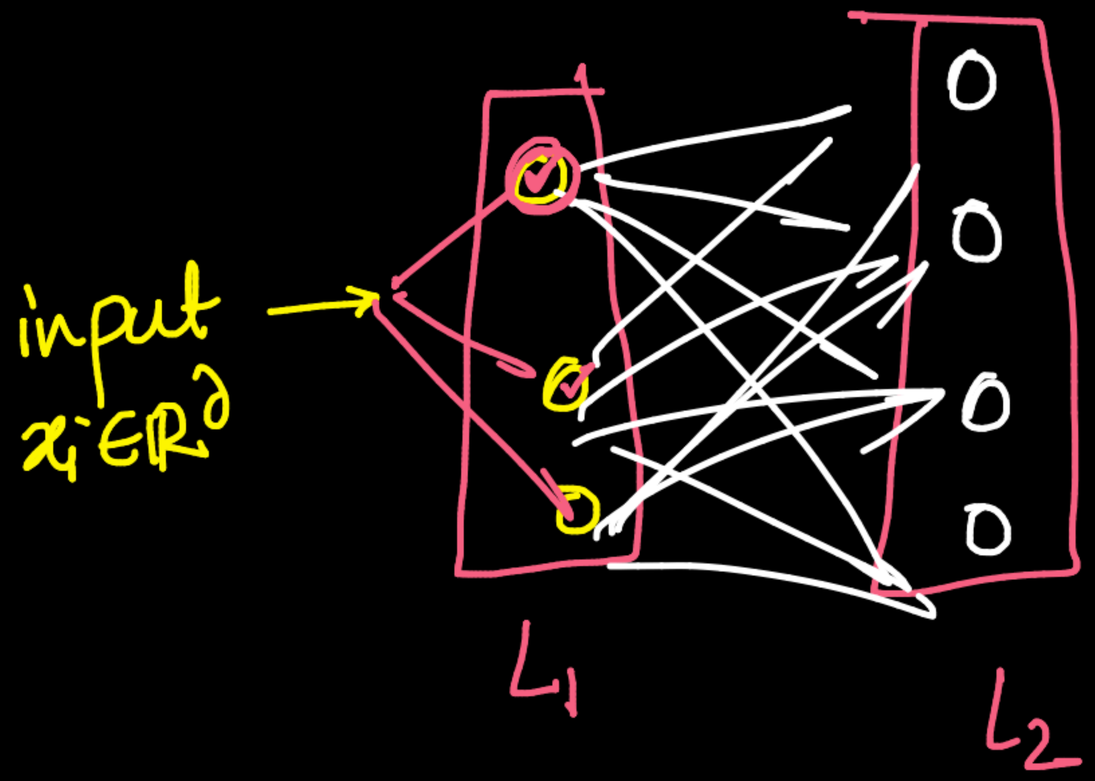
[✓ Back-propogation → 1980's (Hinton & others)
↗ Understand + code]

(How MLPs are trained)

↳ Most of DL-NN

{ → CV
→ NLP/text : speech ...

Q



[possible]

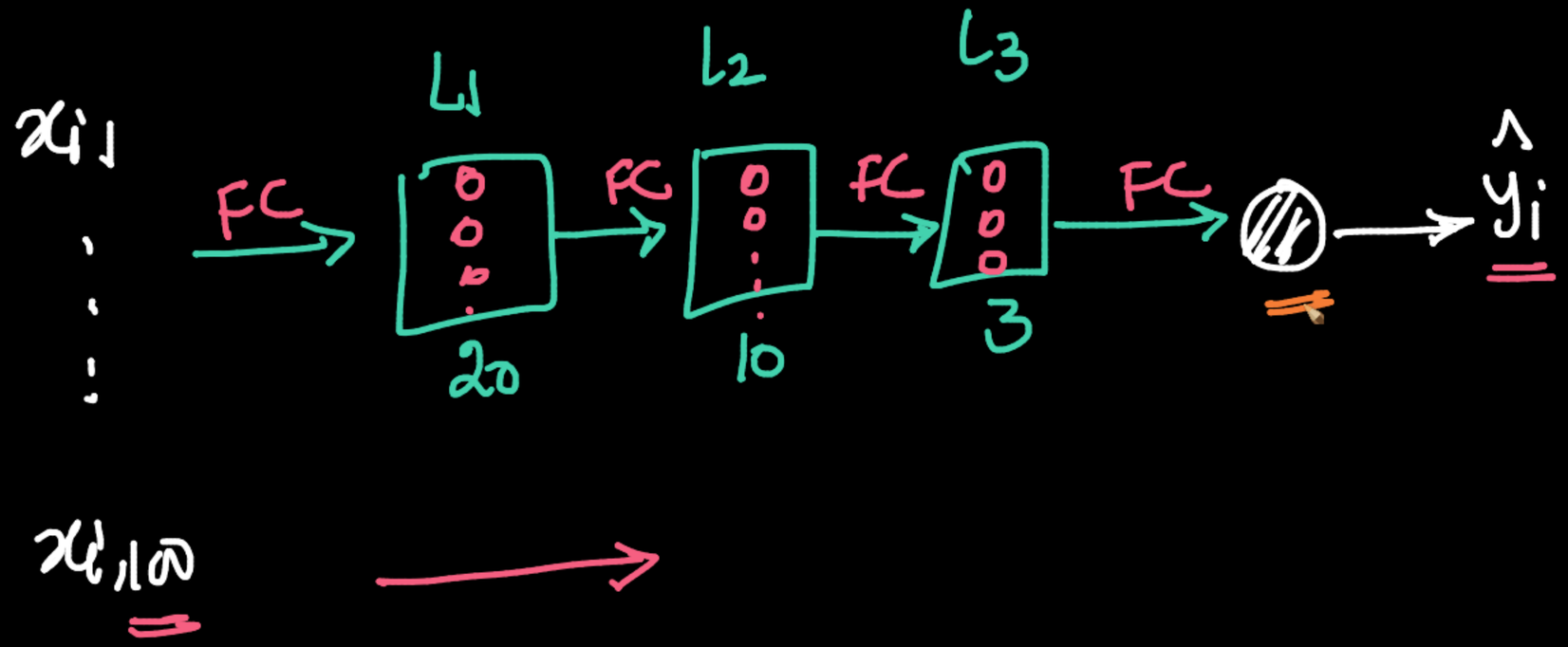
U-Net (CV)

$y \{ \overline{f_\theta()}$

{ # neurons in each layer
{ # layers

$$\begin{cases} f_{11} : f_0 \\ f_{12} : \text{anolkūn aelvūn} \end{cases}$$

$\Rightarrow$ tons of More hyper-param


L_1
L_2
L_3
$x_{i,1}$
$x_{i,100}$
FC
FC
FC
FC
20
10
3
$\hat{y}_i$

Recap:

1. MLP ≈ function composition
2. Activation fns have to be non-linear
3. Logistic reg r & Lr-SVM → Single Neuron model
4. Brute force: Compute $\frac{\partial Loss}{\partial w_j}$ $\forall$ weight j + GD

Recap:

✓ (1) MLP ≈ function composition $f \circ g : g \circ f$

✓ (2) Activation fns have to be non-linear

(3) Logistic reg r & Lr-SVM → Single Neuron model

(4) Brute force : Compute $\frac{\partial Loss}{\partial w_{ij}^{k}}$ $\forall weight_{ij}^{k}$

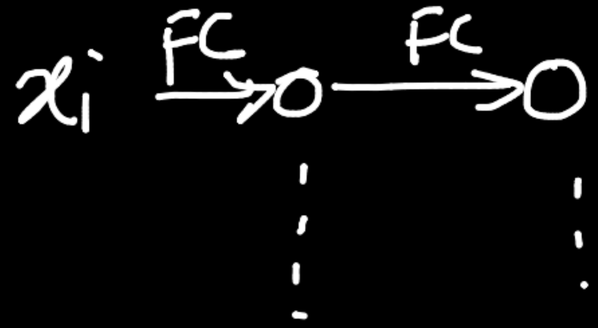
+ GD

w_{ij}^{k}

Q

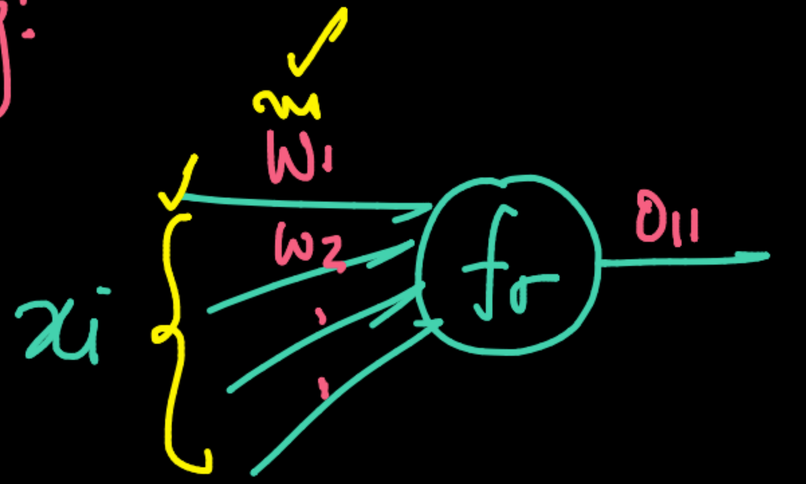
why should all activn fns be non-linear?

$$\alpha' \left(\alpha \left(\underbrace{W^T x_i}_{\text{lin}} + b \right) + \beta \right) + \beta'$$



$$f \circ g(x) = f_L(g_L(x)) = h_L(x)$$

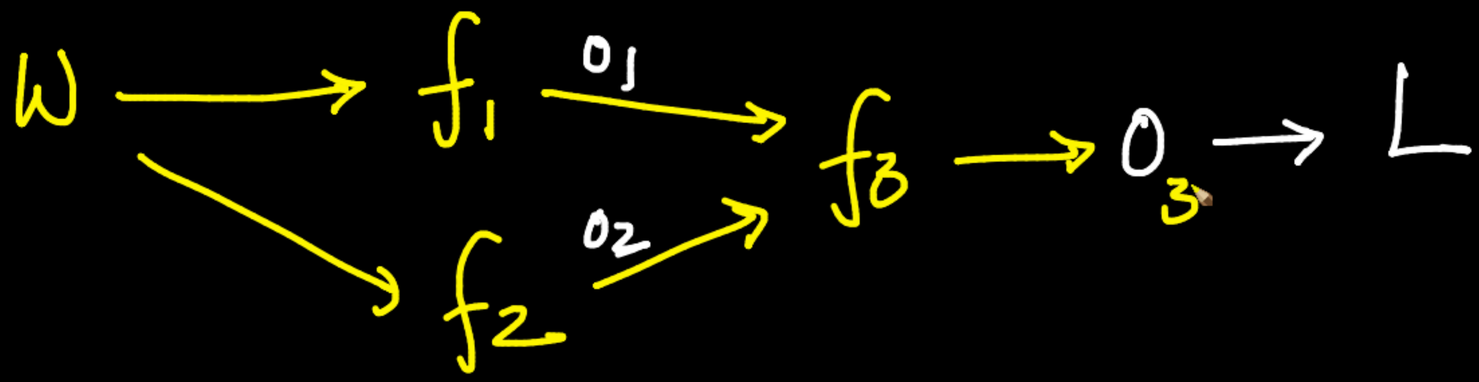
logistic reg:



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w_1}$$

$\frac{\partial L}{\partial o_{11}} \rightarrow \sum_{i=1}^{n}$

e.g.:

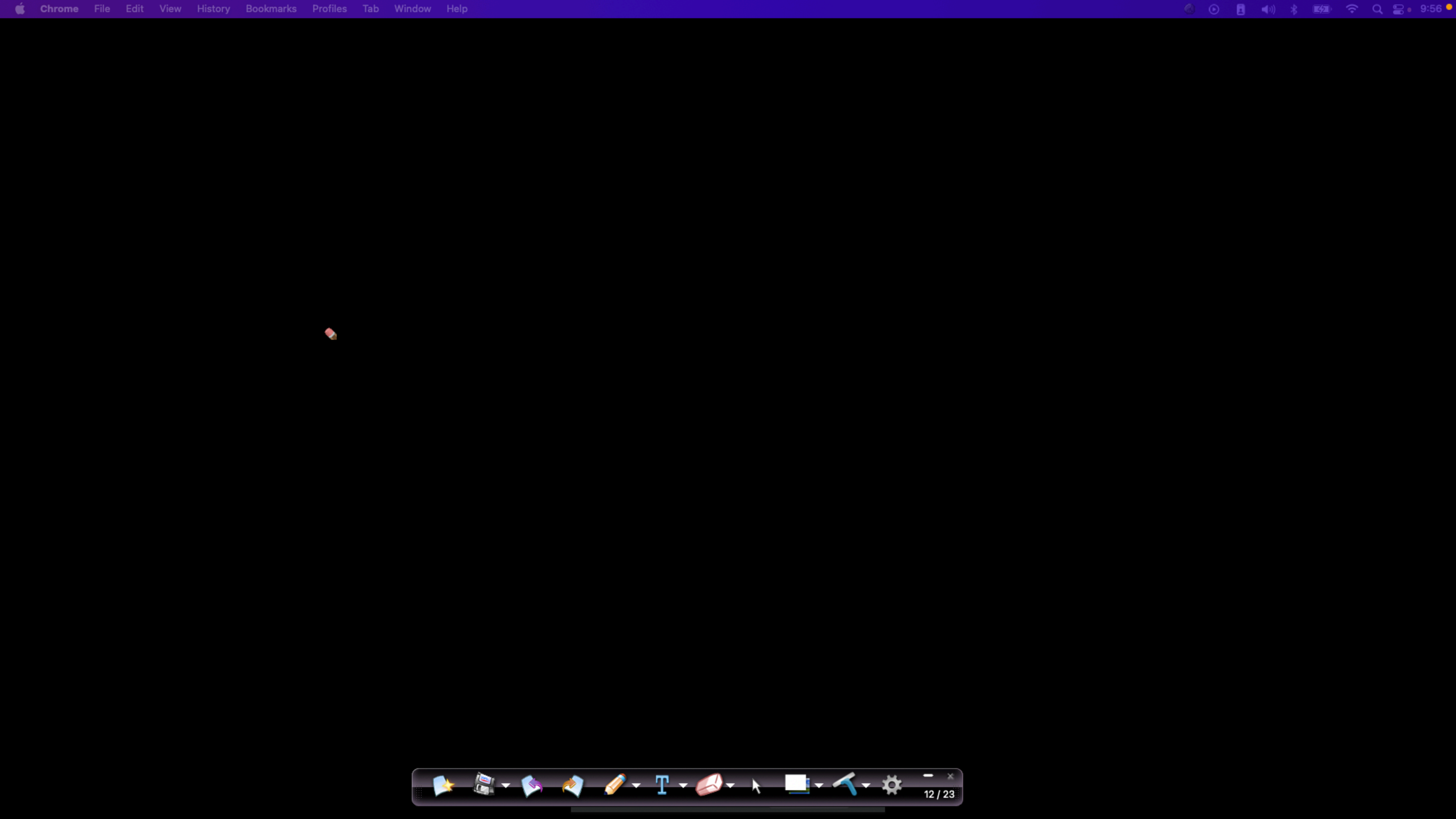


$$Loss = L(y, f_3(o_1, o_2))$$

$$= L(y, f_3(f_1(w), f_2(w)))$$

$$\text{Loss} = L\left(y,\ f_3\left(\underbrace{f_1(w)}_{o_1},\ \underbrace{f_2(w)}_{o_2}\right)\right)$$

o_3


Chrome File Edit View History Bookmarks Profiles Tab Window Help
9:56
12 / 23

Obj:

$$\frac{\partial L}{\partial w^1_{11}} = ?$$

$$\frac{\partial L}{\partial w^3_{11}} = \frac{\partial L}{\partial o_{31}} \frac{\partial o_{31}}{\partial w^3_{11}}$$

$$\frac{\partial L}{\partial w^2_{11}} = \frac{\partial L}{\partial o_{31}} \cdot \frac{\partial o_{31}}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial w^2_{11}}$$

Chain-rule!

One-path

$$\frac{\partial L}{\partial o_{31}} \cdot \frac{\partial o_{31}}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w'_{11}}$$

(4) $\forall i,j,k$

$$w^k_{ij}\Big|_{new} = w^k_{ij}\Big|_{old} - \eta \frac{\partial L}{\partial w^k_{ij}}\Big|_{w^k_{ij}}$$

$$b^k_{ij}\Big|_{new} = b^k_{ij}\Big|_{old} - \eta \frac{\partial L}{\partial b^k_{ij}}\Big|_{b^k_{ij}}$$

↳ back-prop

n = 100 Million

d = 1000 - dim

GD: $\frac{\partial L}{\partial w_{ij}^{k}}$ using all n points

SGD: $\frac{\partial L}{\partial w_{ij}^{k}}$ using one random pt

✓ batch-GD: $\frac{\partial L}{\partial w_{ij}^{k}}$ using a random of m pts

Challenge: $\frac{\partial L}{\partial w^k_{ij}}$

$$\frac{\partial L}{\partial w^1_{11}} =$$

Challenge: $\frac{\partial L}{\partial w^k_{ij}}$

$$\frac{\partial L}{\partial w^1_{11}} =$$

(4) $\forall i,j,k$

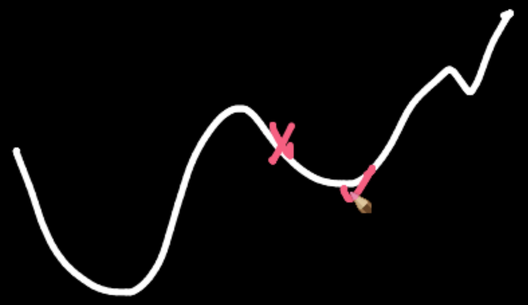
$$w^k_{ij}\Big|_{new} = w^k_{ij}\Big|_{old} - \eta \left.\frac{\partial L}{\partial w^k_{ij}}\right|_{w^k_{ij}\,old}$$

$$b^k_{ij}\Big|_{new} = b^k_{ij}\Big|_{old} - \eta \left.\frac{\partial L}{\partial b^k_{ij}}\right|_{b^k_{ij}\,old}$$

↳ back-prop

(5) Repeat 2, 3 & 4 till convergence

$$\begin{cases} w^k_{ij\,new} \simeq w^k_{ij\,old} \\ b^k_{ij\,new} \simeq b^k_{ij\,old} \end{cases}$$



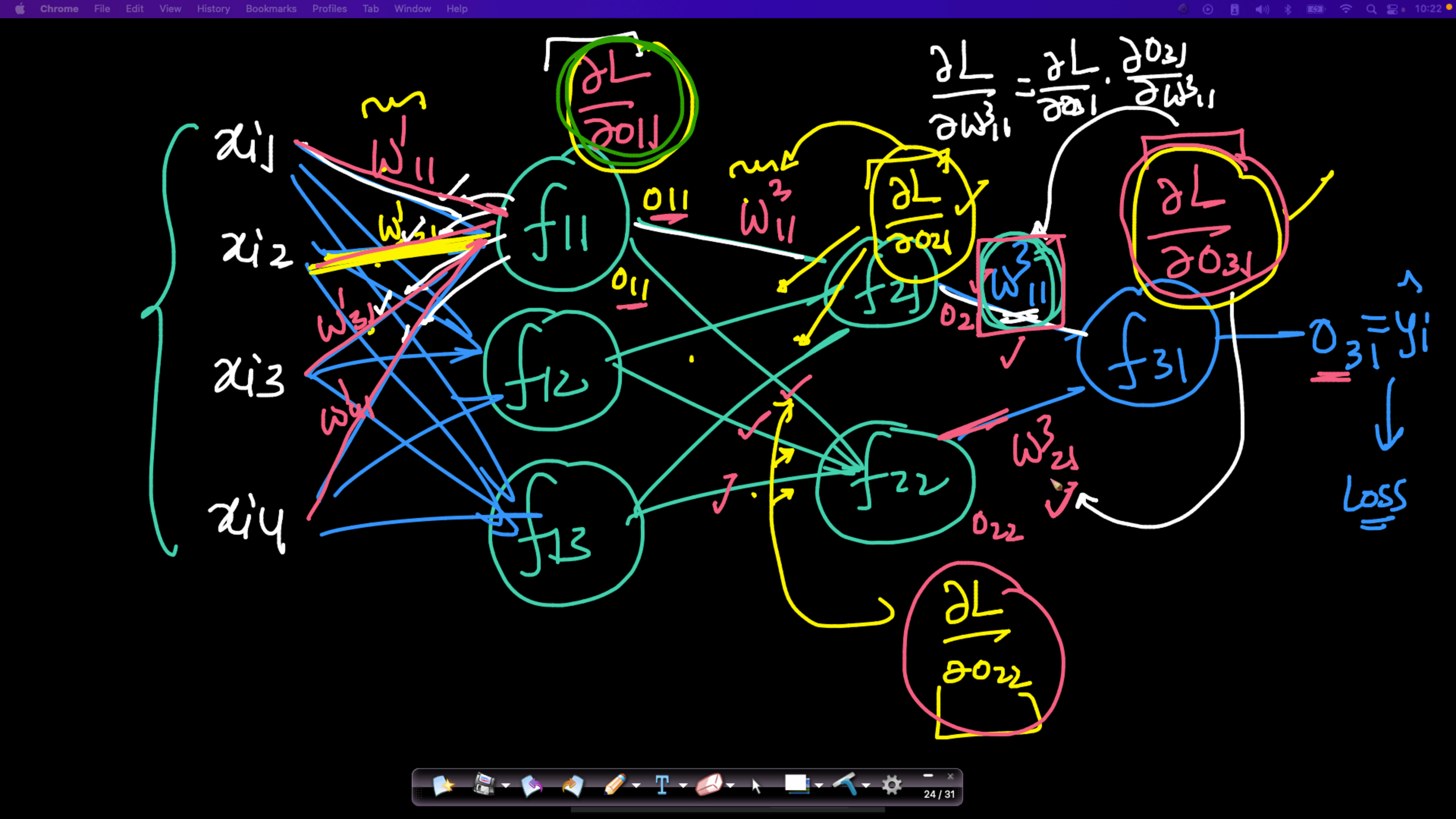
Challenge: $\frac{\partial L}{\partial w^k_{ij}}$

$\frac{\partial L}{\partial w^1_{11}}$: backwards (using chain-rule)

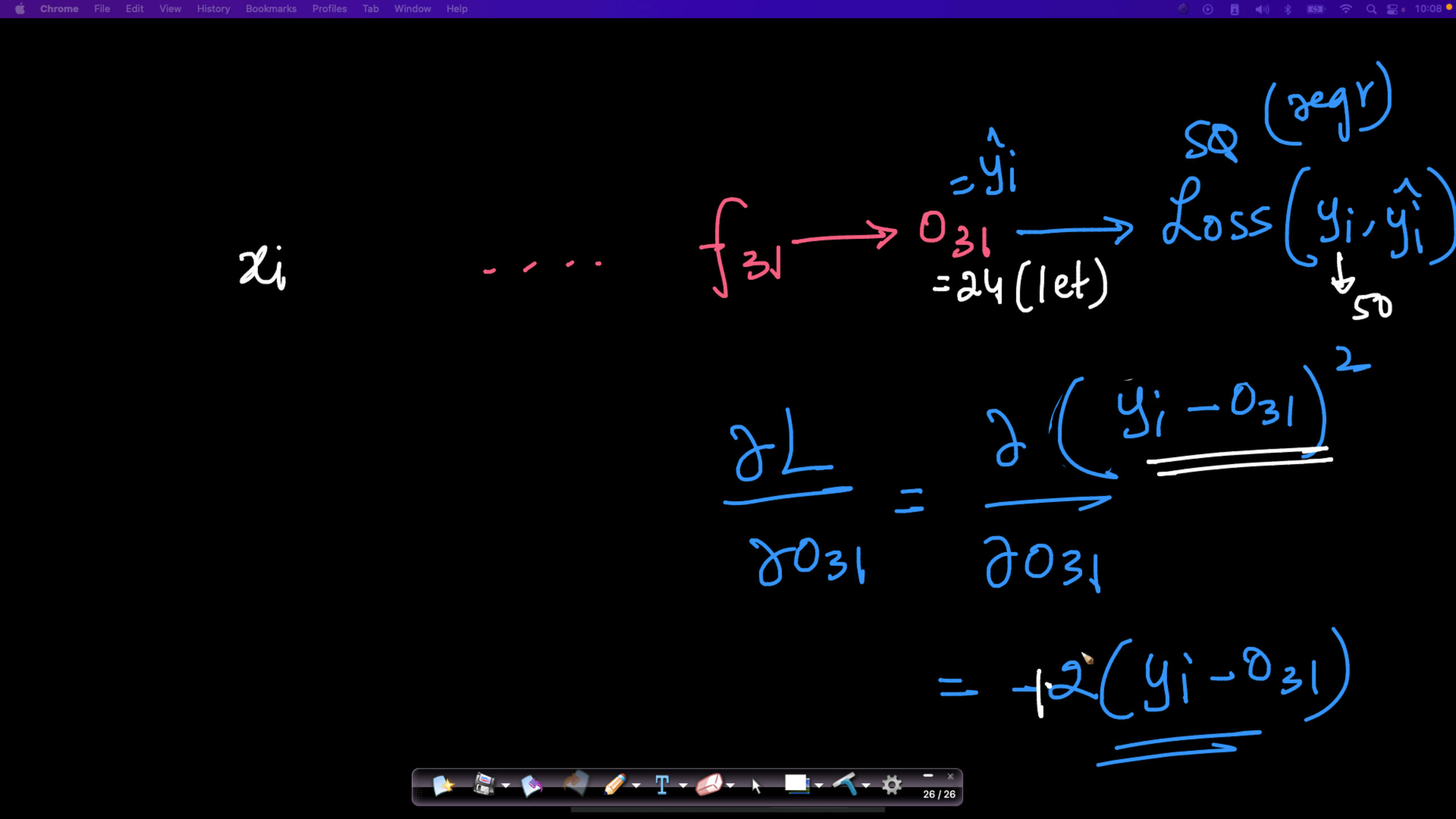
from loss till

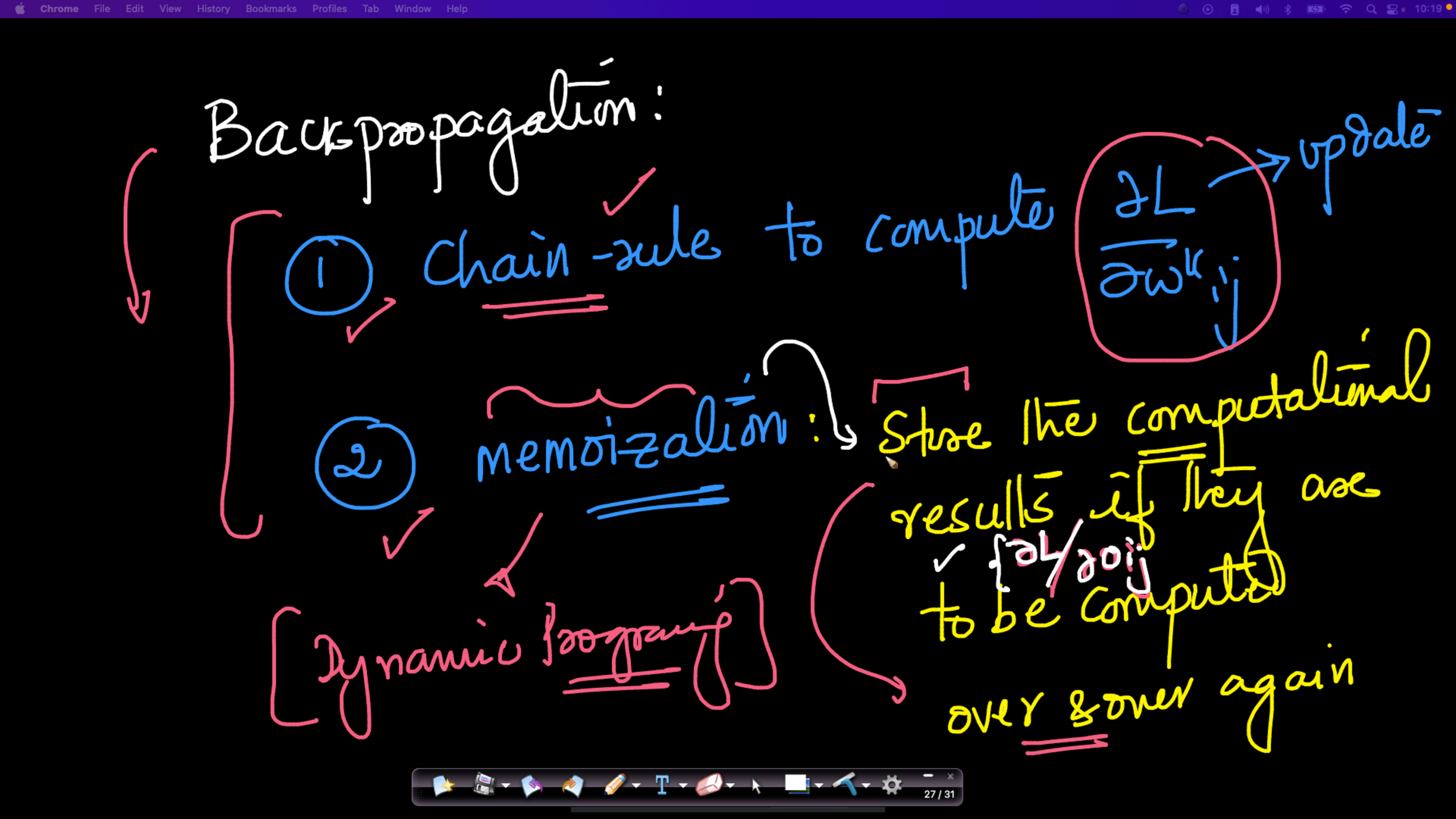
first layer

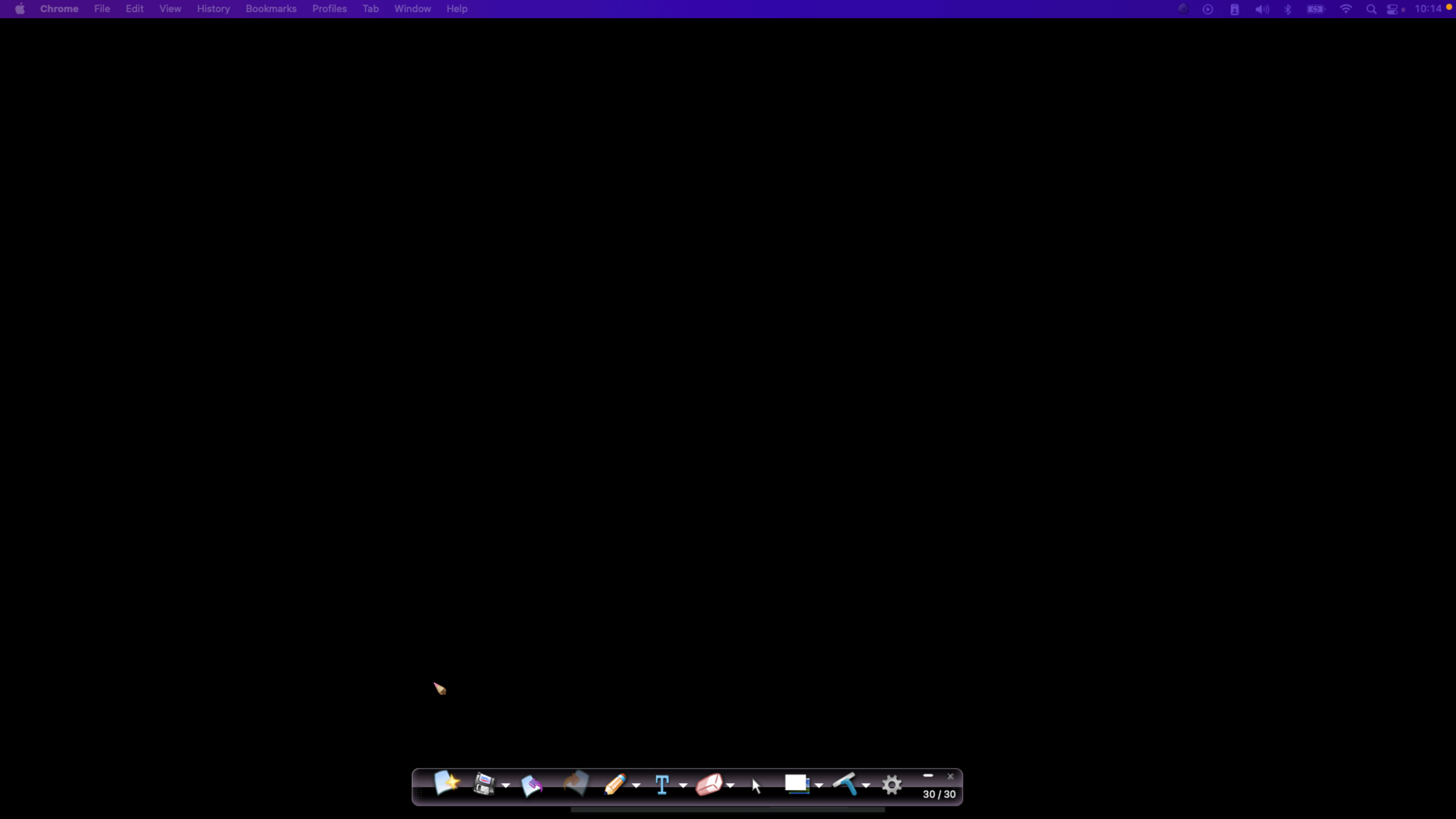
input → (f) → output

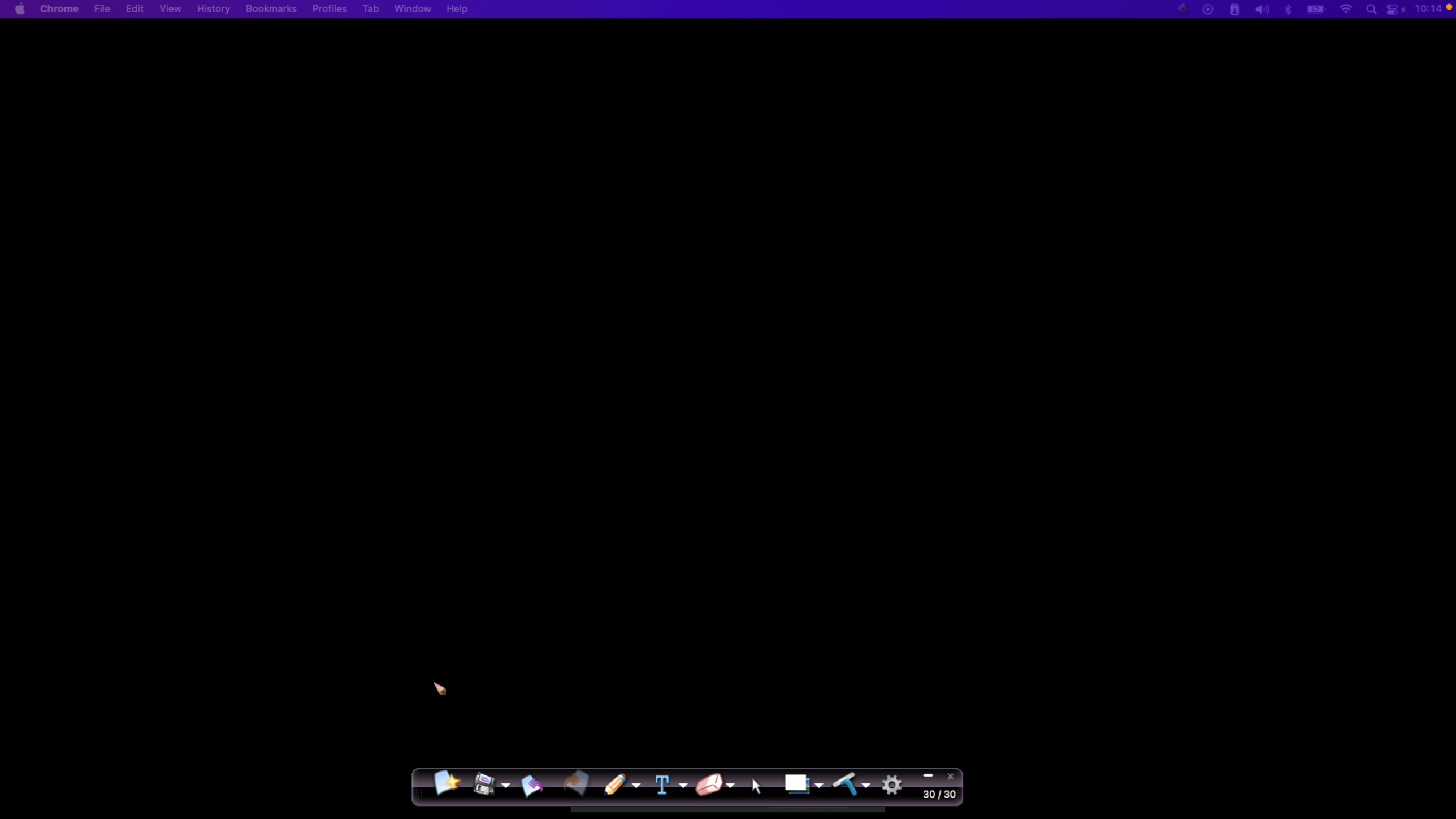

$\frac{\partial L}{\partial w^3_{11}} = \frac{\partial L}{\partial o_{31}} \cdot \frac{\partial o_{31}}{\partial w^3_{11}}$
x_{i1}
x_{i2}
x_{i3}
x_{i4}
w^1_{11}
w^1_{21}
w^1_{31}
w^1_{41}
f_{11}
f_{12}
f_{13}
o_{11}
o_{11}
w^2_{11}
f_{21}
f_{22}
o_{22}
o_{21}
w^3_{11}
w^3_{21}
f_{31}
$o_{31} = \hat{y}_i$
Loss
$\frac{\partial L}{\partial o_{11}}$
$\frac{\partial L}{\partial o_{21}}$
$\frac{\partial L}{\partial o_{31}}$
$\frac{\partial L}{\partial o_{22}}$
24 / 31

$$\begin{cases} w'_{11}: & \dfrac{\partial L}{\partial w'_{11}} = \dfrac{\partial L}{\partial o_{11}} \cdot \dfrac{\partial o_{11}}{\partial w'_{11}} \\ w'_{21}: & \dfrac{\partial L}{\partial w'_{21}} = \dfrac{\partial L}{\partial o_{11}} \cdot \dfrac{\partial o_{11}}{\partial w'_{21}} \end{cases}$$


Chrome File Edit View History Bookmarks Profiles Tab Window Help
10:08
x_i ..... f_{31} → o_{31} = 24 (let) = $\hat{y}_i$ → SQ Loss $(y_i, \hat{y}_i)$ (regr)
50
$\frac{\partial L}{\partial o_{31}} = \frac{\partial (y_i - o_{31})^2}{\partial o_{31}}$
$= +2(y_i - o_{31})$
26 / 26


Backpropagation:
1 Chain-rule to compute ∂L/∂w^k_{i,j} → update
2 memoization: → Store the computational results if they are to be computed over & over again
Dynamic Programming

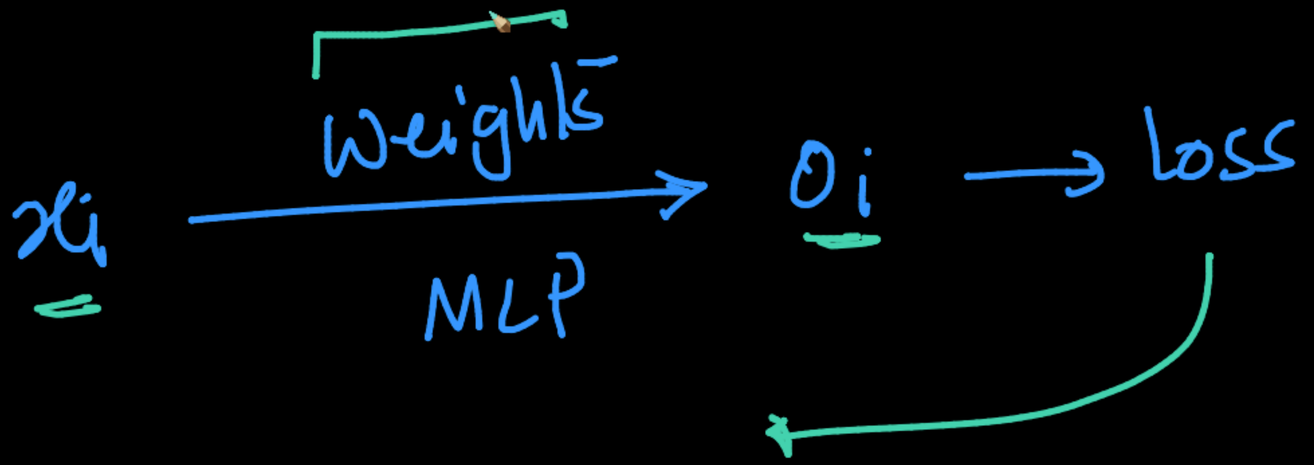

Chrome File Edit View History Bookmarks Profiles Tab Window Help
10:14
30 / 30


Chrome File Edit View History Bookmarks Profiles Tab Window Help
10:14
30 / 30

(e.g)

$$o = w_1 x_1 + w_2 x_2$$

$$\frac{\partial o}{\partial w_1} = x_1 \qquad \frac{\partial o}{\partial w_2} = x_2$$


weights
x_i
MLP
o_i
loss

Back-prop

Given: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$

Task: w_{ij}^{k}

1. initialize w_{ij}^{k} & $\eta \rightarrow$ learning rate (randomly)

(2) Forward loop (batch-SGD)

(3) Loss

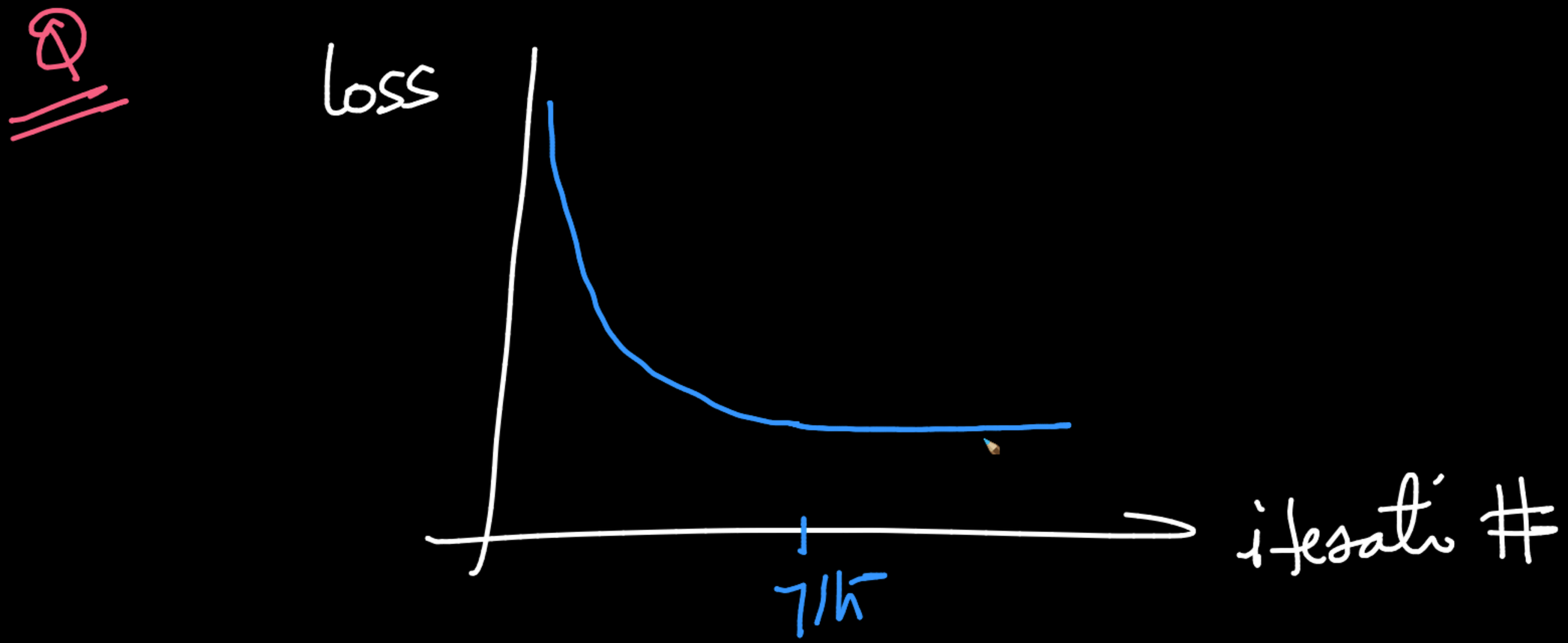
(4) Update weights backwards (o/p-input)

↳ Keep this $\frac{\partial L}{\partial o_{ij}}$ in memory & reuse → chain rule

(5) Repeat 2, 3, 4 till convergence

1. activation fns & loss fns should be differentiable

2. slow ← simple back-prop (later)



Q

Mini-batch SGD

100 items → (w_{ij}^{k}) → Disk

100 iter

Activation:

req: non-linear fn:

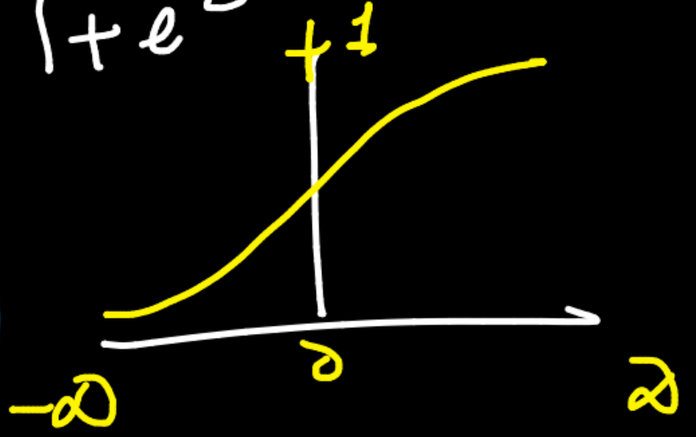
$$w^T x + b = z$$

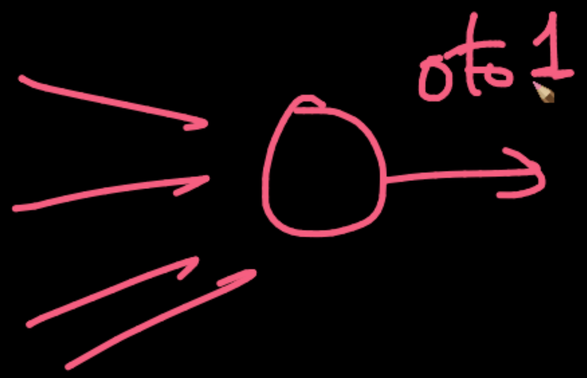
1980's & 90's:

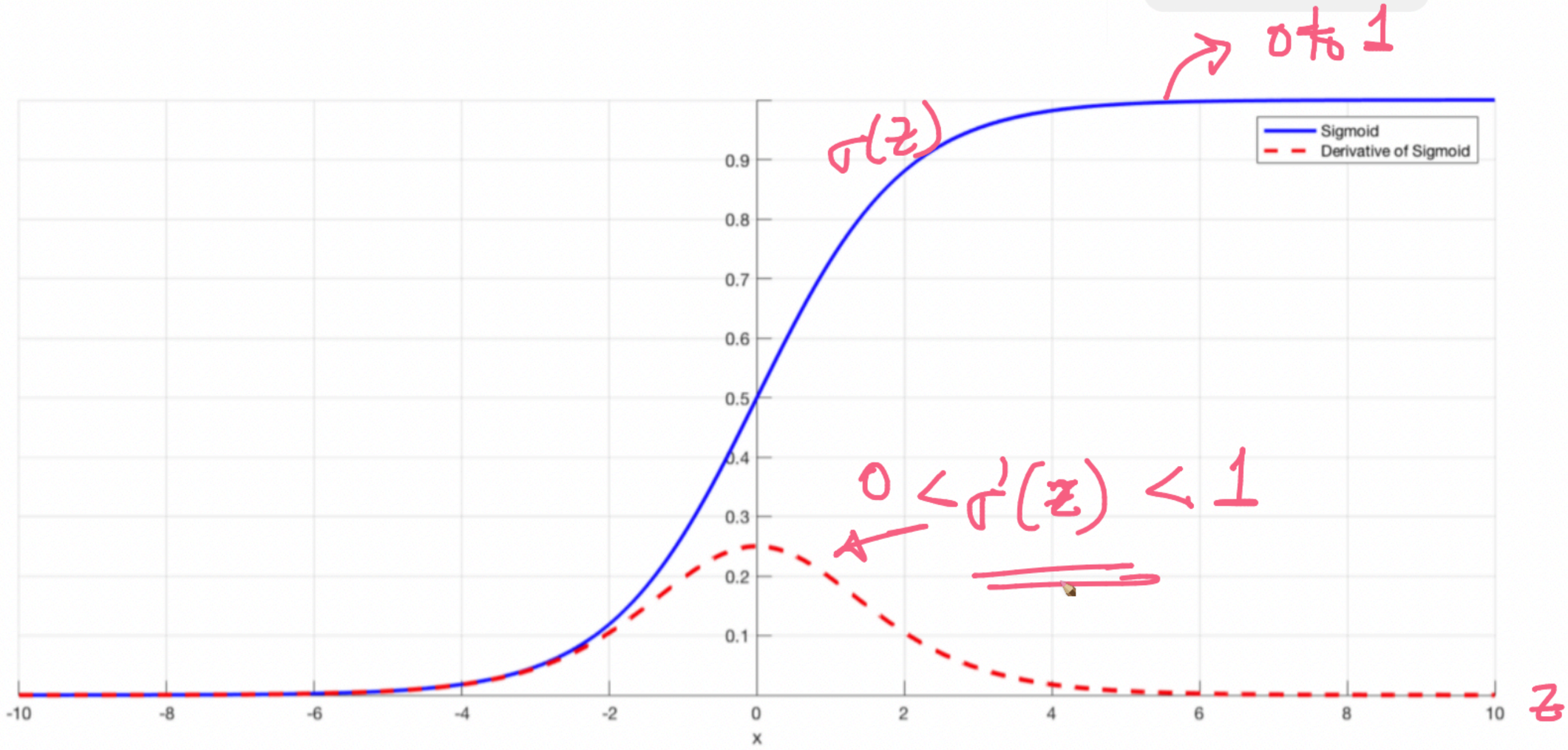
Sigmoid ← logistic regn..

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z}$$

$$\sigma'(z) = \frac{\partial \sigma}{\partial z} = \sigma(z)\,(1-\sigma(z))$$




0 to 1



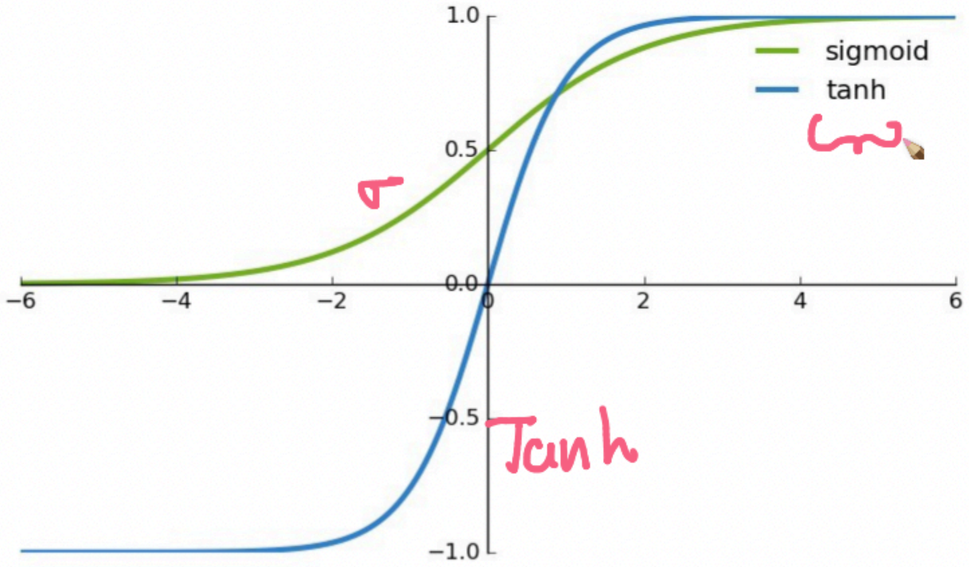

Tan h

2)tanh or Hyperbolic:

The tanh function is just another possible functions that can be used as a nonlinear activation function between layers of a neural network. It actually shares a few things in common with the sigmoid activation function. They both look very similar. But while a sigmoid function will map input values to be between 0 and 1, Tanh will map values to be between -1 and 1.

You wil

Like th
is that t
actual f
derivat



$y_i \in \{0, 1\}$

$y_i \in \{-1, 1\}$

Derivative of tanh(z):

a=(e^z-e^(-z))/(e^z+e^(-z))

use same

Like the sigmoid function, one of the interesting properties of the tanh function is that the derivative can be expressed in terms of the function itself. Below is the actual formula for the tanh function along with the formula for calculating its derivative.

$$a = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$

$$\frac{da}{dz} = 1 - a^2$$

Derivative of tanh(z):

a=(e^z-e^(-z))/(e^z+e^(-z)

use same u/v rule

da=[(e^z+e^(-z))*d(e^z-e^(-z))]-[(e^z-e^(-z))*d((e^z+e^(-z))]/[(e^z+e^(-z)]^2

da=[(e^z+e^(-z))*(e^z+e^(-z))]-[(e^z-e^(-z))*(e^z-e^(-z))]/[(e^z+e^(-z)]^2

da=[(e^z+e^(-z)]^2-[(e^z-e^(-z)]^2/[(e^z+e^(-z)]^2

da=1-[(e^

Like the sigmoid function, one of the interesting properties of the tanh function is that the derivative can be expressed in terms of the function itself. Below is the actual formula for the tanh function along with the formula for calculating its derivative.

$$a = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$\frac{da}{dz} = 1 - a^2$$

Derivative of tanh(z):

a=(e^z-e^(-z))/(e^z+e^(-z)

use same u/v rule

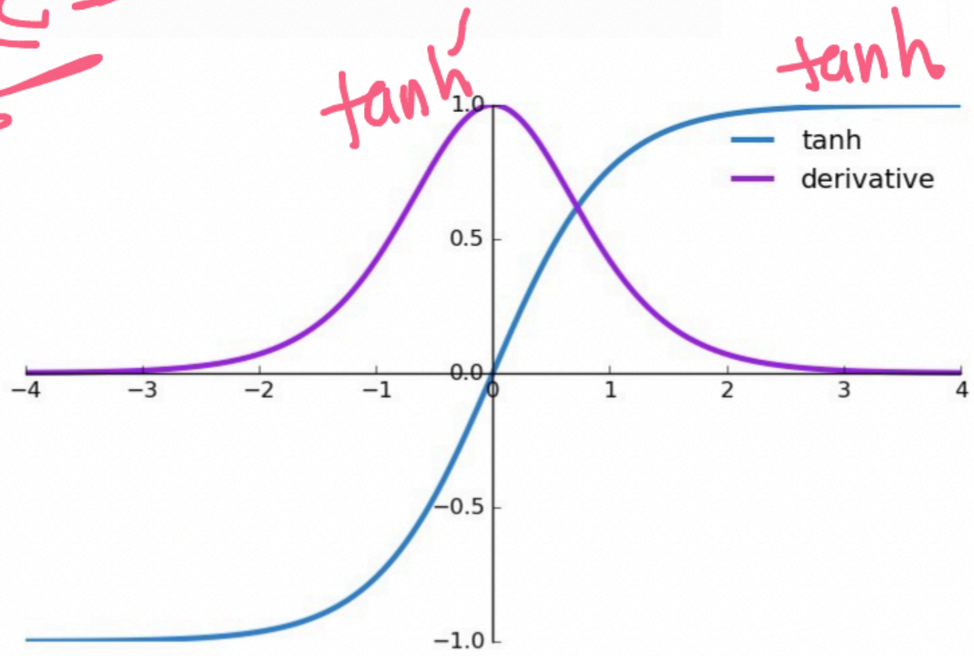
da=[(e^z+e^(-z))*d(e^z-e^(-z))]-[(e^z-e^(-z))*d((e^z+e^(-z))]/[(e^z+e^(-z)]^2

da=[(e^z+e^(-z))*(e^z+e^(-z))]-[(e^z-e^(-z))*(e^z-e^(-z))]/[(e^z+e^(-z)]^2

da=[(e^z+e^(-z)]^2-[(e^z-e^(-z)]^2/[(e^z+e^(-z)]^2

da=1-[(e^

$0 < \tanh'(z) \leq 1$

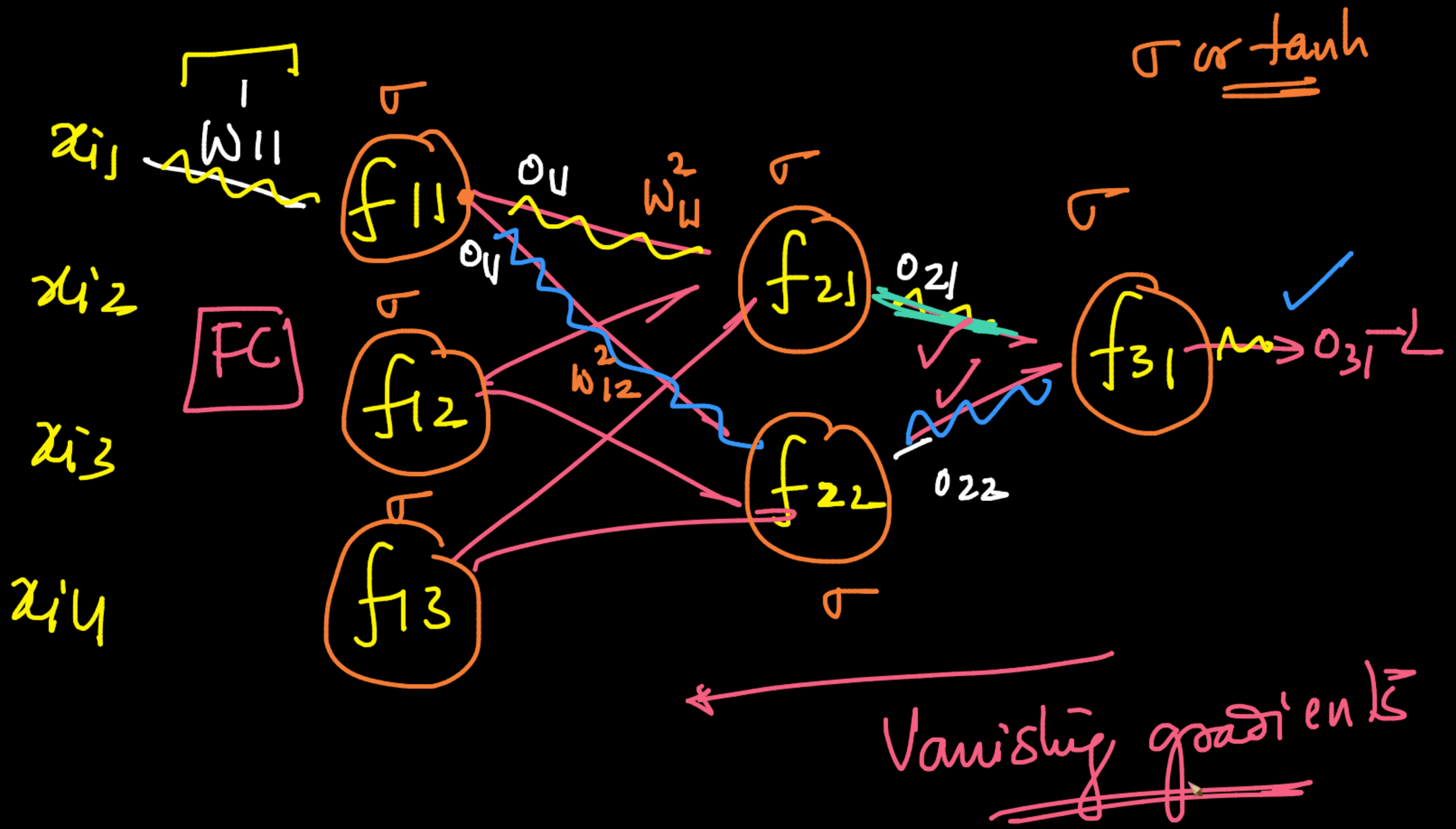


$$\frac{\partial \tanh(z)}{\partial z} = 1 - \tanh^2(z)$$

Traditionally:

Sigmoid : $\sigma(z)$; $0 < \sigma'(z) < 1$

Tanh(z): $0 < \tanh'(z) \leq 1$



σ or tanh

$$\frac{\partial L}{\partial w'_{11}} = \frac{\partial L}{\partial o_{31}} \cdot \left\{ \frac{\partial o_{31}}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w'_{11}} \right.$$

$\frac{\partial o_{21}}{\partial o_{11}} \to \frac{\partial f_{21}}{\partial o_{11}}$

o_{31} to o_{21} to 1 · o_{21} to o_{11} to 1 · o_{11} to w'_{11} to 1

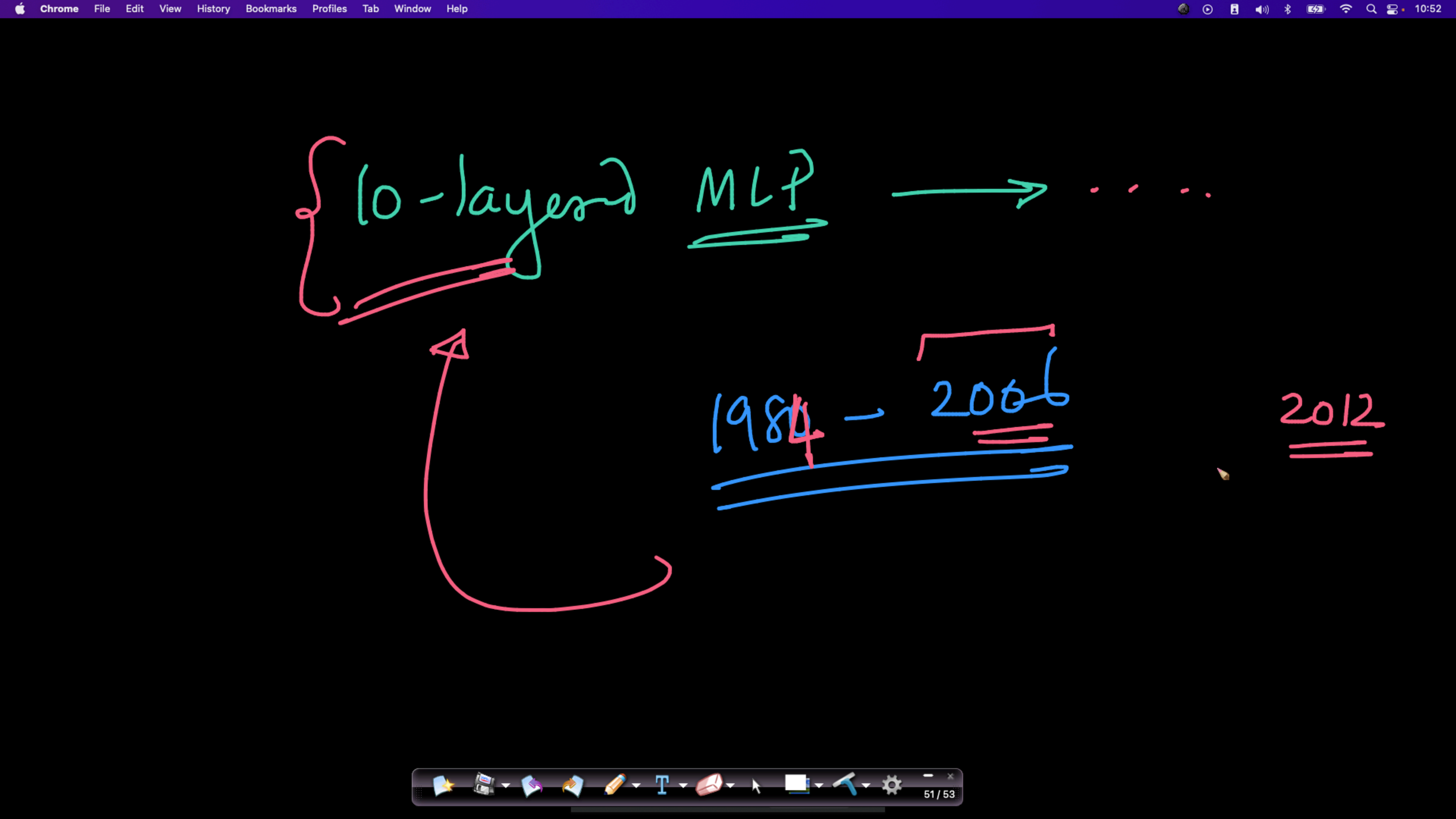
$$+ \frac{\partial o_{31}}{\partial o_{22}} \cdot \frac{\partial o_{22}}{\partial o_{11}} \cdot \left. \frac{\partial o_{11}}{\partial w'_{11}} \right]$$

o_{31} to o_{22} to 1 · o_{22} to o_{11} to 1 · o_{11} to w'_{11} to 1

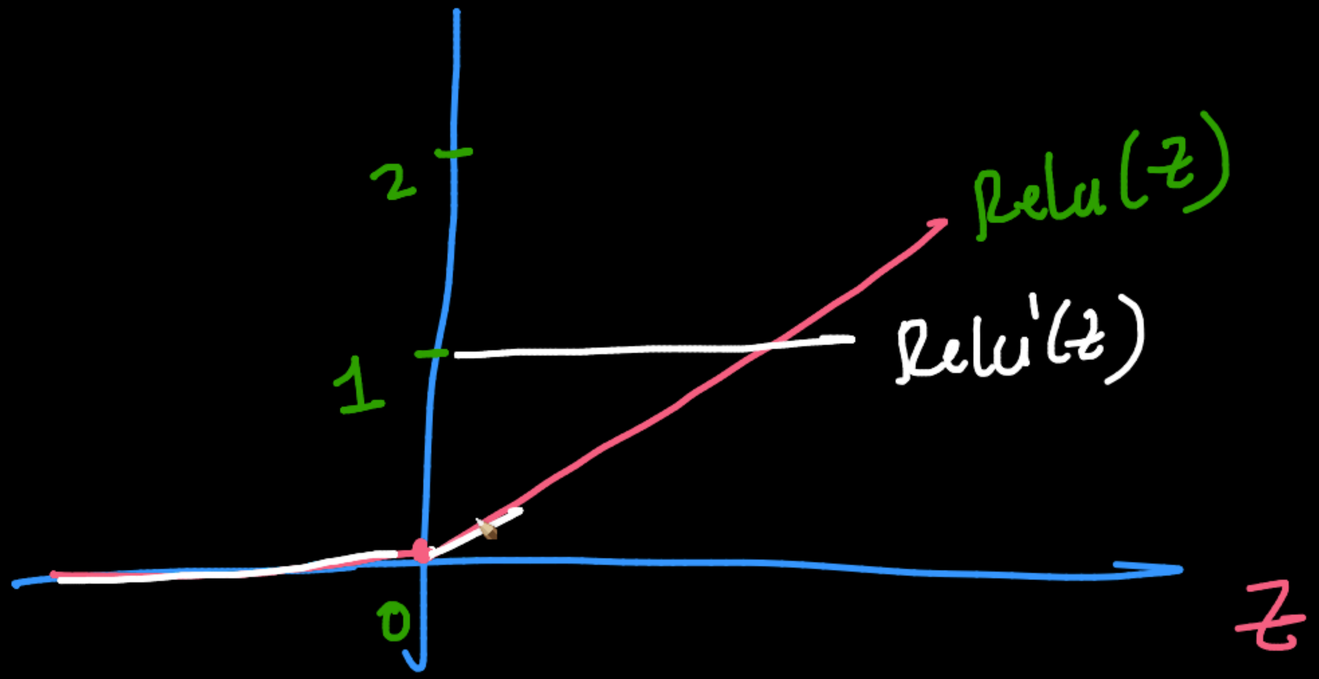
$$w'_{11\,new} = w'_{11\,old} - \eta \frac{\partial L}{\partial w'_{11}}\Bigg|_{w'_{11\,old}}$$

(updated) v.v.v. slowly because of vanishing gradients

minuscule


{10-layers} MLP
1984 - 2006
2012

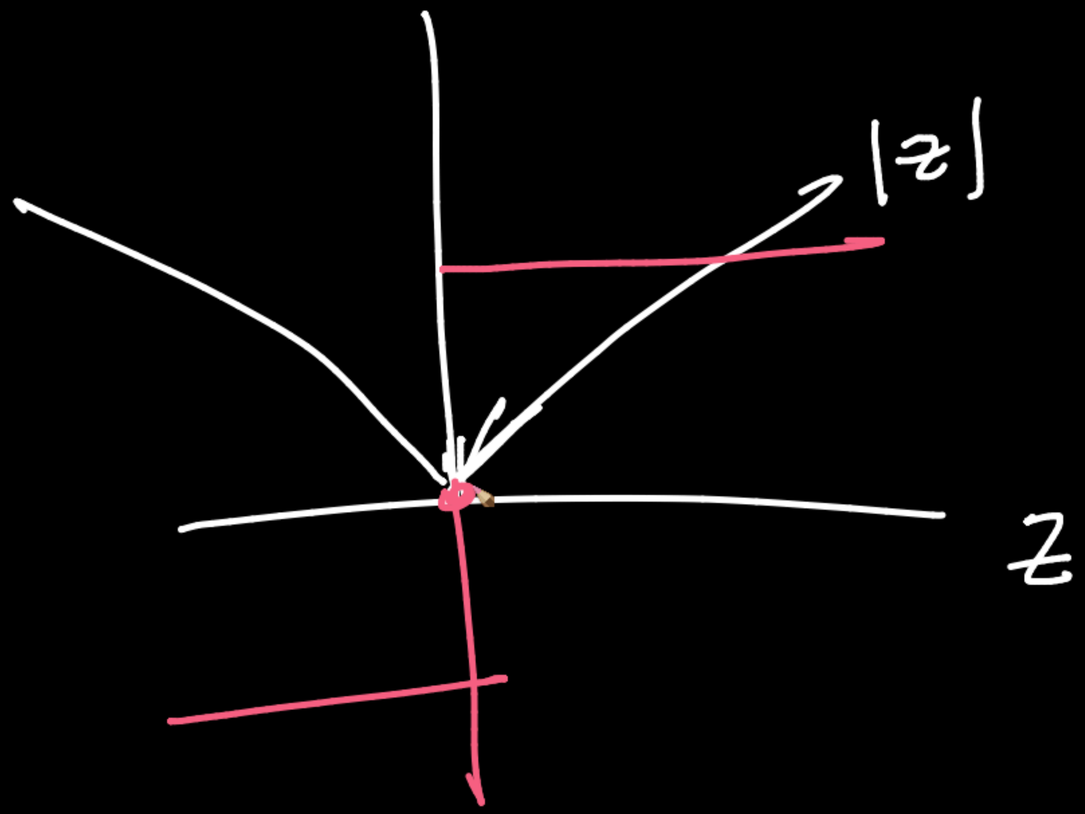
ReLU



$$Relu(z) = max(0, z)$$

$$Relu'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

Simple & fast



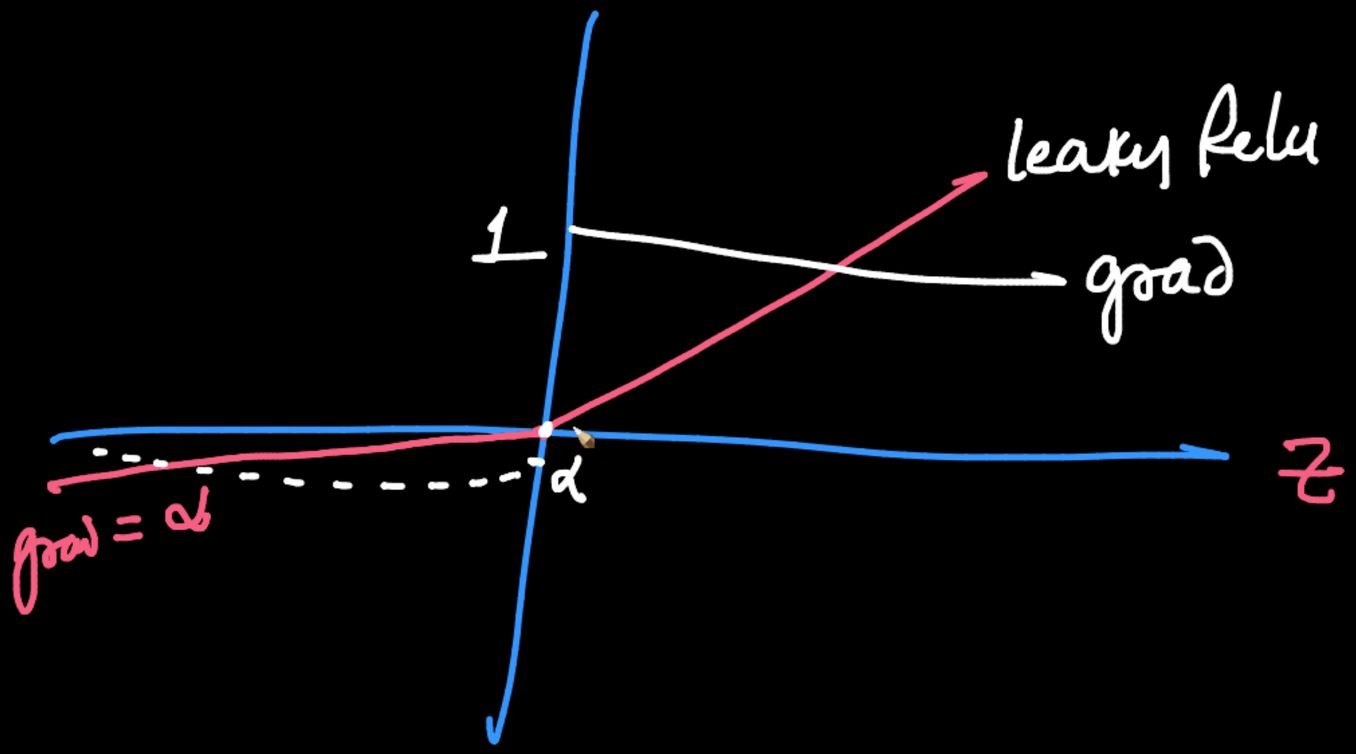
Most widely used

$$\text{Relu}'(z) = \begin{cases} 1 \\ 0 \end{cases}$$

$1 \times 1 \times 1 \cdots$ no vanishing grad

$1 \times 1 \times 1 \times 0 \cdots$ grad = 0 $\Rightarrow$ no update

leaky - Relu



Next - class:

→ Softmax & CE → K-classes

→ code (Softmax, CE, ReLu)

↑ Backprop from scratch ...

Backprop: Chain-rule + memoization

multiplying lots of partial-der

tanh, Sigmoid, Relu/leaky Relu

0 to 1 → Vanishing grad

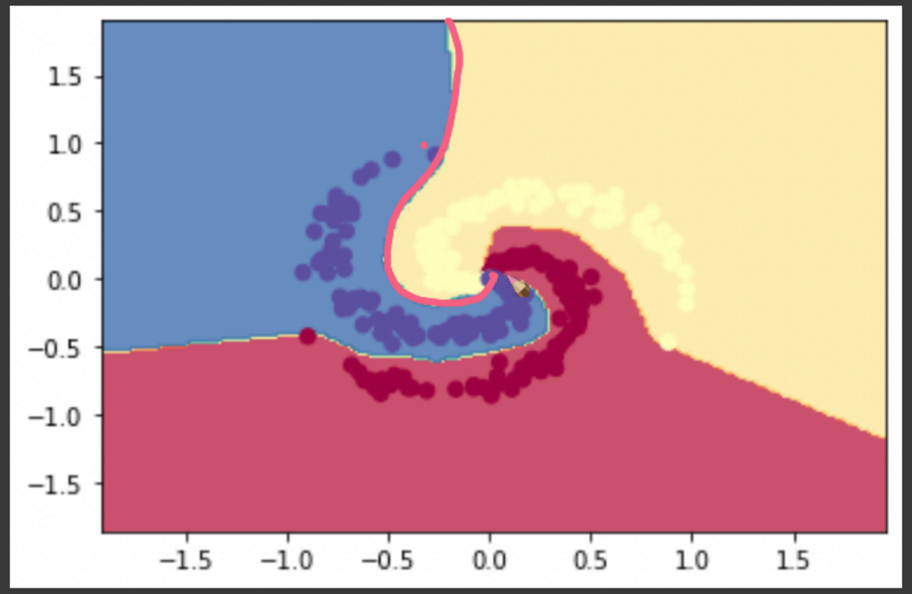
> 1 → exploding gradients

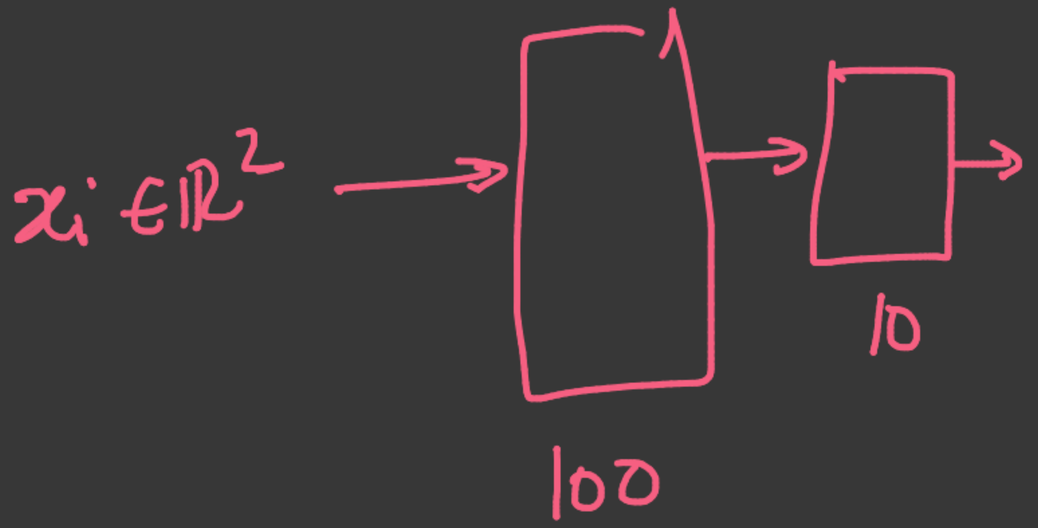
1.2 x 1.3 x 1.4x - -

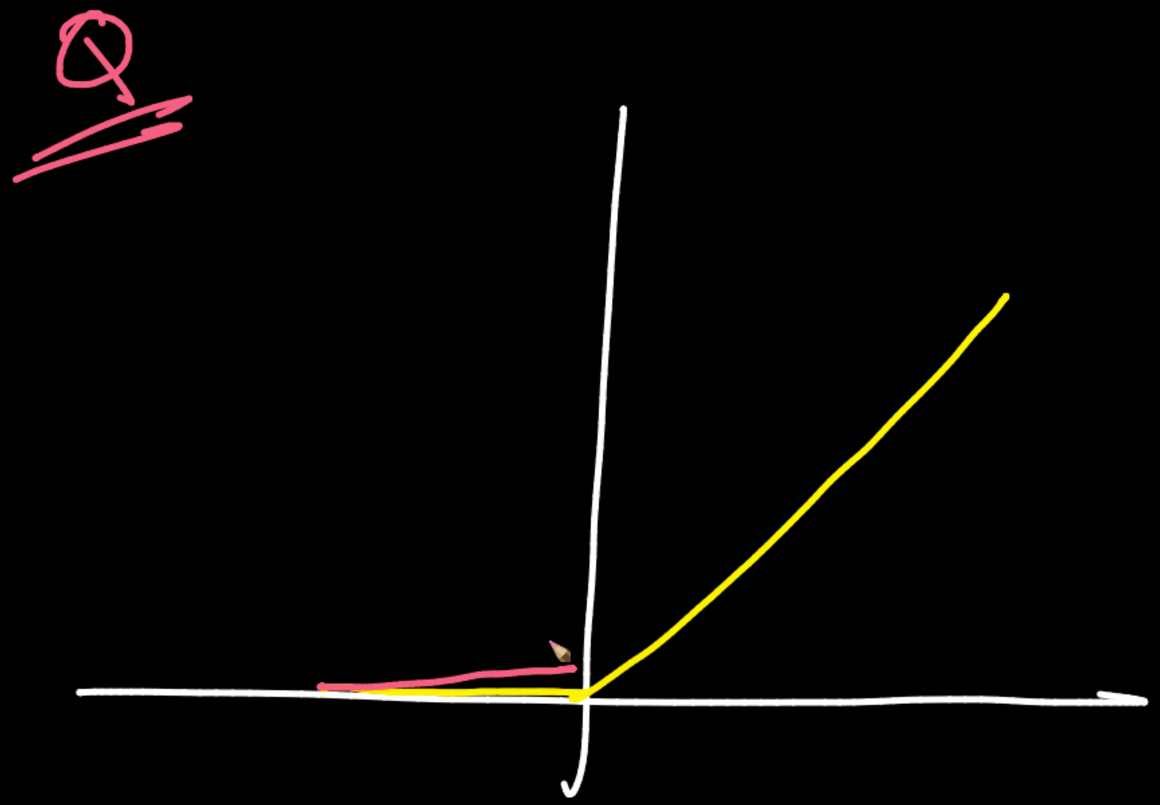
NLP (RNN)

58 / 58

[ ]



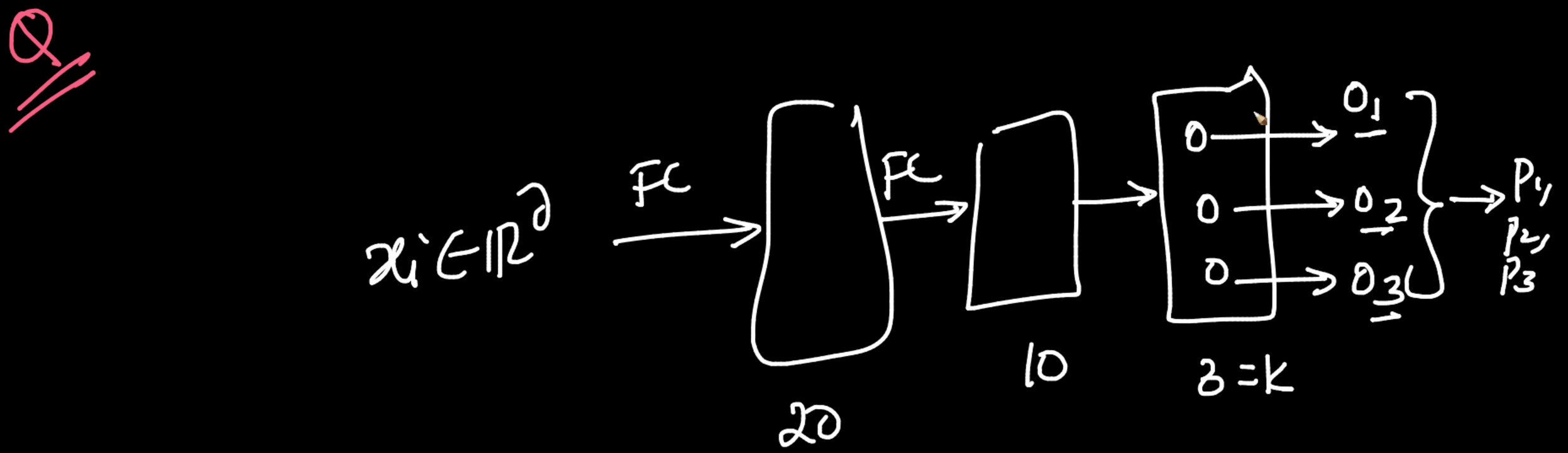




ReLu(. . . .

[Code → next class]

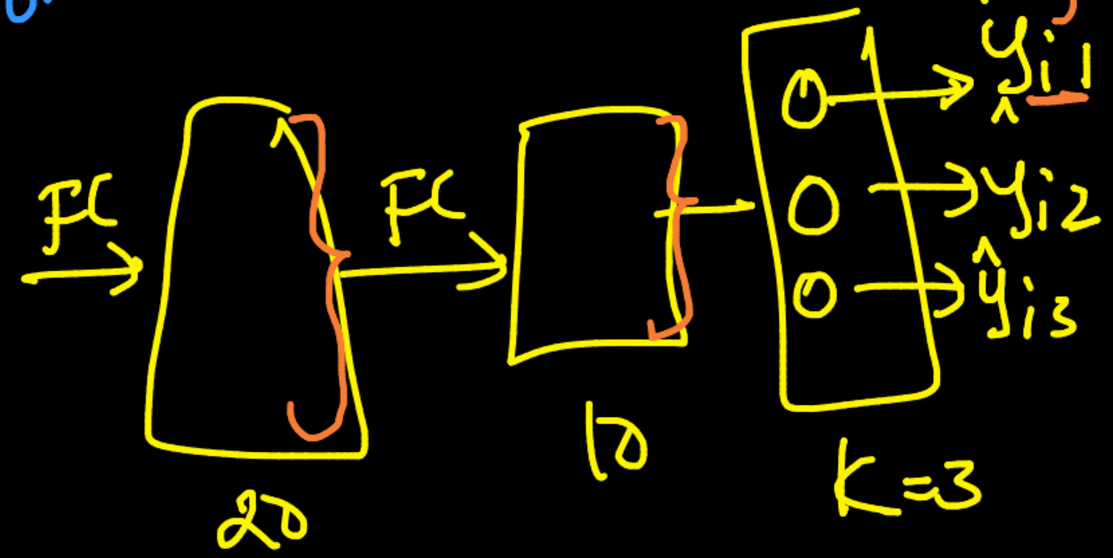
TF → 2-layer Relu


Q
$x_i \in \mathbb{R}^d$
FC
FC
20
10
3 = k
O_1
O_2
O_3
P_1
P_2
P_3

[Regression] ← Vector of $y_i \in \mathbb{R}^{k=3}$ → Sale price, # bedrooms, Rent Per Month

$x_i \in \mathbb{R}^d$

↳ home sq.ft, loc - - -



Q

mini-batch =

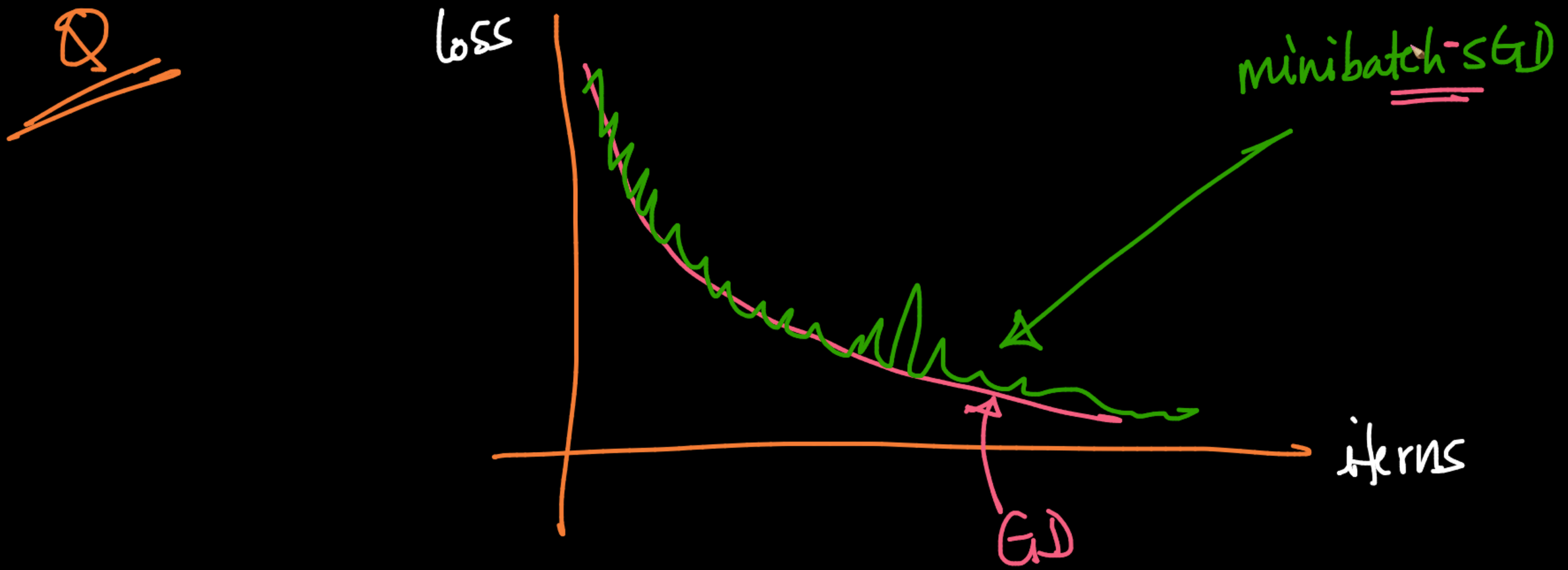
K-random (K = 1000) data points for each iter^{n}

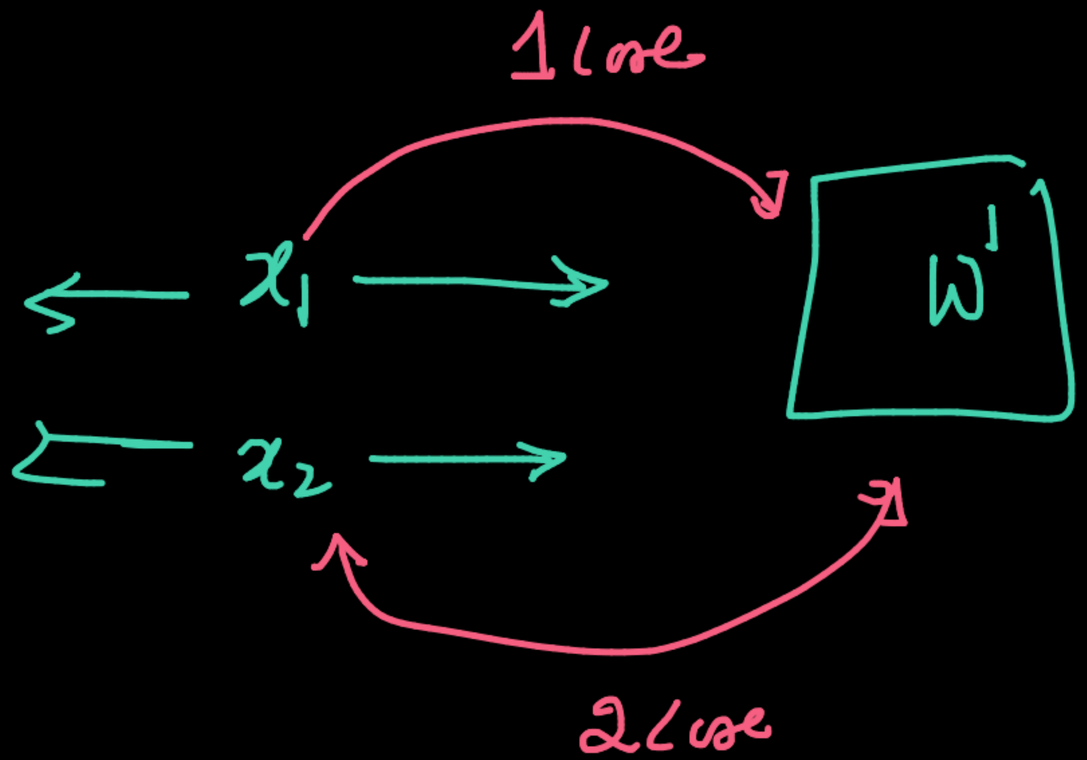
$n = 10^7$

epoch: (later)

1. Shuffle $10^7 = n$ pts randomly
2. { 1000 pts } every time per iter^{n}

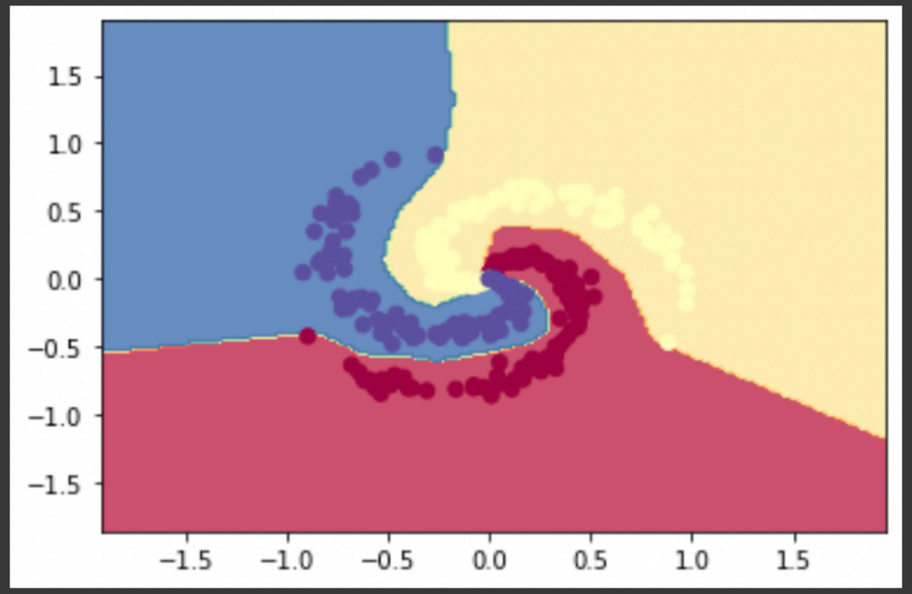
↳ 10^4 times / iterns


Q
loss
minibatch-sGD
iterns
GD



CPU: 8-cores

GPU: 1000's Acores (lakh)


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5