cs231n ss1 svm

svm_loss_naive

$$\begin{cases} W & D\times C \\ X & N\times D \\ Y & N \\ reg \end{cases} \rightarrow \begin{cases} loss \\ grad & D\times C \end{cases}$$

*클래스 개수 = C

*학습 data = N

$\hat{y} = XW$

$\hat{y} - y + \Delta > 0 \Rightarrow loss \uparrow$ (Δ delta)

$$\Sigma loss := \frac{\Sigma loss}{N} + \lambda \|W\|^2$$

1) loss function 의 gradient dW

2) loss 계산.

① Loss

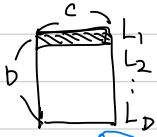
$$L = \frac{1}{N}\sum_i \sum_{j\neq y_i} \left[\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + \Delta)\right] + \lambda \sum_k \sum_l W_{k,l}^2$$

i : 각 sample

j : 각 class 별 예측치의 index

y_i : 〃 정답인덱스 〃

② Derivative



$$\frac{dL_i}{dW} = \left[\frac{dL_i}{dW_1} \ \frac{dL_i}{dW_2} \cdots \frac{dL_i}{dW_C}\right]$$

$$= \begin{bmatrix} \frac{dL_i}{dW_{11}} \\ dW_{21} \\ \vdots \\ dW_{D1} \end{bmatrix}_{D\times C}$$

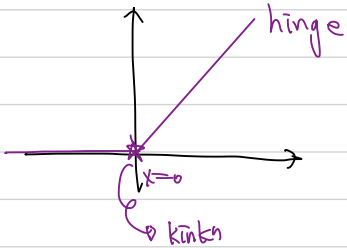
$\xrightarrow{dW}$

$$\nabla_{W_j} L_i = \mathbb{1}(\hat{f} - f + \Delta > 0)\, x_i$$

$$\nabla_{W_{y_i}} L_i = \mathbb{1}(\hat{f} - f + \Delta > 0)\,(-x_i)$$

∵ W가 두종류, W_j, W_{y_i}

→ 두가지 derivative 모두 계산해야함.

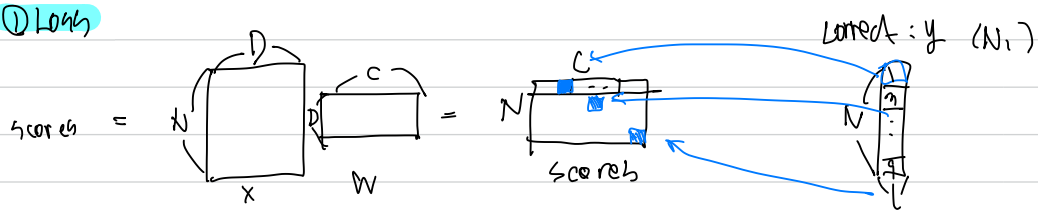


failed gradcheck ← 여기 계산한 게 맞음.

반복시 → 주의

svm_loss_vectorized

① Loss



yscores : 축 index = (0, 1), (1, 2) ---, (N-1, y) → (N,)

☺ = scores[0~N, y]

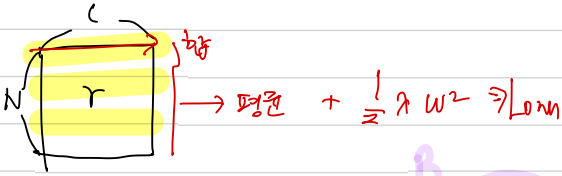
margin : scores − yscores + 1 // 0

NXC Nx1 → soon Broadcasting △ max ?

A

정답인 경우 제거 (0 처리)

margin 값

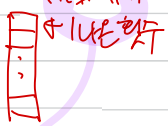


B

② derivative

Binary [margin > 0 : 1 → sum → 틀린 개수

< 0



→ dW

$\nabla_{w_j} L_i = 1(f - f + \Delta > 0)\, x_i$

$\nabla_{w_{y_i}} L_i = 1(f - f + \Delta > 0)\,(-x_i)$

∵ W가 두개, w_j, w_{y_i}

→ 두가지 derivative 모두 계산해야함.

A−B

$\hat{y}$ y_i $\hat{y}$

$-\eta$

1 or 0

* 기본에 Binary 형태로 해서 정답인 point 값 = 0 이었음.

와 X 내적 → 

DXC

$+x_i$

$-x_i$ × 정답 횟수.

Binary'

NXC

Linear classifier

$X : N \times D$

$y : (N, \ )$

classes = C

$D \times C$ $\quad$ $N \times D$

W이용 → X : (N×D 여기(N, 1) ypred

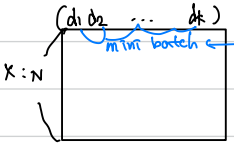
$\hat{y} = \underbrace{XW}_{N \times C}$ → axis 1 argmax

W 3073 × 10

X 49 000 × 3073

Layers.py

def affine_forward



reshape to $D = d_1 \times d_2 \times \cdots \times d_k$ → output

dim(Output) = M

X : shape $(N, d_1, d_2, \cdots, d_k)$ $\quad$ out : $N \times M$

w : $D \times M$ → cache : (x, w, b)

b : $(M, \ )$

$x' \cdot w \Rightarrow x'w \oplus b = out$

$N \times D$ $\quad$ $D \times M$ $\quad$ $N \times M$

reshape & broadcast

$(1, M) \rightarrow (N, M)$

def affine_backward

Upstream 미분값

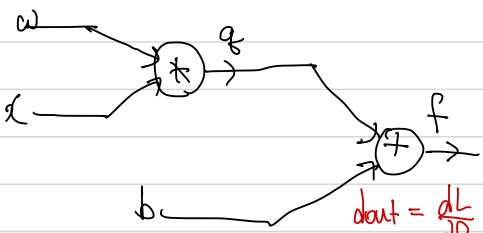
dout : (N, M) → dx $(N, d_1, d_2, \cdots, d_k)$

cache : (x, w, b) $\quad$ dw (D, M)

db $(M, \ )$

dout : $\quad$ M — dann score derivatives

(pseudo)



$dout = \frac{dL}{df}$

$q = w * x$

$f = q + b$

$\frac{df}{db} = 1$ $\quad$ $\frac{df}{dq} = 1$

$\frac{dq}{dw} = x$ $\quad$ $\frac{dq}{dx} = w$

(Sol) $\frac{dL}{db} = \frac{dL}{df}\frac{df}{db} = \frac{dL}{df} = dout$

$(M, \ )$ $\quad$ $N \times M$

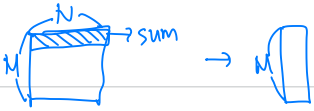
$\frac{dL}{dw} = \frac{dL}{df}\frac{df}{dw} = \frac{dL}{df}\frac{df}{dq}\frac{dq}{dw} = dout \cdot 1 \cdot x$

$D \times M$ $\quad$ $N \times M$ $\quad$ $(N, d_1, d_2, \cdots d_k) \Rightarrow D$

$\frac{dL}{dx} = \frac{dL}{df}\frac{df}{dx} = \frac{dL}{df}\frac{df}{dq}\frac{dq}{dx} = dout \cdot 1 \cdot w$

$N, d_1, \cdots, d_k$ $\quad$ $N \times M$ $\quad$ $D \times M$

In Detail :

① $dout^T$ = 

② $(dout^T x')^T$

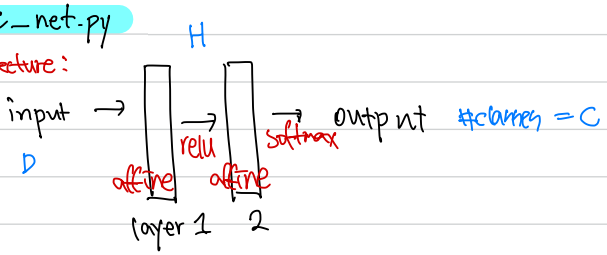
$M \times N \quad N \times D$

③ $dout \cdot W^T \rightarrow N \times D \rightarrow N \times (D_1 \times D_2 \times \cdots \times D_k)$

$N \times M \quad M \times D$

fc_net.py

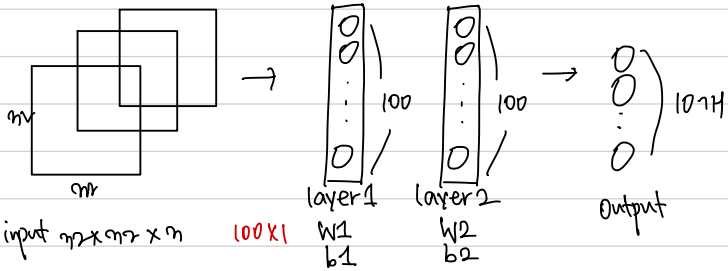
Architecture :



Notice : 1) fully-connected

2) Optimizing 하는 Solver 따로 존재

3) self.params ← learnable parameter를 저장해 두는 딕셔너리



input $m_2 \times m_2 \times n$ $\quad$ 100×1

⊕ reg, weight scale

self.params → weight : 가우스 분포 0 으로 초기화 / 표준편차 = weight_scale

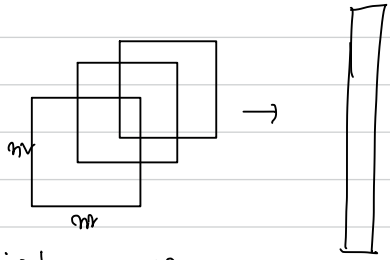
bias = zero <초기화>

def loss : x : (N, d1, d2, ..., dk)
y : (N,) labels

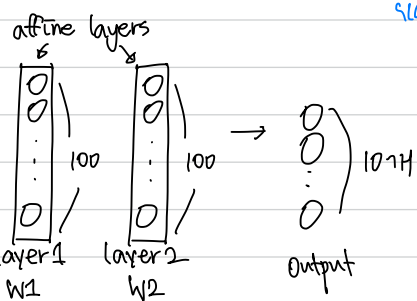
→ y is None → test-time forward ⇒ scores : (N, C) (classification scores)

→ y ≠ None → training-time for/back ⇒ (loss, grads)

loss : scalar
grads : dictionary (W1, W2, b1, b2)



input m×m×n



x (m0m2, 1)

W1 : (m0m2, 100)
b1 = (100,)
⇓
(, 100) → relu
↓
(100,)

W2 = (100, 10)
b2 = (10,)
↓
→ softmax → (10, 1)

$w_1^T x$
100 × 1

→ out : (m0m2 × 100) → (m0m2 × 10)

<backward>

grads = { } ⇒ loss 값, gradients

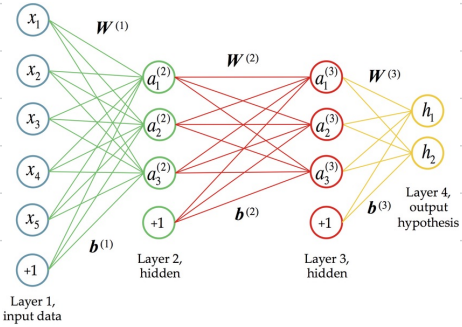
softmax 와 loss 계산

self.params[k] → grads[k]

① softmax_loss(x, y) → loss, dx
(N×C) (N,)

② affine_backward(dout, cache)
→ dx, dw, db N×M x×w

참고 Architecture for 2층



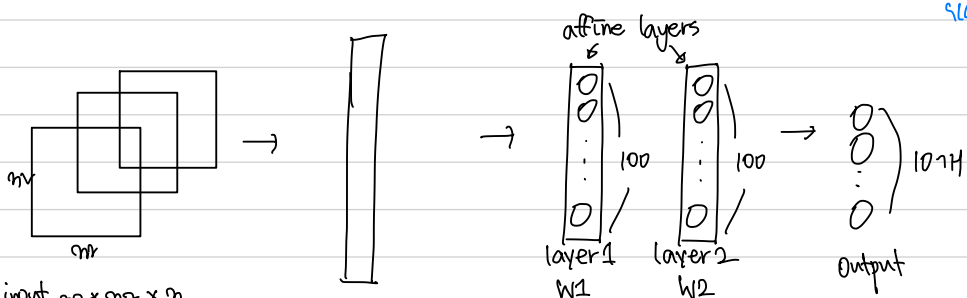
def loss : $x : (N, d_1, d_2, \dots, d_k)$

$y : (N,)$ labels

→ y is None → test-time forward ⇒ scores : (N, C) (classification scores)

→ y ≠ None → training-time for/back ⇒ (loss, grads) (dict: W1, W2, b1, b2)

loss: scalar, grads: dictionary



X (20012,1)

W1 : (20012, 100) W2 : (100, 10)

b1 = (100,) b2 = (10,)

← EXAMPLE