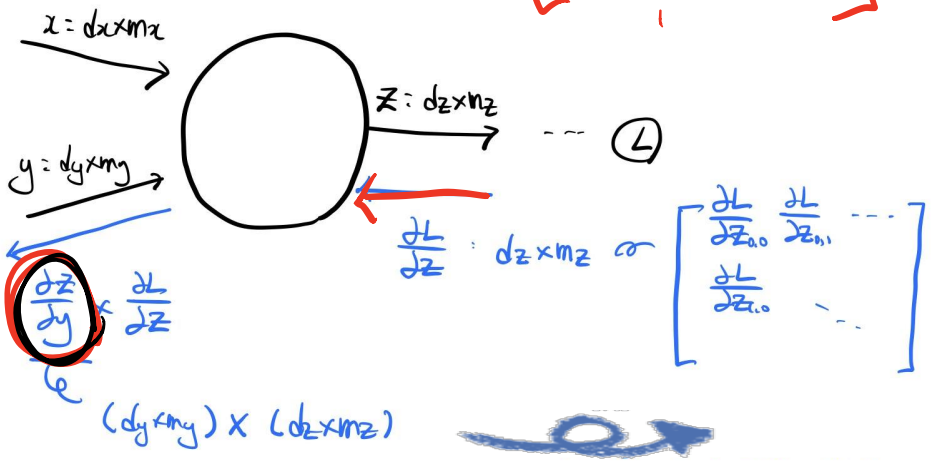
Backpropagation : Computational Graph

Matrix in, Matrix out

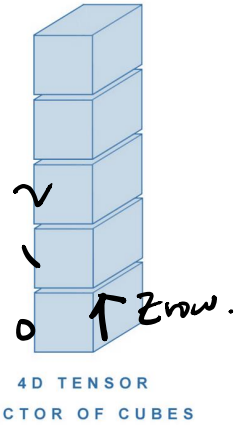


Backpropagation : Computational Graph

Matrix in, Matrix out

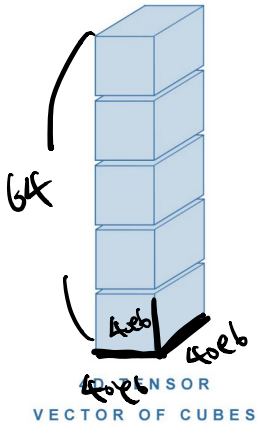


" " General Jacobian은 **4D Tensor**



Backpropagation : Computational Graph

Matrix in, Matrix out



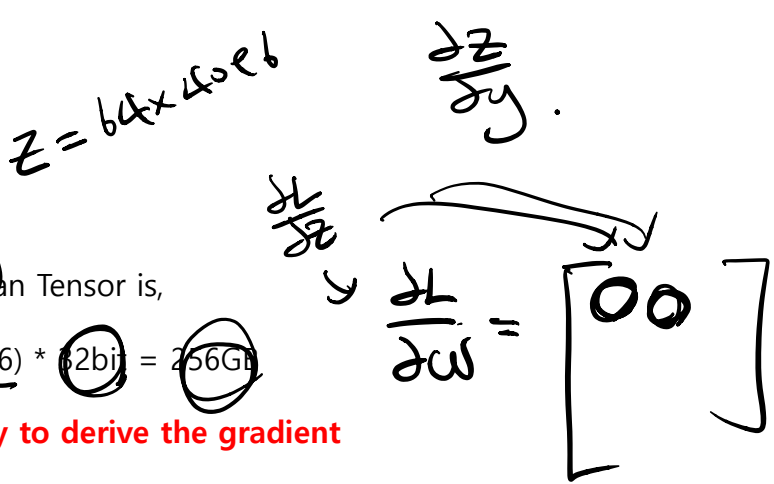
if inputs are,

x : 64 * 4096
y : 4096 * 4096

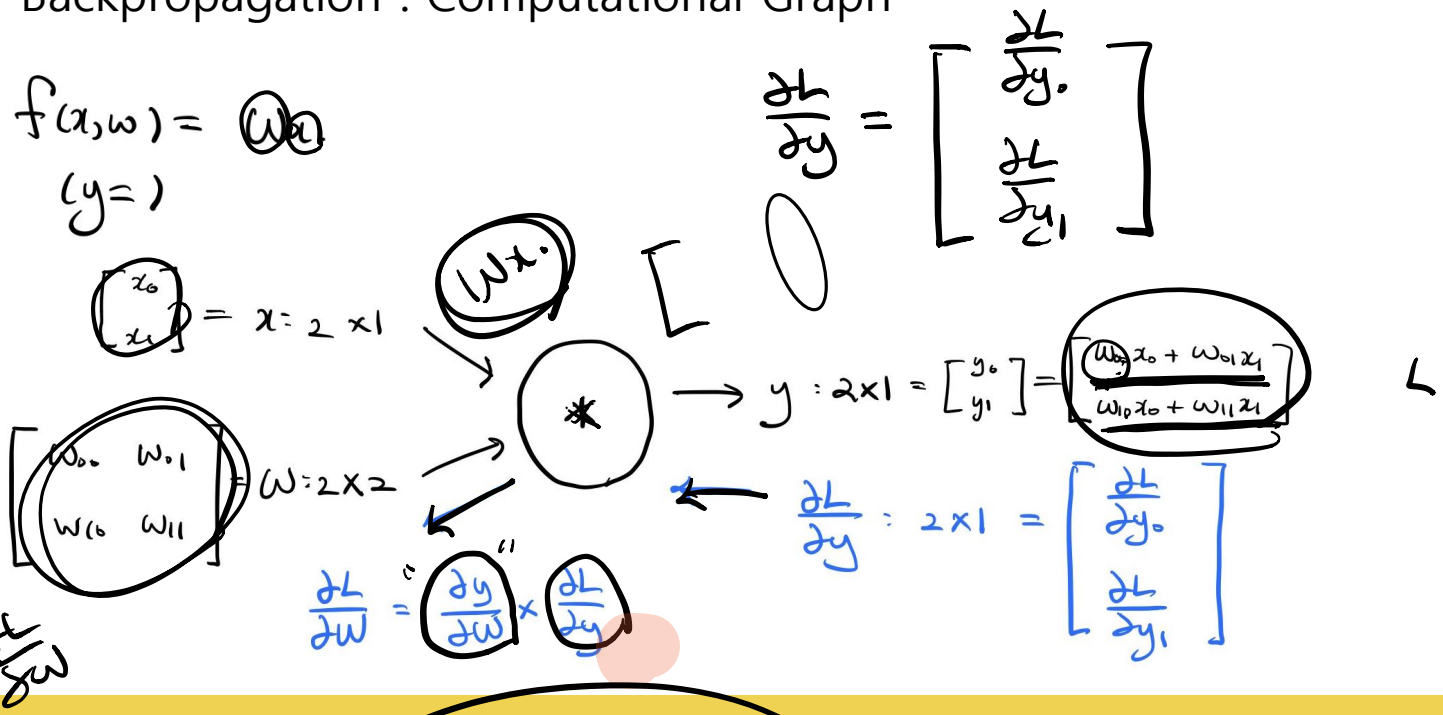
the size of the 4D Jacobian Tensor is,

(64 * 4996) * (4096 * 4096) * 32bit = 256GB

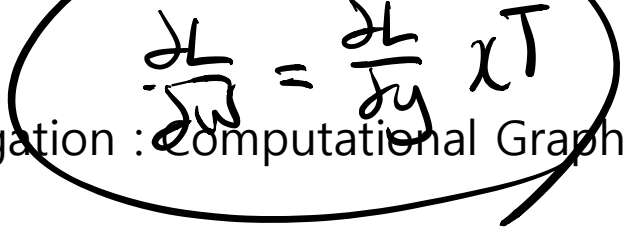
Should find another way to derive the gradient



Backpropagation : Computational Graph



Backpropagation : Computational Graph



$\frac{\partial L}{\partial W}$

instead of computing the General Jacobian, compute the gradient "elementwise"

$\frac{\partial L}{\partial W} = ?$

$\frac{\partial L}{\partial W} = \begin{bmatrix} \blacksquare \end{bmatrix}$

$\frac{\partial L}{\partial W_{00}} = x_0 \frac{\partial L}{\partial y_0}$ $\quad \frac{\partial L}{\partial W_{01}} = x_1 \frac{\partial L}{\partial y_1}$

$\frac{\partial L}{\partial W_{10}} = x_0 \frac{\partial L}{\partial y_1}$ $\quad \frac{\partial L}{\partial W_{11}} = x_1 \frac{\partial L}{\partial y_1}$

$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial y} x^T = \begin{bmatrix} \frac{\partial L}{\partial y_0} \\ \frac{\partial L}{\partial y_1} \end{bmatrix} \begin{bmatrix} x_0 & x_1 \end{bmatrix}$.

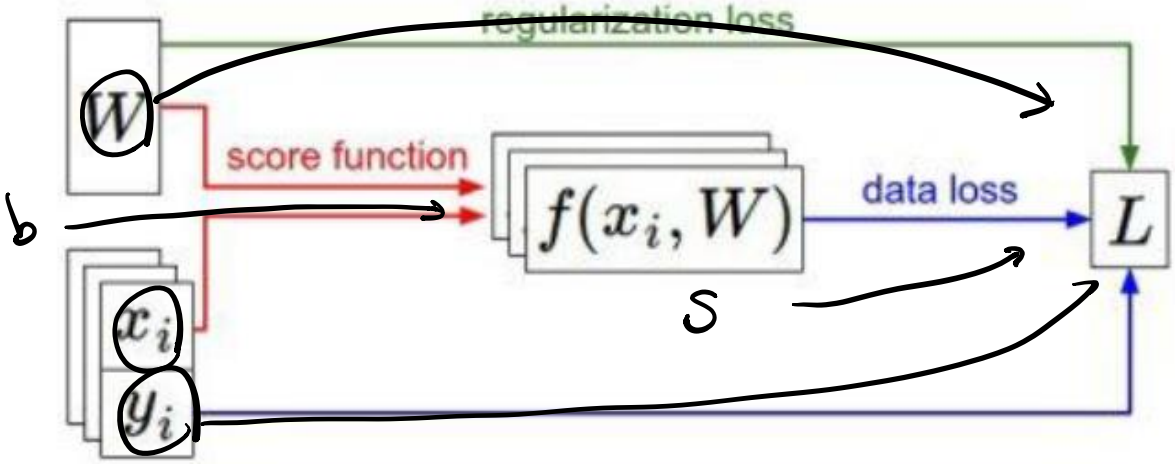
Backpropagation : Implementation

지금까지는, Computational Graph를 이용해,

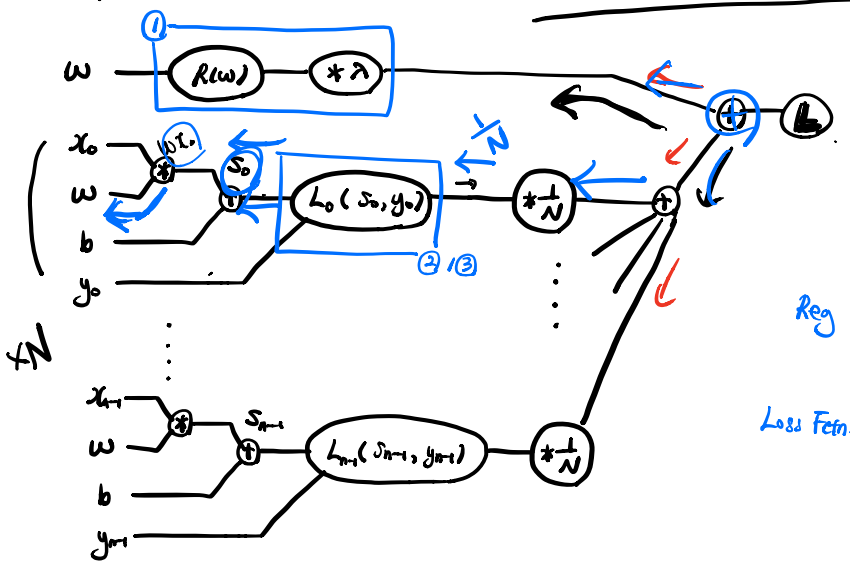
복잡한 함수의 gradient를 구하는 과정을 연습했다.

then, how can backpropagation be implemented in our linear classifier?

Backpropagation : Implementation



Computational Graph of, $L = \frac{1}{N}\sum_{i=1}^{N} L_i(Wx_i + b, y_i) + \lambda R(W)$



$\frac{\partial L}{\partial W}$, $\frac{\partial L}{\partial b}$.

Reg - ① Regularization Term

Loss Fcn. - ② Multiclass SVM Loss, ③ Cross-Entropy Loss

$\frac{\partial R}{\partial W_{0,0}} = 2W_{0,0} \cdots \frac{\partial R}{\partial W} = 2W$

1) $\frac{dL}{dW} = ?$

① $R(W) = \sum_k \sum_l W_{kl}^2 \rightarrow \lambda \times R(W)$.

$\frac{\partial}{\partial W}(\lambda R(W)) = \lambda \frac{\partial}{\partial W}(R(W)) = (2\lambda) \cdot W$

② M-SVM. $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$ (-1)

s.t. $k \neq y_i$

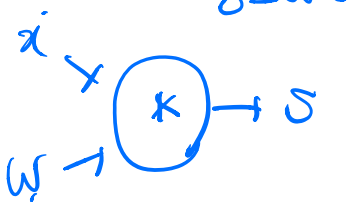
$S = Wx_0 + b$

$\frac{\partial L_i}{\partial S} = \begin{bmatrix} \frac{\partial L}{\partial S_0} \\ \frac{\partial L}{\partial S_1} \\ \vdots \end{bmatrix}$

$\frac{\partial L}{\partial S_k} = 1\left(S_k - S_{y_i} + \Delta > 0\right) \left(\sum_{\substack{j \neq y_i \\ k = y_i}} (-1)\left(S_j - S_k + \Delta > 0\right) \right)$

$$\frac{\partial L_i}{\partial s} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

$s = wx$



$\frac{\partial L_i}{\partial s}$

$$\begin{bmatrix} \frac{\partial L_i}{\partial s_0} \\ \frac{\partial L_i}{\partial s_1} \\ \vdots \end{bmatrix} \begin{bmatrix} x_0 & x_1 & \cdots \end{bmatrix}$$

$\frac{\partial s}{\partial w} = \frac{\partial L_i}{\partial s} x_i^T$

$$= \begin{bmatrix} \frac{\partial L_i}{\partial s_0} x_0 & \frac{\partial L_i}{\partial s_0} x_1 & \cdots \\ & & \end{bmatrix}$$

$$\frac{\partial L_i}{\partial w_k} = -\left(\sum_{j \neq y_i} 1\left((s_j - s_{y_i} + \Delta) > 0\right) \right) x_i^T \quad \text{if } k = y_i.$$

$$\left(1\left((s_j - s_{y_i} + \Delta) > 0\right) \right) x_i^T \quad \text{otherwise}$$

(3) Cross-Entropy

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \qquad \frac{\partial L_i}{\partial s}$$

$$= -s_{y_i} + \log\left(\sum_j e^{s_j}\right) \qquad \ln x \to \frac{1}{x}$$

$$\frac{\partial L_i}{\partial s_{y_i}} = -1 + \frac{e^{s_{y_i}}}{\sum_j e^{s_j}}$$

$$\frac{\partial L_i}{\partial s_k} = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

$$\frac{\partial L_i}{\partial s} \times x^T = \frac{\partial L_i}{\partial W}$$

$$\frac{\partial L}{\partial W} = (2\lambda)\cdot W + \frac{1}{N}\sum_{i=1}^{N} \frac{\partial L_i}{\partial W}$$

M-SVM & CE.

Backpropagation : Implementation

Implement the ~~S~~GD (or just Gradient Descent Algorithm)

to your Linear Classifier!

Backpropagation : Implementation

Model . . . . . .

Model.fit()

앞으로 나올 MLP (Multilayer Perceptron), CNN (Convolutional Neural Network)는

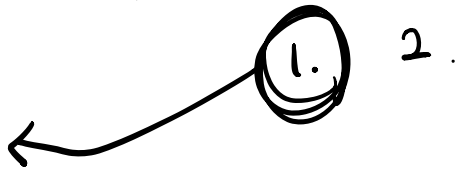
훨씬 복잡한 Computational Graph 가짐.

but, 대부분의 function (or node)의 gradient는 검색을 통해 찾을 수 있으며,

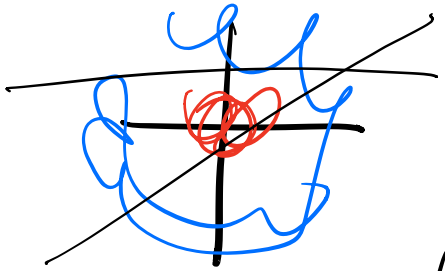
대부분 라이브러리에 이미 구현되어 있다!

Review

1. **Optimization**
   - SGD vs. ~~Random Search~~
   - Analytic Gradient vs. ~~Numeric Gradient~~
2. **Backpropagation**
   - Chain Rule
   - Computational Graph
     - scalar
     - vector
     - matrix
   - Implementation



Preview on Next Class

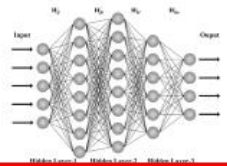


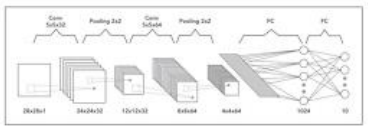
Optimization : Calculating Gradient

IDEA 2. Analytic

지금은 simple linear classifier만 다루고 있지만,

앞으로 다룰 model은





then, how should we derive the gradient?

ANN.

Multilayer Perceptron

Linear Classifier 단독으로는

풀 수 없는 문제들 존재...

여러 개를 중첩시켜 보자!