Neural networks - 2

Structure

① Recap (15 mins)

- MLP Recap
- Softmax Recap
- Categorical Cross-Entropy
- Training Neural networks

Problems Addressed

→ Multi-Class Classification

↳ Stack multiple LR units

→ How to interpret as probability?

→ Softmax soln

↳ Better than normal division

→ Matrix View of Multiple LR's

$$X \quad \begin{bmatrix} x_1^1 & \cdots & x_n^1 \\ x_1^2 & \cdots & x_n^2 \\ \vdots & & \\ x_1^m & \cdots & x_n^m \end{bmatrix} \qquad W \quad \begin{bmatrix} w_1^1 & w_1^2 & \cdots & w_1^C \\ w_2^1 & w_2^2 & \cdots & \\ \vdots & \vdots & & \vdots \\ w_n^1 & w_n^2 & & w_n^C \end{bmatrix}$$

$m \times n$ $\quad$ $n \times C$

$$b$$

$m \times C$

$$\begin{bmatrix} b_1 & b_2 & b_3 \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \end{bmatrix}$$

Broadcasting

→ Entropy

→ Measure of surprise

- → Cross Entropy Loss

$H(P,Q) = \sum P(x) \log Q(x)$

$H(P,Q)$ ⇒ measures how well one prob distribution Q (predicted) approximates another distribution P (true distribution)

"Expected no. of bits needed to encode events from true distribution P, using optimal code for predicted distribution"

→ If Q matches P, then cross entropy reduces to entropy of P.

→ If Q diverges from P, then H(P,Q) ↑, because more bits are needed to encode events from P using code optimize by Q

Backpropagation (60 mins)

Problem?

→ We want to minimize our loss,

= So we have a function

How do we optimize a function?

$$Loss(W) = \frac{1}{} \sum loss(x, W, y_i)$$

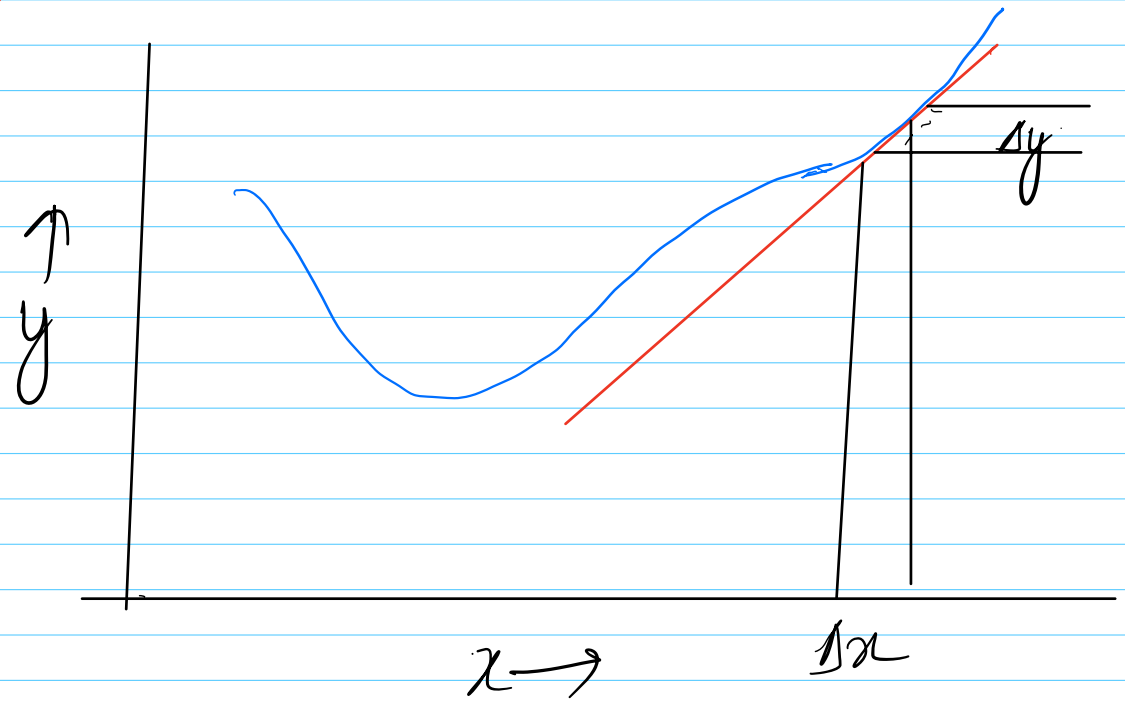
T

$$W_{opt} = \underset{W}{\operatorname{argmin}}\ Loss(W)$$

→ minimize error over training samples and hope to work ☺

Move to next come back

Quick Intro to Derivatives



→ wts derivative

$$\Delta y = \alpha \Delta x$$

(At fine enough resolution, any smooth func^n is locally linear)

Cases

① x & y Scalar

$$y = f(a)$$

$$\Delta y = \alpha \Delta x$$

② y Scalar, x vector

y = Scalar value

x = vector

Δx = ??

$\Delta x = [\Delta x_1, \Delta x_2, \ldots \Delta x_B]$

$\Delta y = \alpha \Delta x$

where

$\alpha = \nabla_x y = \left[\frac{\partial y}{\partial x_1}, \ldots \frac{\partial y}{\partial x_B}\right]$ ← Partials

⇒ This is derivative

→ Gradient is just transpose of above

→ Here derivative $\nabla_x f(X)$ is a scalar function $f(X)$

of multi-variate input X

$$\boxed{\nabla_x f(X)^T = \text{Gradient}}$$

$$\nabla_X f(X) = \left[\frac{\partial f(X)}{\partial x_1}, \frac{\partial f(X)}{\partial x_2} \cdots \frac{\partial f(X)}{\partial x_n} \right]$$

$$\boxed{df(X) = \nabla_X f(X) \cdot dX}$$

We want this to be maximized

→ Vector inner product

When is this inner product maximized?

→ When both vectors align

i.e

dX and $\nabla_X f(X)^T$ are in same direction

i.e

dX is exactly in dir^n of gradient

Hence we take a step in direction of gradient

→ Why we need this ??

$\nabla_x f(X) = 0$

and solve for X

→ Becomes complex, to solve

$$f(x_1, x_2, x_3)$$

$$= x_1^2 + x_1(1 - x_2) + x_2^2 - x_2 x_3 + x_3^2 + x_3$$

Grad

$$\nabla_x f^T = \begin{bmatrix} 2x_1 + 1 - x_2 \\ -x + 2x_2 - x_3 \\ -x_2 + 2x_3 + 1 \end{bmatrix}$$

$$\nabla_x f = 0 \Rightarrow \begin{bmatrix} 2x_1 + 1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 + 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

3 eqns, 3 unknowns

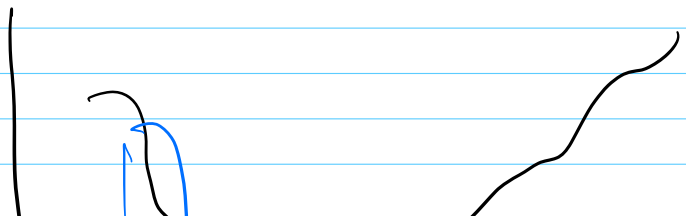
$$AX = B \Rightarrow A^{-1}AX = A^{-1}B$$

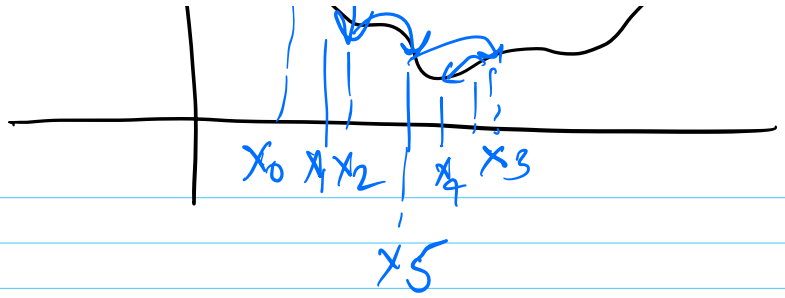
NNs have millions of params,
doing matrix inversion is computationally expensive at such dimension, so we need to find other way

What can we use then?

→ Iterative Solutions

① Start from guess





② Take a step towards hopefully better value of X.

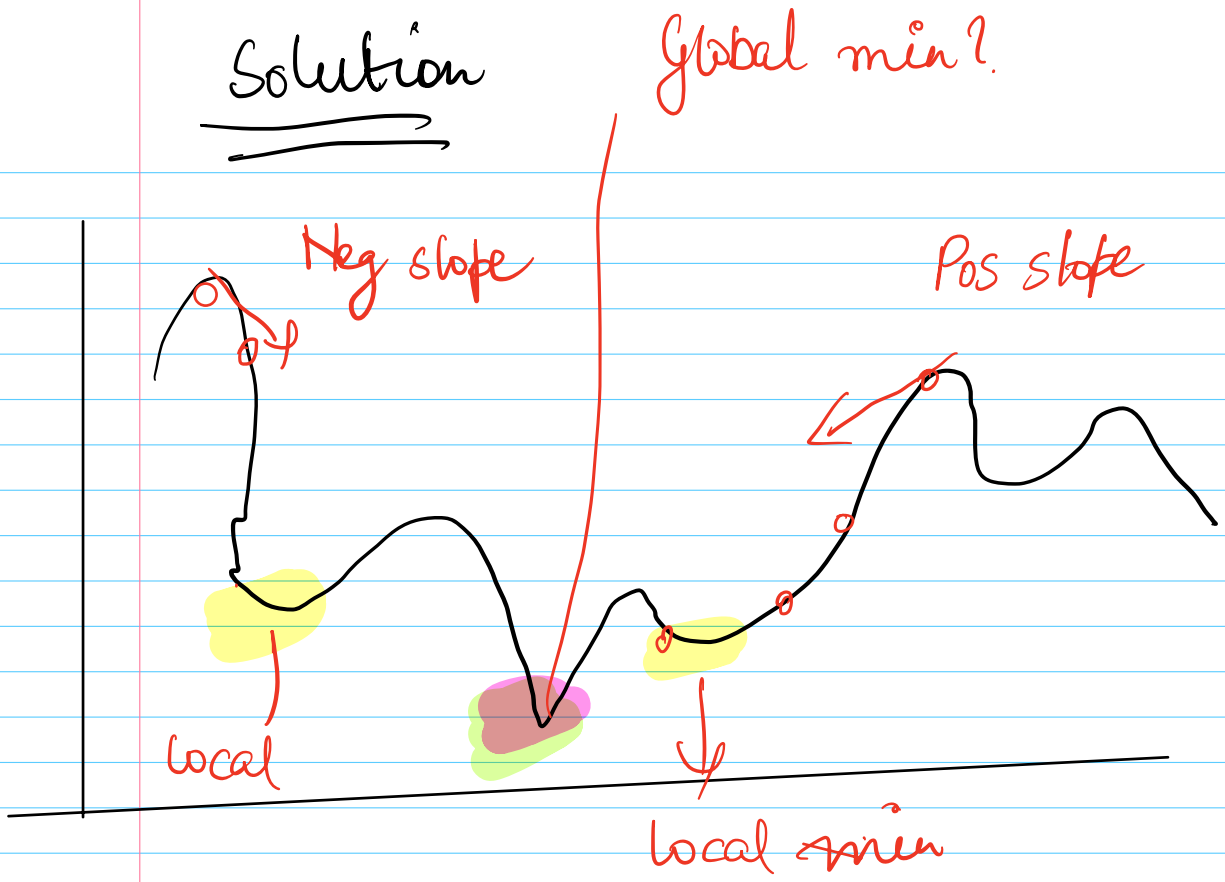
③ Stop when happy or $f(X)$ no longer ↑

② Hmm, ok Problems

→ How to decide which dir^n to go to? in a sea of million params

→ How much to hop, step size?

Solution



① Where to move

-ve derivative → move right

+ve → move left

So, a rough "good" soln

① Init X

② while $f'(x^k) \neq 0$

If sign $(f'(x^k))$ +ve

$$x^{k+1} = x^k - \text{step}$$

else

$$x^{k+1} = x^k + \text{step}$$

② How much to step?

Q.) Select all true about derivatives

→ At any loc X, there may many dir^n in which we can step, such that f(X) ↑

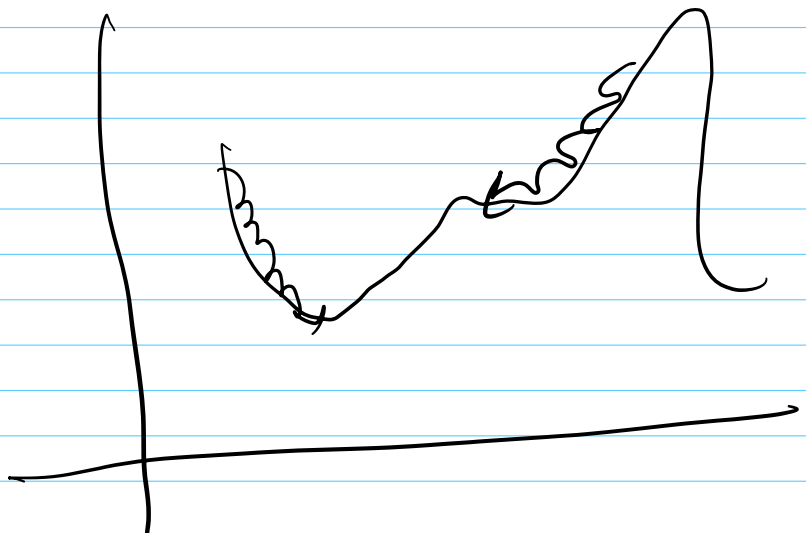
→ Dir^n of gradient is

dirⁿ of steepest / fastest ↑

→ Gradient is derivative

→ Dirⁿ of gradient, f ↑

→ Move opposite, f ↓



→ for convex, appropriate step size will always find min

→ for non-convex it will find local minima / saddle / inflection points

Hmm.. We should use this to find min of our neural network.

Algo

① Init w, k = step

② do $w^{k+1} = w^k - \eta^k \nabla Loss(w^k)^T$

$k = k+1$

while

$$|Loss(W^t) - Loss(W^{t-1})| > \varepsilon$$

Coming back to Our NN

⇒ Our function a NN

⇒ We do init, as discussed above

⇒ We create function which is NN

⇒ We attach loss function

to our function, and finally compose a differentiable func^n ??

why ?? So that we can use iterative solution to find out best params

$$loss = f(Inp, w, outputs)$$

or

$$loss = \frac{1}{T} \sum loss\left(f(X, w), label\right)$$

$f(X, w)$ → N.N

⇒ So we do forward pass → to get this loss

→ Then get gradient

→ Then take a step

⇒ So why we prefer one loss function over another?

⇒ Better loss surface,

the gradient should push us to faster convergence

when prediction deviates from label

⇒ which depends on loss function

Hmm, we are getting closer to solve our bieces !!!

Now problem

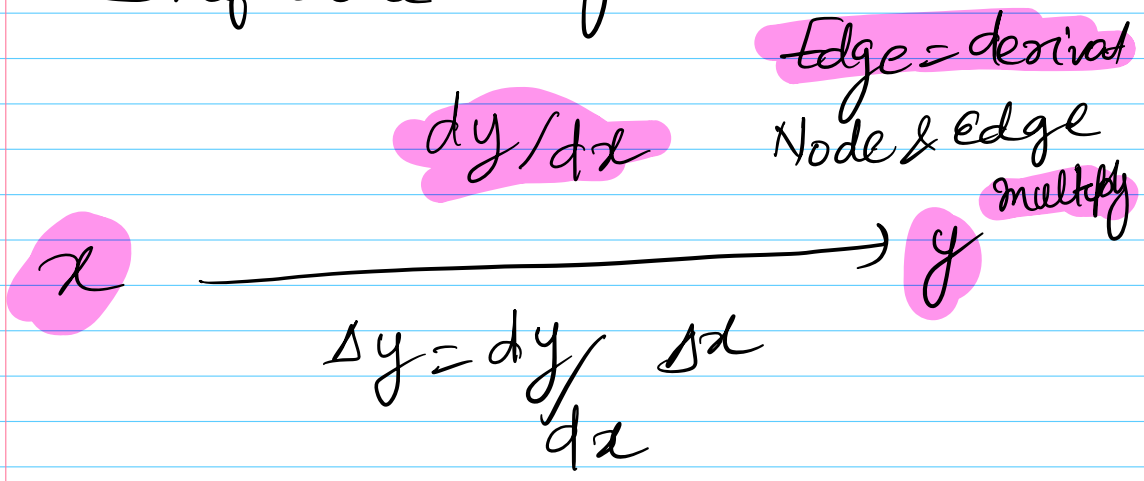
① We must figure out how to compute derivative of loss w.r.t each weight

A Detour

We saw

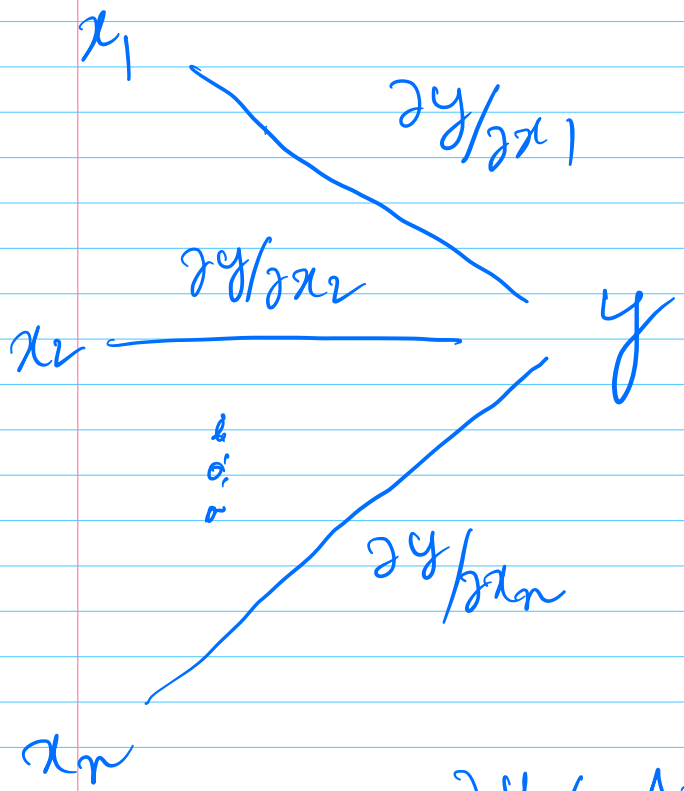
$$\Delta y \approx dy/dx \, \Delta x$$

Influence diagram :



→ what about scalar out, vector in

$$y = f(x_1, x_2, x_3 \ldots x_n)$$

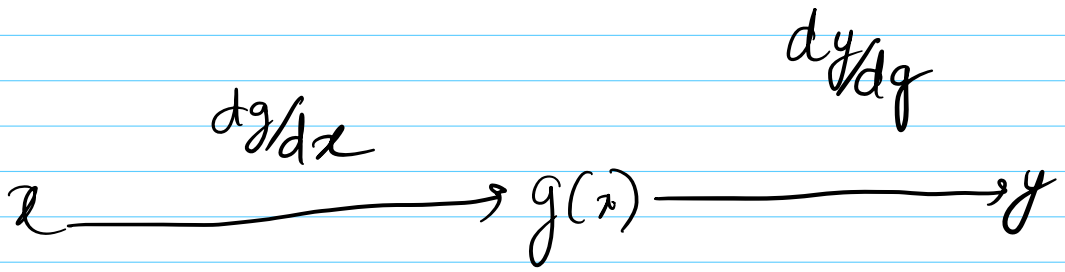


$$\Delta y \approx \frac{\partial y}{\partial x_1} \Delta x_1 + \frac{\partial y}{\partial x_2} \Delta x_2 + \vdots$$

$\partial y / \partial x_m \quad \Delta x_m$

What about

$$y = f(g(x))$$

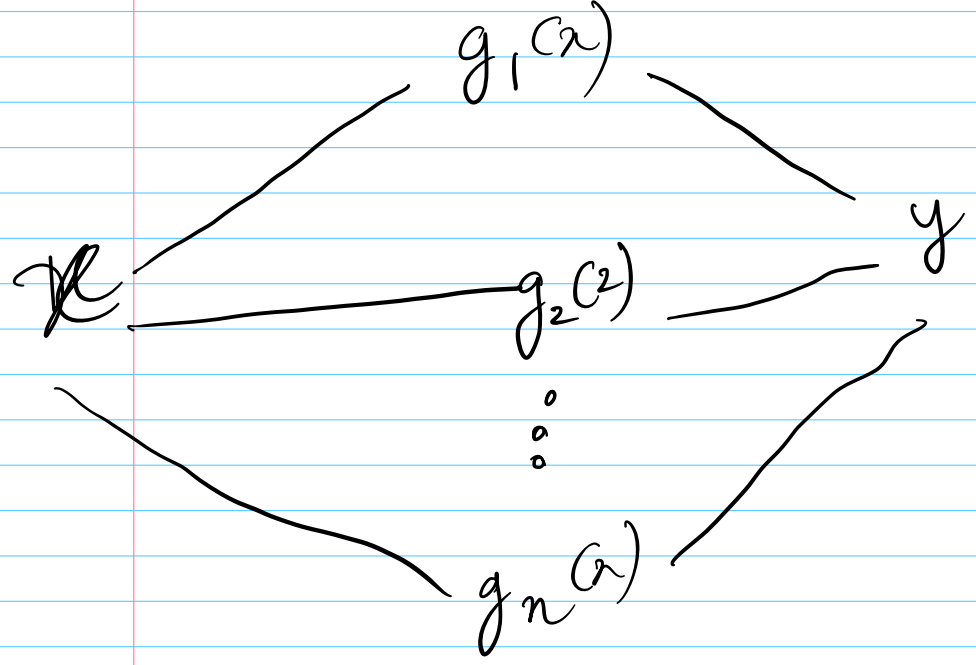


$$\Delta y = \frac{dy}{dg} \Delta g; \quad \Delta g = \frac{dg}{dx} \Delta x$$

$$\Delta y = \frac{dy}{dg} \frac{dg}{dx} \Delta x$$

What about

$$y = f(g_1(x), g_2(x) \ldots)$$



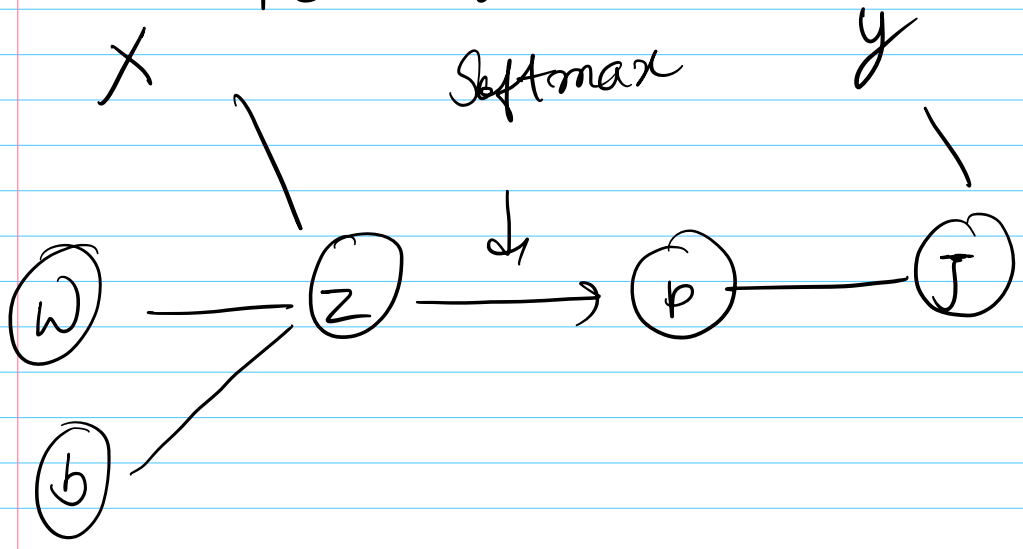
$$\Delta y = \frac{dy}{dg_1}\frac{dg_1}{dx} + \frac{dy}{dg_2}\frac{dg_2}{dx} + \ldots$$

↑

Chain Rule

Coming back

lets take our Neural Network



$$\text{loss} = \sum_{jp} y_i \log p$$

∃ forward pass is required to compute derivative,

$$\frac{d\,\text{loss}(Y, P)}{1 \cdot R}$$

$d\,w_{i,j}$

which requires intermediate and final output

$\Rightarrow$ So using chain rule,

$$\underset{[1 \times n_c]}{\frac{\partial J}{\partial w}} = \frac{\partial J}{\partial p} \underset{n_c \times n_c}{\times} \frac{\partial p}{\partial z} \times \frac{\partial z}{\partial w}$$

for now, without going into math, let me tell you that

$$\frac{\partial z}{\partial w} = \frac{\partial (w^T x)}{\partial w} = x$$

$$\frac{\partial J}{\partial p} \times \frac{\partial p}{\partial z} = (p_k - y_i) \quad dz$$

Eg: $y_i = 2$ $[0, 1, 0]$

$p = [0.2, 0.3, 0.5)$

$[0.2-0, 0.3-1, 0.5-0]$

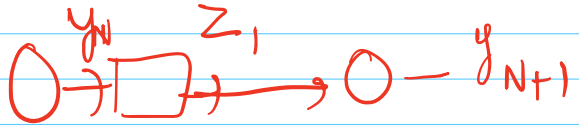
$= [0.2, -0.7, 0.5]$

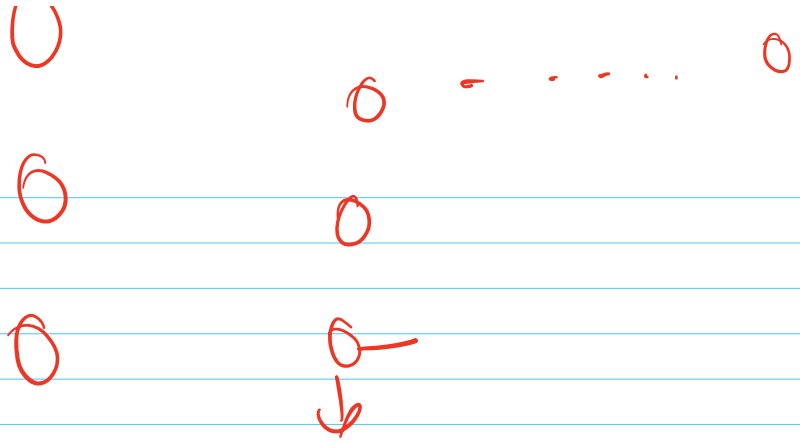
Refer to Notebook

$$\frac{\partial Div}{\partial w_{ij}} = \sum \frac{d Div}{\partial z_j} \frac{\partial z}{\partial y_i}$$

$$\frac{\partial Div}{\partial w_{ij}}^{(N)} = y_i^{(N-1)} \frac{\partial div}{\partial z_j}^{(N)}$$

for any inside layer



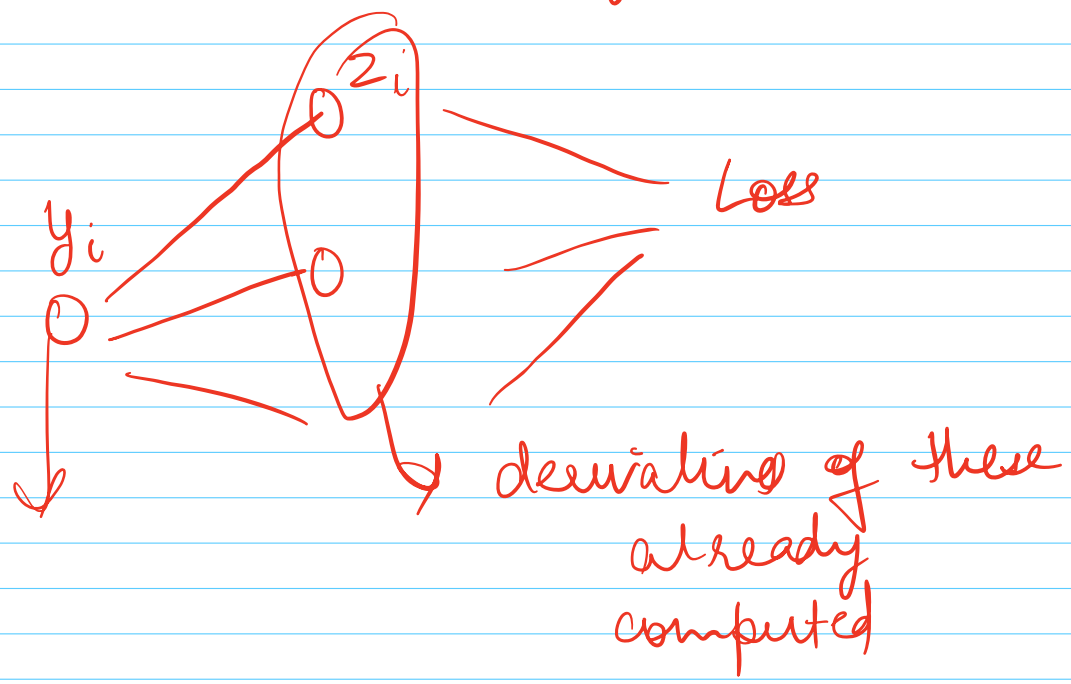


If we do back then derivative

$$\frac{\partial \text{Loss}}{\partial z_i^{(N+1)}}$$

has already been computed

which is simply



So