Lecture 3.
Model Optimization

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**

21

Review

1. **Loss**
2. **Loss Function**
   - Multiclass SVM Loss
   - Cross-Entropy Loss
3. **Regularization**
   - Why it is needed : to discourage large weight matrix
   - Overfitting

Review

How is a Model Optimized / Updated?

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

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

IDEA 1. Random Search

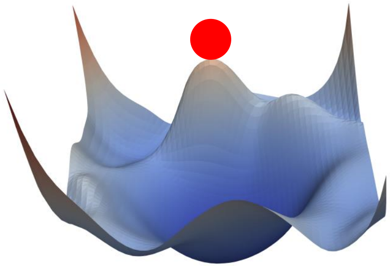
1. Randomly set W and b
2. Select (W, b) that gives the minimum loss in training

Obviously very poor ☹

Optimization : Idea

IDEA 2. Gradient Descent

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

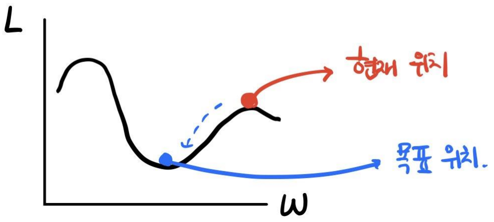


Loss의 Global Minimum으로 가려면 어떻게 해야할까?

Optimization : Idea

IDEA 2. Gradient Descent

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



● 에서 ● 로 가는 방법?

● 에서의 L의 기울기 방향으로 가면 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

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

IDEA 2. Gradient Descent

learning rate

$$W_{new} = W_{old} - lr \cdot \frac{\partial L}{\partial W}\bigg|_{W = W_{old}}$$

$$b_{new} = b_{old} - lr \cdot \frac{\partial L}{\partial b}\bigg|_{b = b_{old}}.$$

Optimization : Calculating Gradient

learning rate

$$w_{new} = w_{old} - lr \cdot \frac{\partial L}{\partial w}\Bigg|_{w = w_{old}}$$

$$b_{new} = b_{old} - lr \cdot \frac{\partial L}{\partial b}\Bigg|_{b = b_{old}}.$$

then, how do we calculate the Gradient?

Optimization : Calculating Gradient

IDEA 1. Numeric

미분의 정의 활용

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

h에 10^{-4} 같은 작은 수 대입.

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

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

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

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

도함수를 직접 구해서 값을 대입해 미분값 구하기

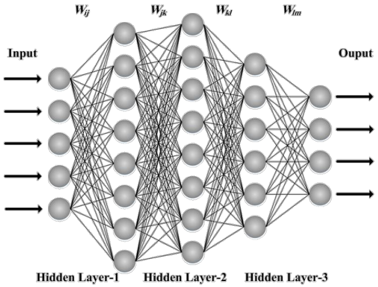
$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda \sum_{k}\sum_{l} W_{kl}^2$$

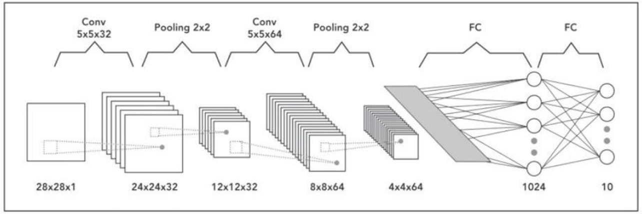
Optimization : Calculating Gradient

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

We compute the Gradient via, **Backpropagation**

Backpropagation : Idea

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

1. 복잡한 함수는 간단한 함수/연산들의 합성함수로 나타낼 수 있다.

2. 간단한 함수들에 Chain Rule을 적용하여, 복잡한 함수의 미분계수를 구한다.

Backpropagation : Computational Graph

복잡한 함수를 간단한 함수/연산들의 합성으로 나타낼 때,

Computational Graph를 활용 (Node : Operator, Leaf : Operand)

Backpropagation : Computational Graph

f(x, y, z) = (x + y) * z

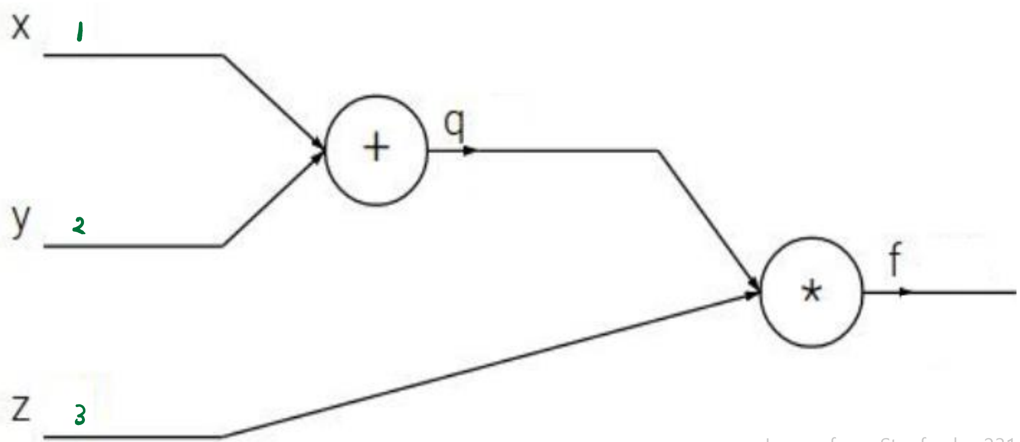
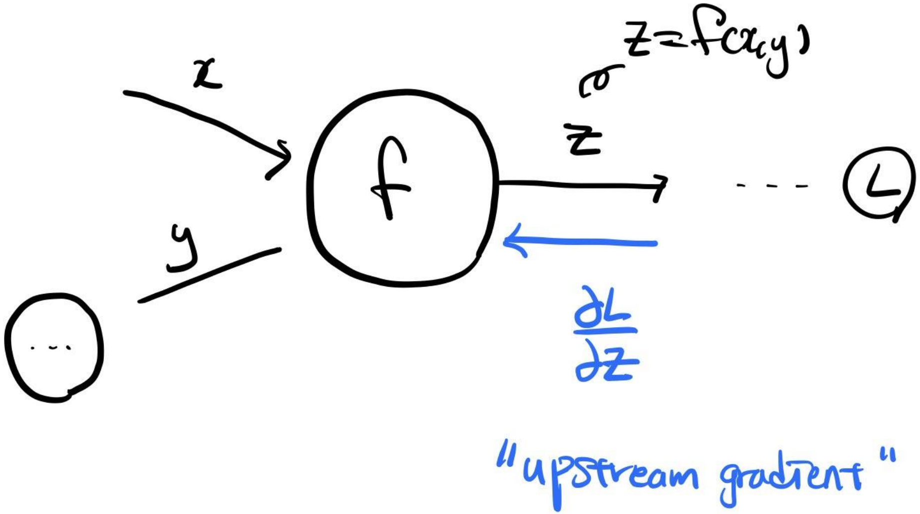


Image from Stanford cs231n

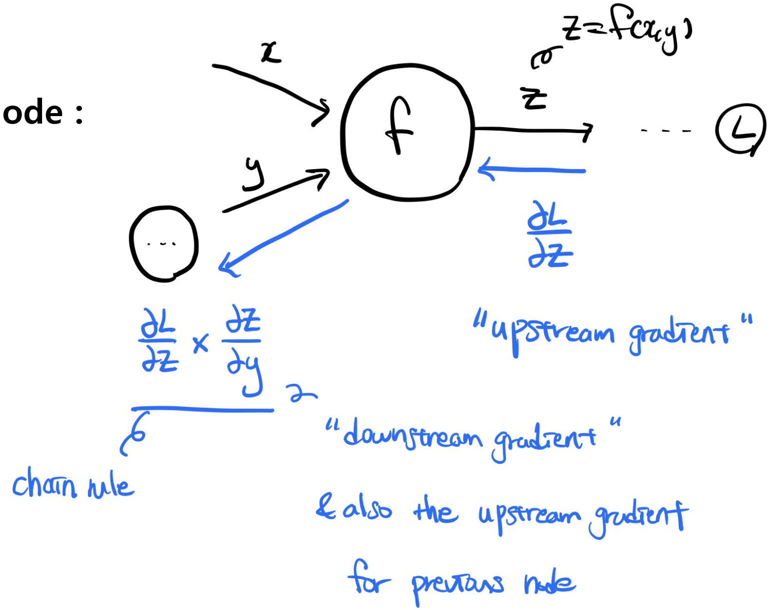
Backpropagation : Computational Graph

Interpreting each node :



Backpropagation : Computational Graph

Interpreting each node :



Backpropagation : Computational Graph

Too simple…?

Try

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

Backpropagation : Computational Graph

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

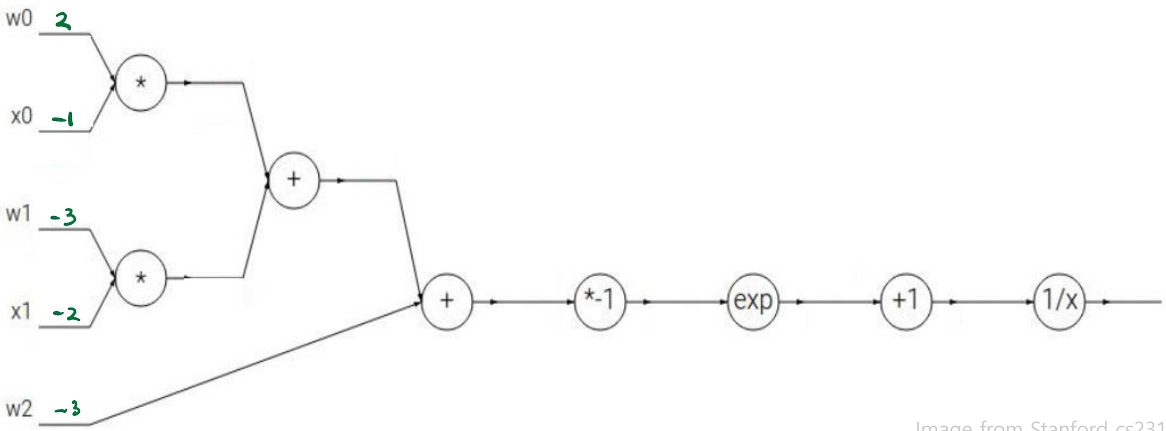


Image from Stanford cs231n

Backpropagation : Computational Graph

Actually, inputs are not scalars

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

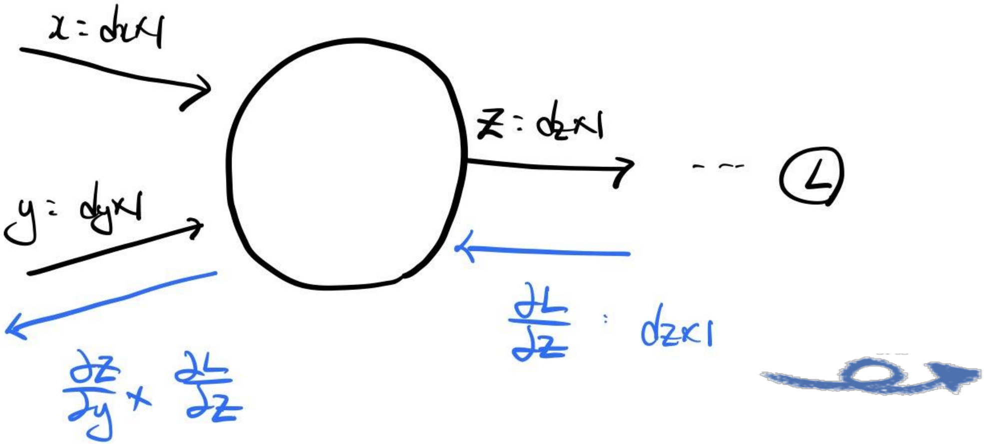
x : vector

W : matrix

b : vector

Backpropagation : Computational Graph

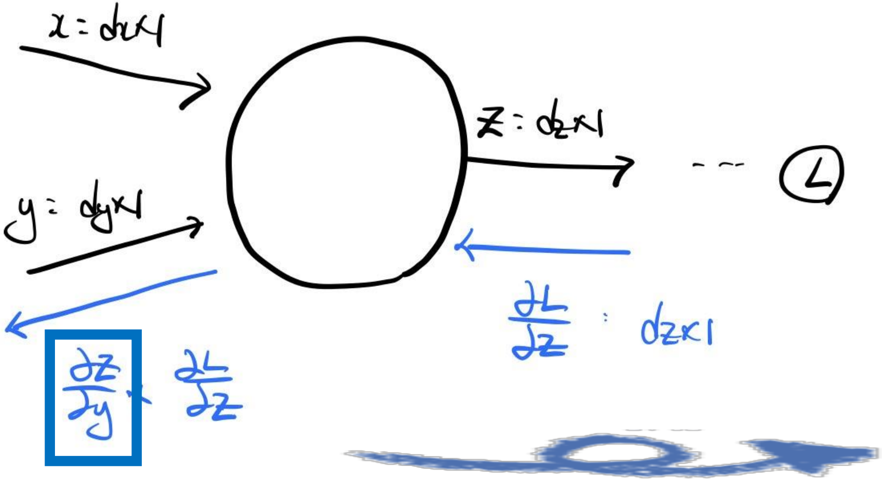
Vector in, Vector out



scalar L을 Z의 element들로
편미분한 vector

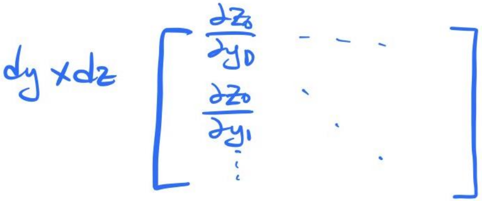
Backpropagation : Computational Graph

Vector in, Vector out



Jacobian

: element-wise derivative matrix



Backpropagation : Computational Graph

Vector in, Vector out

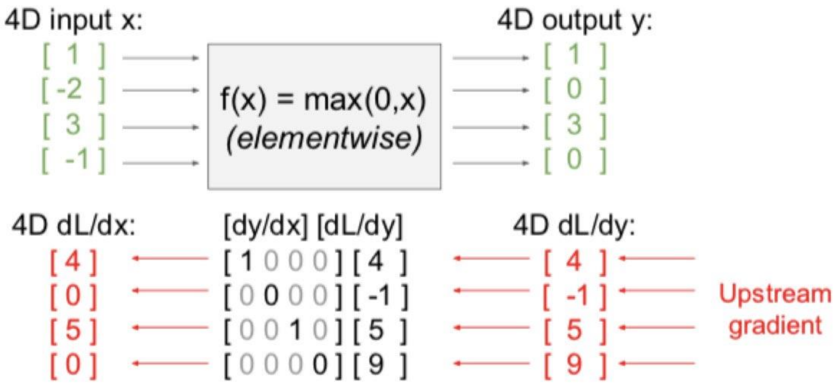
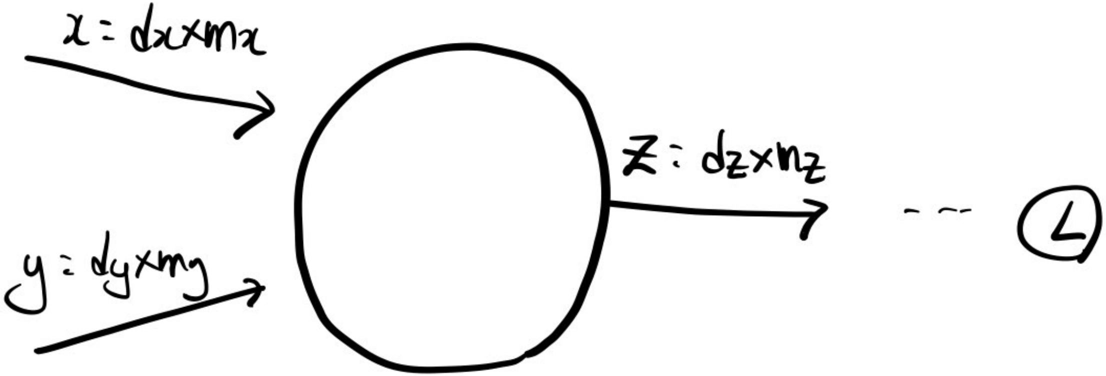


Image from Stanford cs231n

Backpropagation : Computational Graph

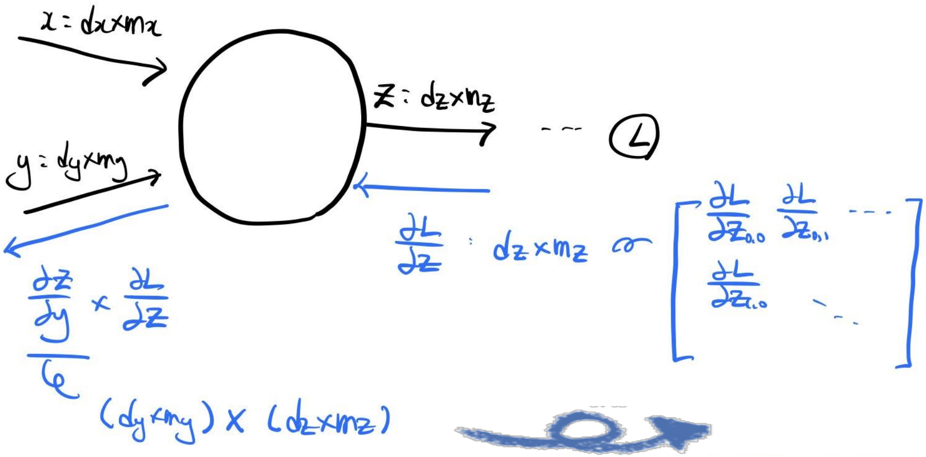
Matrix in, Matrix out

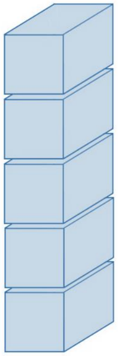


Backpropagation : Computational Graph

Matrix in, Matrix out

General Jacobian은 **4D Tensor**



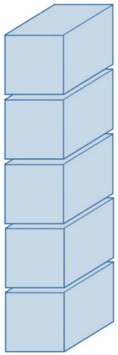


4D TENSOR

VECTOR OF CUBES

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

$$f(x, w) = Wx.$$

$$(y =)$$

$$\begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = x : 2 \times 1$$

$$\begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} = W : 2 \times 2$$

$$*$$

$$y : 2 \times 1 = \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} = \begin{bmatrix} W_{00}x_0 + W_{01}x_1 \\ W_{10}x_0 + W_{11}x_1 \end{bmatrix}$$

$$\frac{\partial L}{\partial y} : 2 \times 1 = \begin{bmatrix} \frac{\partial L}{\partial y_0} \\ \frac{\partial L}{\partial y_1} \end{bmatrix}$$

$$\frac{\partial L}{\partial W} = \frac{\partial y}{\partial W} \times \frac{\partial L}{\partial y}$$

Backpropagation : Computational Graph

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

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

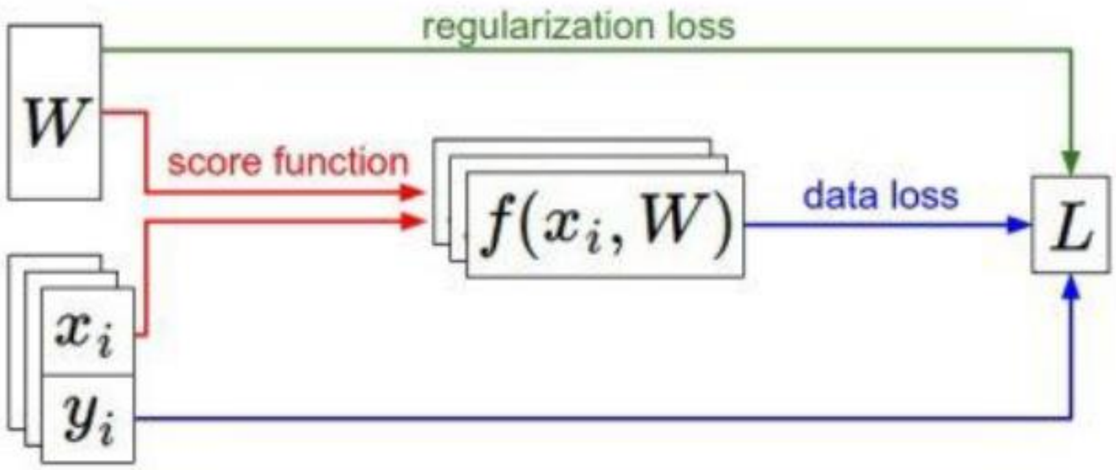
Backpropagation : Implementation

지금까지는, Computational Graph를 이용해,

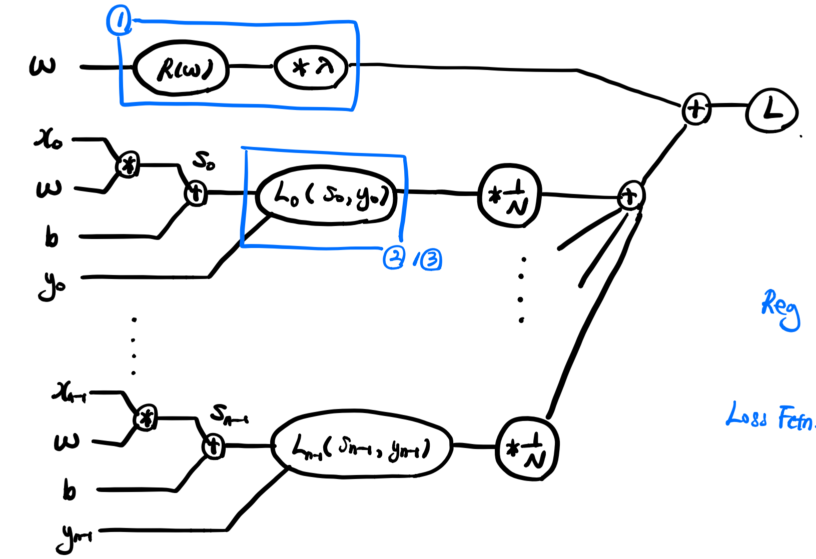
복잡한 함수의 gradient를 구하는 과정을 연습했다.

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

Backpropagation : Implementation



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



Reg - ① Regularization Term

Loss Fctn. - ② Multiclass SVM Loss

③ Cross-Entropy Loss

1) $\frac{dL}{\partial W} = ?$

Backpropagation : Implementation

Implement the SGD (or just Gradient Descent Algorithm)

to your Linear Classifier!

Backpropagation : Implementation

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

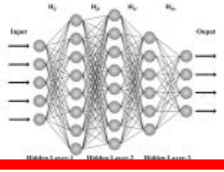
Preview on Next Class

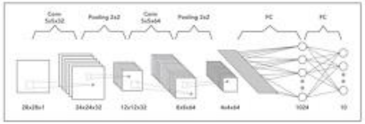
Optimization : Calculating Gradient

IDEA 2. Analytic

지금은 simple linear classifier만 다루고 있지만,

앞으로 다룰 model은





then, how should we derive the gradient?

Multilayer Perceptron

Linear Classifier 단독으로는

풀 수 없는 문제들 존재…

여러 개를 중첩시켜 보자!