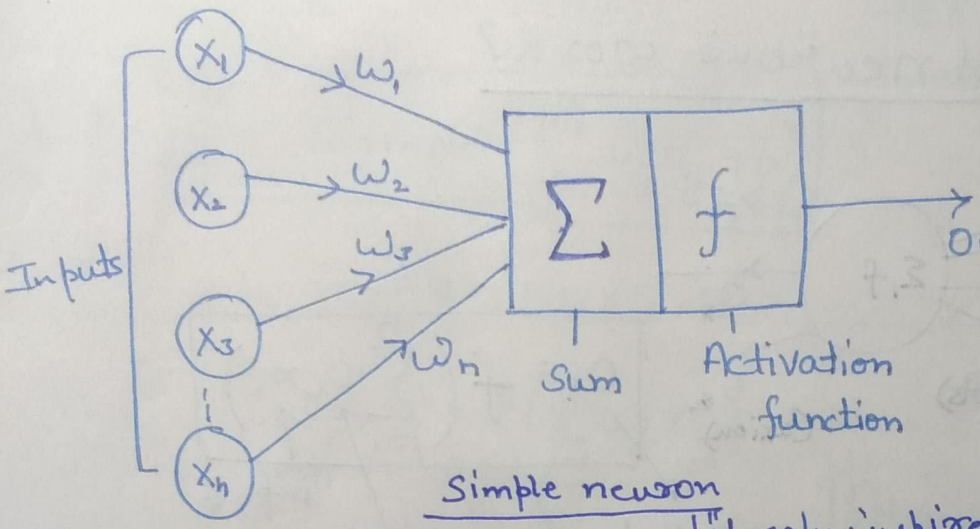
"DEEP LEARNING"

Neural Nets.

① History of Neural Networks.

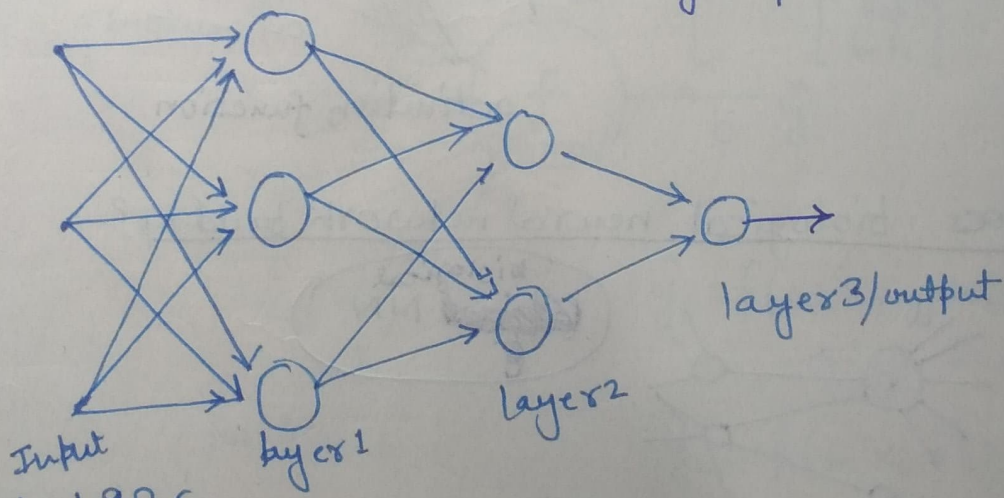
→ "Perceptron" → 1957 [Roseblatt]

similar to ↳ "Logistic regression"



Simple neuron

↳ "loosely inspired from biology"



Around → 1986

↳ "Back propogation" → chain rule of diff

1990+ AI funding died called 'AI winter'.

2006 → "Hinton" paper came

2012 → Image Net (large data) @ stanford.

Image dataset.

Image → object?

DNN model (computation)

"huge margin"

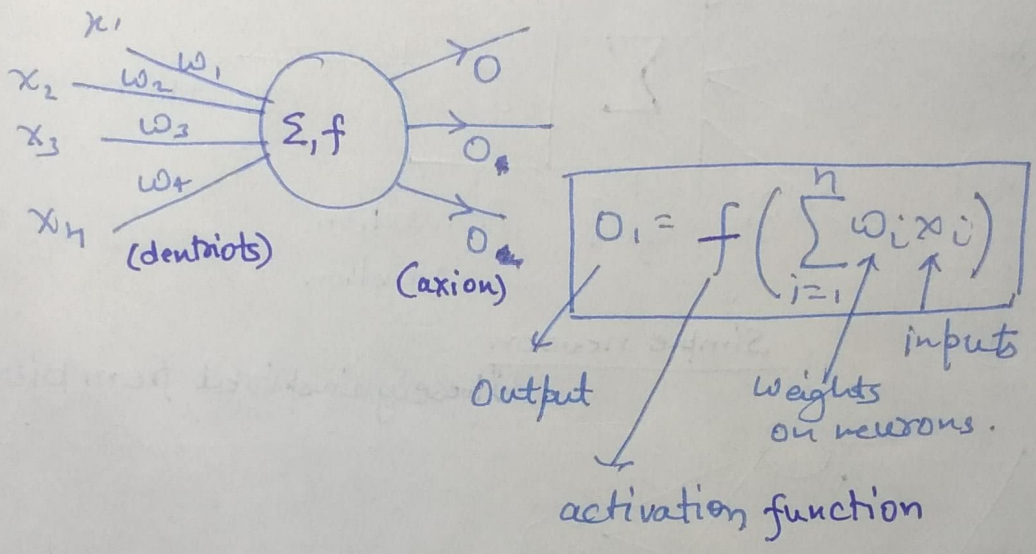
↳ this DNN beat every algorithm by Huge margin.

then funding started by Google, Ms, Fb, baidu, Amazon and golden days of AI started.

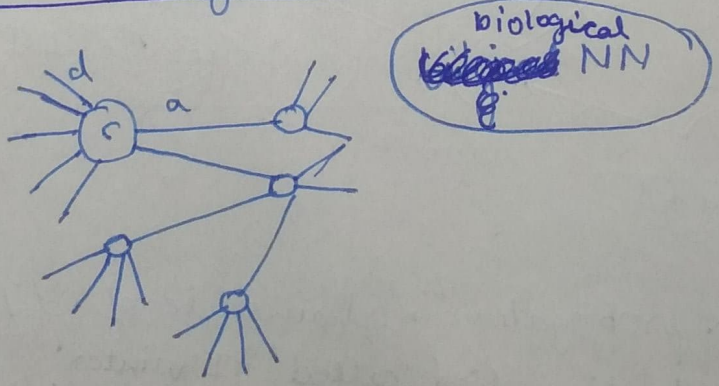
DL - CNN → 1990's but popular due to ① Large dataset.
② Huge Computational power
③ New Algorithms.

- NN → "classical" 1986
- MLP → deep NN 2006
- CNN →
- RNN →

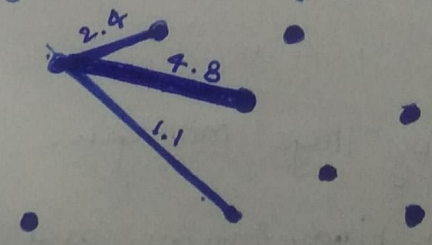
② How biological neurons work?



③ How does biological neural network build up?

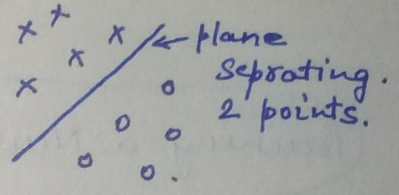


* learning ⇒ connecting neurons with edges.
all biological learning ⇒ weights on neural connections.



- Artifical Neural network is just connection of neural nets and weights on neural Connections.

4) Logistic Regression & Perceptron



L.R :- $x_i \rightarrow \hat{y}_i \rightarrow$ predicted value of y_i

$$\hat{y}_i = \text{sigmoid}(w^T x_i + b)$$

$D = \{x_i, y_i\}$ ← Data

Train L.R. $\Rightarrow w, b$ ← weight & bias are obtained by training.

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

$b \in \mathbb{R}$

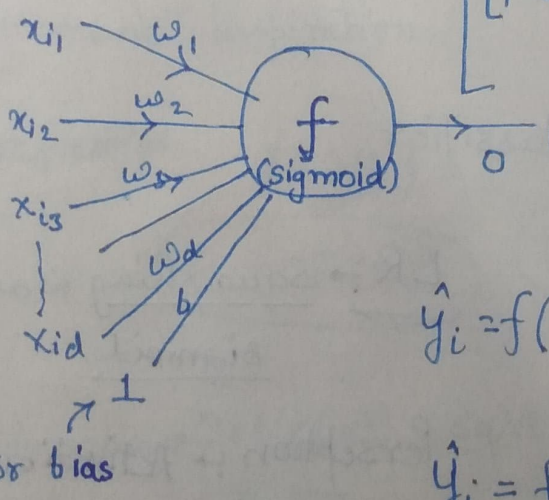
L.R: $$\hat{y}_i = \text{Sigmoid}\left(\sum_{j=1}^{d} w_j x_{ij} + b\right)$$

$x_i = [x_{i1}, x_{i2}, x_{i3}, x_{i4} \ldots\ldots x_{id}]$

$$O = f\left(\sum_{j=1}^{d} w_j x_{ij}\right)$$

O ← output

[[Neuron interpretation] [of] [L.R]]



$$\hat{y}_i = f(w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + \ldots\ldots + w_d x_{id} + 1 \times b)$$

$$\hat{y}_i = f(w^T x_i + b)$$

if f is sigmoid function then NN is logistic regression.

In L.R.

Task was from $D = \{x_i, y_i\}$

to find Task = w_i, b ? $\forall i = 1 \ldots d$

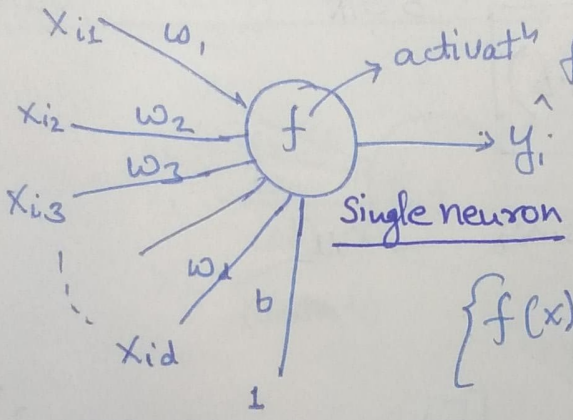
SGD → Lr. regression & optimization

Training a Neural Nets ⇒ Weights on edges / vertices.

So for LR ⇒ Neuron was made
& Sigmoid was Activation function

Perceptron → 1957

$w^T x = [w_1\ w_2\ w_3 \cdots w_j]_{j \times 1} \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ij} \end{bmatrix}_{1 \times j}$



activat^n function

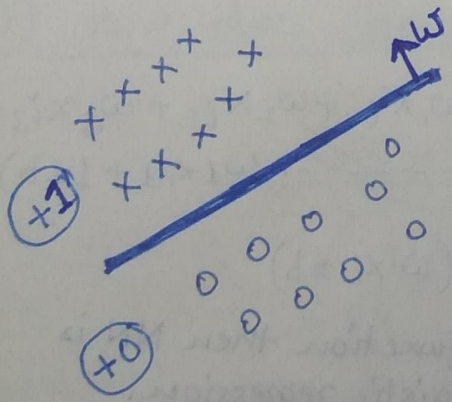
Single neuron

$w^T x = [w_1 w_2 w_3 \cdots w_j]_{j \times 1} \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ij} \end{bmatrix}_{1 \times j}$

$\sum_{i=1}^{d} w_j x_{ij} + b$.

$$\begin{cases} f(x) = 1 & \text{if } w^T x_i + b > 0 \\ = 0 & \text{otherwise} \end{cases}$$

Perceptron → Lr. classifier



LR :⇒ Squashing was done by sigmoid.

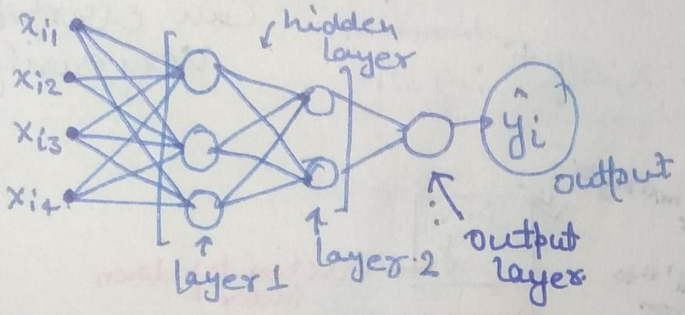
Perseptron :⇒ Activation function.

(5)

MULTI-LAYERED PROGRAM PERCEPTRON (MLP)

Perceptron → single neuron

↳ similar to logistic regression



(Q) why should be care about (MLP)

(a) biological inspiration

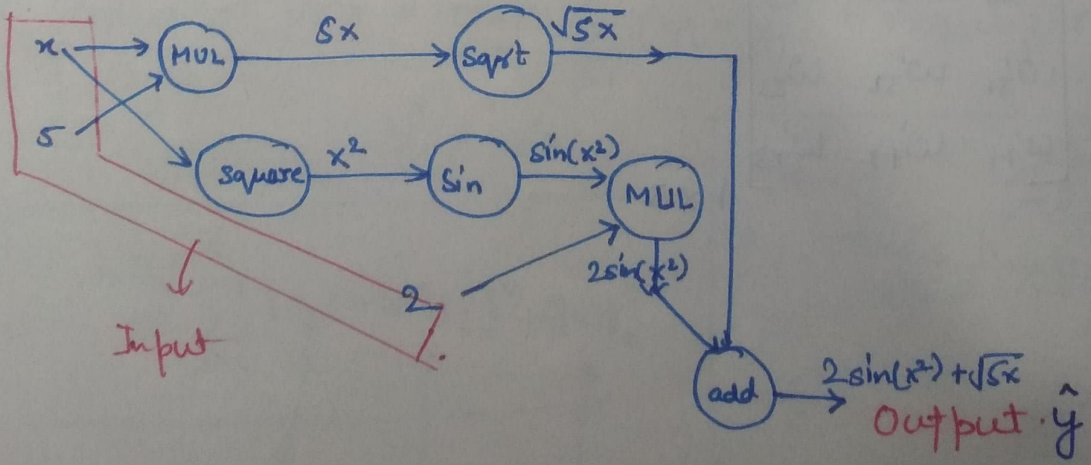
Neuroscience → human, rats, monkeys.

(b) Mathematic inspiration

regression $\{x_i, y_i\} = D$

$y_i = F(x_i)$ $\quad y_i \in R$

Q. Say a complex function → $2*\sin(x^2) + \text{sqrt}(x*5)$



So complex problem can be solved easily.

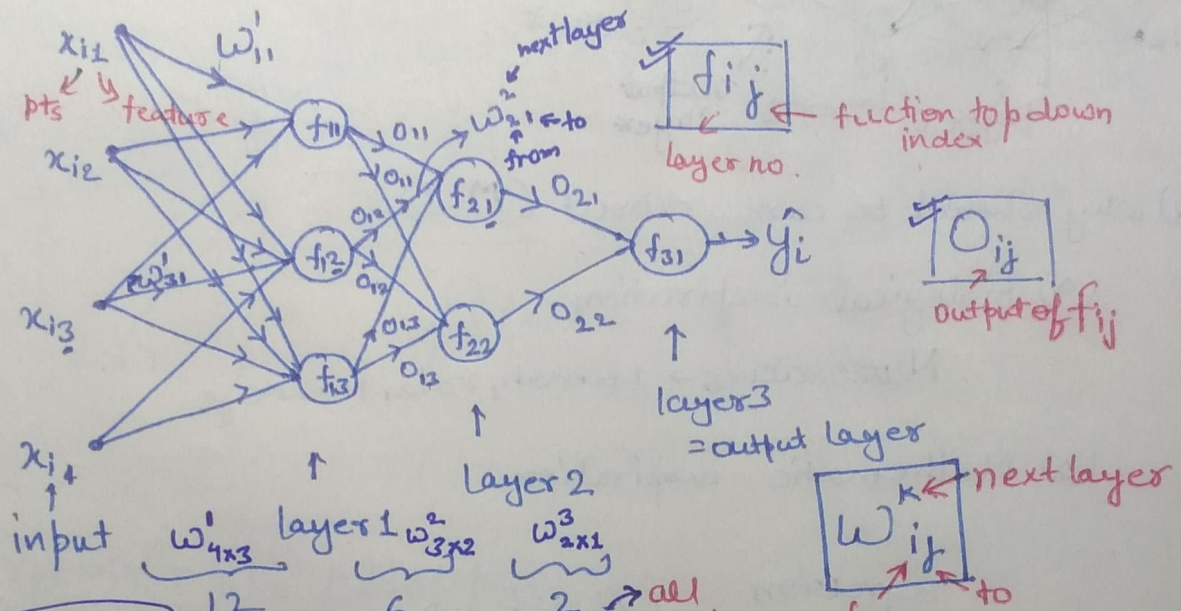
this is MLP-like structure.

Multi-layer → enormous power

Similar concept of function composition:- $g(f(x))$ or $f(g(x))$

MLS (Multilayer structure) → easily overfit.
↳ powerful model and powerful models Can overfit very easily.

(6) Notation $\quad D = \{x_i, y_i\} : x_i \in \mathbb{R}^{4 \leftarrow \text{dimension}} ; y \in R$



all connection made called fully Connected NN.

$$w_1 = \begin{bmatrix} w'_{11} & w'_{12} & w'_{13} \\ w'_{21} & w'_{22} & w'_{23} \\ w'_{31} & w'_{32} & w'_{33} \\ w'_{41} & w'_{42} & w'_{43} \end{bmatrix}_{4\times3}$$

7) Train a single Neuron model

find the best edge weights using D_{tr}. ← training Data.

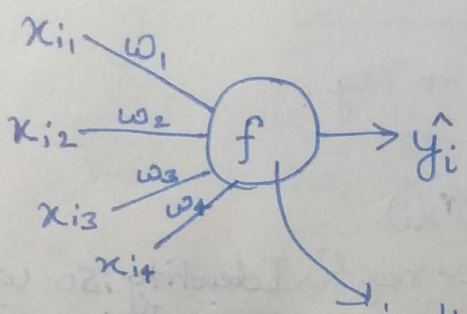
perceptron & LR → single neuron models for classification

we use regression because maths is easy.

Lr. Regression → " " " for regression

Lr. Reg & optim. → SGD

Linear



Lr. reg.

$\hat{y}_i = \sum_{j=1}^{d} w_j x_{ij}$

$x_i \in \mathbb{R}^d$ or $\hat{y}_i = w^T x_i$

$y_i \in \mathbb{R}$

in this case

$\begin{cases} \text{Input} = w^T x_i \\ \text{output} = w^T x_i \end{cases}$ } f is identity function. for this neuron to be working as Lr regression.

In case of Log. Reg.

f = logistic function.

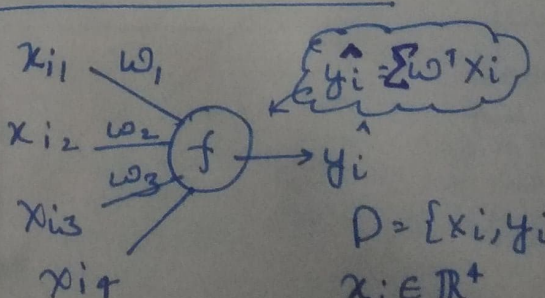
Lr. Regression

Optimisat^n prob = $\min_{w_i} \sum_{i=1}^{n}$ (no. of point) $(y_i - \hat{y}_i)^2$ + regulizer.

$\hat{y}_i = w^T x_i$

$\Rightarrow \min_{w_i} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \|w\|_2^2$

y_i $\hat{y}_i$ regulizer



$D = \{x_i, y_i\}_{i=1}^{n}$

$x_i \in \mathbb{R}^4$

$y_i \in \mathbb{R}$

How to solve using neural nets?

in short how to find different weights?

steps, are given in next page.

① Define Loss function

Loss across all the data ⇒ $L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + reg.$

Loss across just 1 training point ⇐ $L_i = (y_i - \hat{y}_i)^2 \Rightarrow \hat{y}_i = \omega^T x_i$ &

actually $L = \sum_{i=1}^{n} L_i$

L ↑ Sum Loss of across all points. L_i ↑ loss across individual training point i

② Optimization problem

Lr reg. ⇒ $\min_{\omega_i} \sum_{i=1}^{n} (y_i - \underbrace{\omega^T x_i}_{\hat{y}_i})^2 + reg.$

$\hat{y}_i = f(\omega^T x_i)$

↳ Lr reg f is Identity, so we write $\omega^T x_i$ as it as.

Neuron ⇒ $\min_{\omega_i} \sum_{i=1}^{n} [y_i - f(\omega^T x_i)]^2 + reg$

↳ I ⇒ Lr reg

↳ sigmoid ⇒ logistic regression

$$\omega^* = \arg\min_{\omega} \sum_{i=1}^{n} (y_i - f(\omega^T x_i))^2 + reg$$

← general optimization problem of neuron

③ Solve the optimizn problem. (SGD) → Watch optimization Chapter for understanding properly.

a) initialize of ω_i's → random

b) compute $\nabla_\omega L = \begin{bmatrix} \frac{\partial L}{\partial \omega_1} \\ \frac{\partial L}{\partial \omega_2} \\ \frac{\partial L}{\partial \omega_3} \\ \frac{\partial L}{\partial \omega_4} \end{bmatrix}$

c) $\omega_{new} = \omega_{old} - \eta [\nabla_\omega L]_{\omega_{old}}$

$(\omega_i)_{new} = (\omega_i)_{old} - \eta \left[\frac{\partial L}{\partial \omega_i}\right]_{(\omega_i)_{old}}$

$(\omega_i)_{new}$ ↑ update weight

η ← learning rate we drive next

$\left[\frac{\partial L}{\partial \omega_i}\right]_{(\omega_i)_{old}}$ ← with respect to ω_i old

for iter 1 —— K till converse.

GD :-

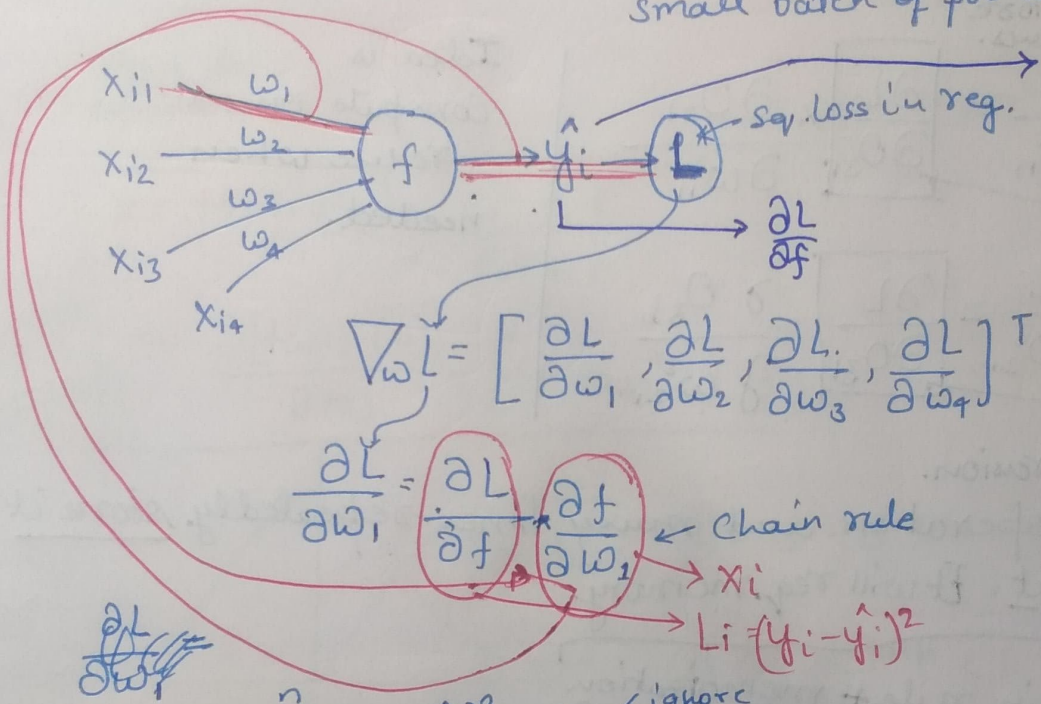
Gradient disent : $\nabla_w L \rightarrow x_i$'s & y_i's → In GD (gradient discent) we do this for all x_i's & y_i's together

→ In SGD we do this for small batch of points or individual point one by one. ↳ SGD.

min-batch SGD

SGD : $\nabla_w L \simeq$ one pt $\{x_i, y_i\}$ ← batch SGD

small batch of points



Now, this $\hat{y}$ predicted & actual may have some error based on that we define Sq loss.

$$\nabla_w L = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial w_3}, \frac{\partial L}{\partial w_4}\right]^T$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial w_1}$$ ← chain rule

$\frac{\partial f}{\partial w_1} \rightarrow x_i$

$\frac{\partial L}{\partial f} \rightarrow L_i = (y_i - \hat{y}_i)^2$

$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \cancel{reg}$$ ignore for now

$$= \sum_{i=1}^{n} (y_i - f(w^T x_i))^2$$

We can calculate $\frac{dL}{dw_1}$ from these two

$$\frac{\partial L}{\partial f} = -\sum_{i=1}^{n} 2x(y_i - f(w^T x_i))$$

$$\frac{\partial f}{\partial w_1} = x_i$$

$$\Rightarrow \frac{\partial L_i}{\partial w_1} = -2x_i(y_i - \hat{y}_i)$$

$$\Rightarrow \boxed{\frac{\partial L}{\partial w_1} = \sum_{i=1}^{n} -2x_i(y_i - \hat{y}_i)}$$

we are given with data. we know actual y_i we get this with respect to old weights

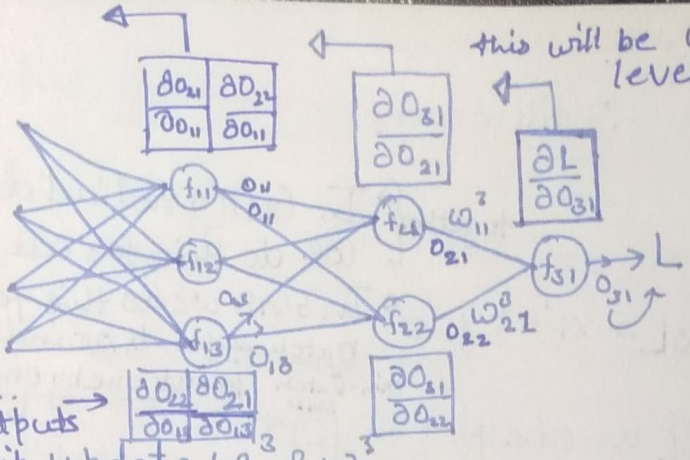
8) Train an MLP chain rule. ↳ go to last.

9) Memoization :-

Algorithm → Dynamic Programming

↓ Not memorisation

↓ Storing the same expression value rather calculating derivatives again & again.



(w^3)

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

$$\frac{\partial L}{\partial w^3_{21}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w^3_{21}}$$

Same expression.

Idea is Compute once & reuse when needed.

if any operation used many times repeatedly, store it and use it. It will req. memory.

chain rule + memoization

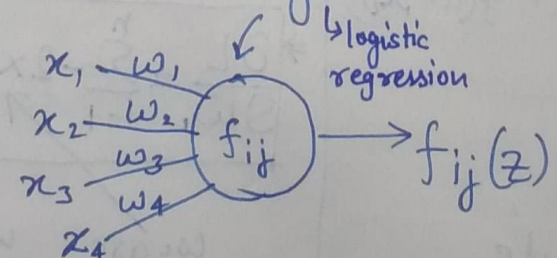
(10) Backpropogation → go to last

Backpropogation Algorithm.

(11) Activation functions (f_{ij})

1980 & 1990's → Sigmoid & tanh ← 2 of the oldest activation function.

easily differentiable.



$$\sum_i w_i x_i = w^T x$$

$$w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

input = z

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

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

↙ easily differentiable.

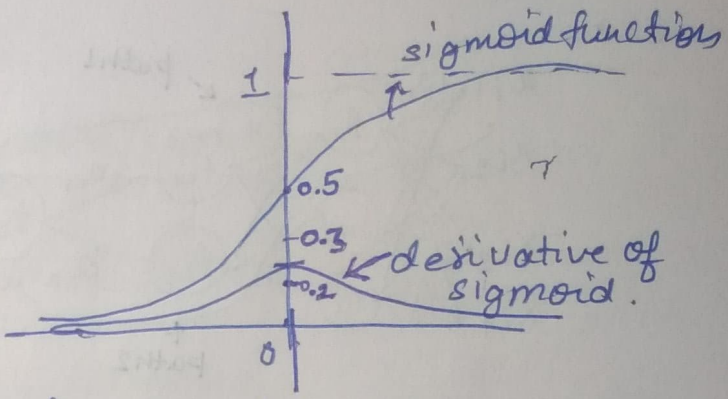
* this seems difficult to differentiate but it is just $\sigma(z)[1-\sigma(z)]$

Req. of Activition

1) differentiable
2) easy to differentiable

$$\frac{\partial \sigma}{\partial z} = \sigma(z)[1-\sigma(z)]$$

↓ sigdiff come in term of sig only.



tan h :-

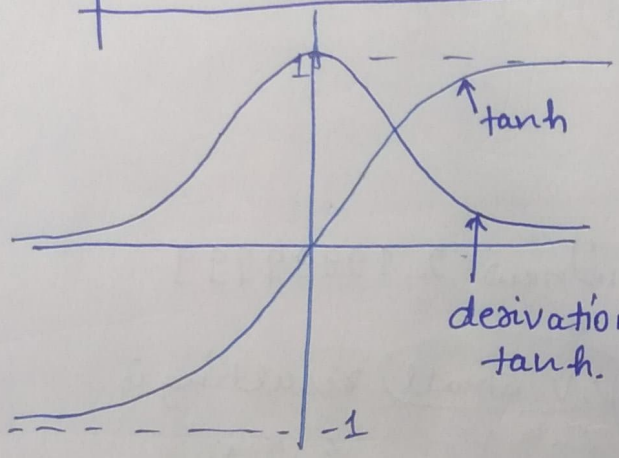
$$\frac{\partial a}{\partial(z)} = 1 - a^2 \; ; a \rightarrow \text{tanh function.}$$

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

$$\frac{\partial[\tanh(z)]}{\partial(z)} = [1 - \tan^2 h(z)]$$

for proof do it yourself.

easily differentiated.



① * -1 to 1 ← tanh

② $0 \leq \frac{\partial \tanh}{\partial z} \leq 1$

* One thing to note shape of sigmoid & tanh is same it is just the range.
sigmoid → (0,1)
tanh → (-1,1)

Modern techniques are very popular.

like ① ReLU.
② Leaky ReLU.

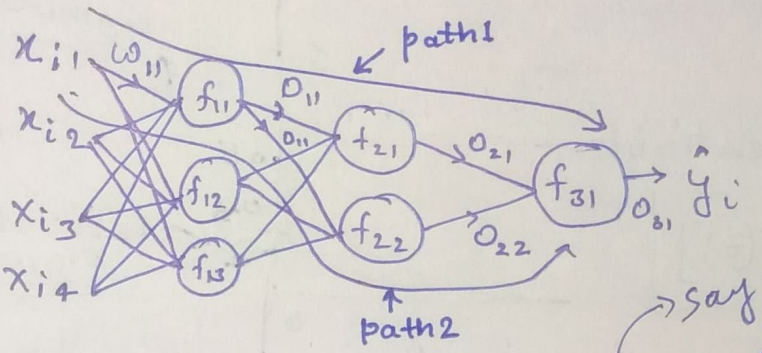
(12) Vanishing gradients

80, 90, 00's → (NN) → problem.

Most important reason why people lost interest in deep-learning because of problem of vanishing gradient.

f_j → Sigmoid.

ex→



$$(w_{11}')_{new} = (w_{11}')_{old} - \eta \frac{\partial L}{\partial w_{11}'}$$

→ say 1 for explanation also it is v v small.

→ v v small

$$\frac{\partial L}{\partial w_{11}'} = \frac{\partial L}{\partial O_{31}} \left[\frac{\partial O_{31}}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial w_{11}'} + \frac{\partial O_{31}}{\partial O_{22}} \times \frac{\partial O_{22}}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial w_{11}'} \right]$$

→ O_{31} is output of sigmoid function & there diff is b/w 0 & 0.25.

$0 < <1$ $0 < <1$ $0 < <1$ $0 < <1$ $0 < <1$ $0 < <1$

$< 10^{-3}$

If $(w_{11}')_{old} = 2.5$

$(w_{11}')_{new} = 2.499$

Say if deep layers.

then

$(w_{11}')_{old} = 2.5 \quad (w_{11}')_{new} = 2.49999999$

① Vanishing gradiend $\frac{\partial L}{\partial w_{ij}^{k}}$ = v.v. small resulting is $(w_{ij}^{k})_{old} \simeq (w_{ij}^{k})_{new}$

Solution Relu.

because derivative of Sigmoid / tanh both less than 1.

updating take many many iteration.

So people were only training 2 hidden layer.

to build deep NN → {10, 20, 100} → then $\frac{\partial L}{\partial w_{ij}^{k}}$ would be of 10^{-10} or 10^{-20} or 10^{-100}

this is the reason we cannot add deep neural network or large no. of layers.

② Exploding gradient

*although we don't have this problem in sigmoid/tanh activation

Chain rule: multi lots of derivation

If $\frac{\partial L}{\partial w_{11}'}$ is greater or v.v. large.

In this case $(w_{11}')_{new} = (w_{11})_{old} - \eta \left(\frac{\partial L}{\partial w_{11}'} \right)$ ← large

large derivative is even bigger problem than small derivation.

then (w_{11}') will never converge and go crazy

*even after 100 epoch we will never converge.

(13) Bias-Variance Tradeoff

(1) #layer ↑ ⇒ more weights/params

⇓

Chances of overfitting ← why?

⇓

high variance.

generally we use more layers and have problem of high variance (overfitting)

ex- (1 layer) = Log. Reg. → + + / − − → higher chance of underfit.

↓ with sigmoid activation function.

⇓

underfit

To solve overfit/underfit add regularizer:-

MLP → multiple layers → overfitting/high var

↳ optimization → add (regularization)

$$L = \sum_{i=1}^{n} loss(y_i, \hat{y}_i) + \sum_{i,j,k} (w^k_{ij})^2$$

L^2 regularizer

this regularization helps to avoid overfitting. (high variance).

⇒ L_2 regularization / L_1 regularization

$$L = \sum loss_i + \lambda L_2$$

reg on weights → we just add λ to give more weightage to regularizer.

larger λ ⇒ lesser overfit

problem with L_1 reg.

L_1 reg ⇒ Sparsity ⇒ some weights becomes 0 → visit old lectures for clarity.

⇓

fewer edges become non-zero.

Then in L2 reg. λ become hyperparameter.

Second hyperparameter is no. of layers.

now, we have 2 hyperparameter

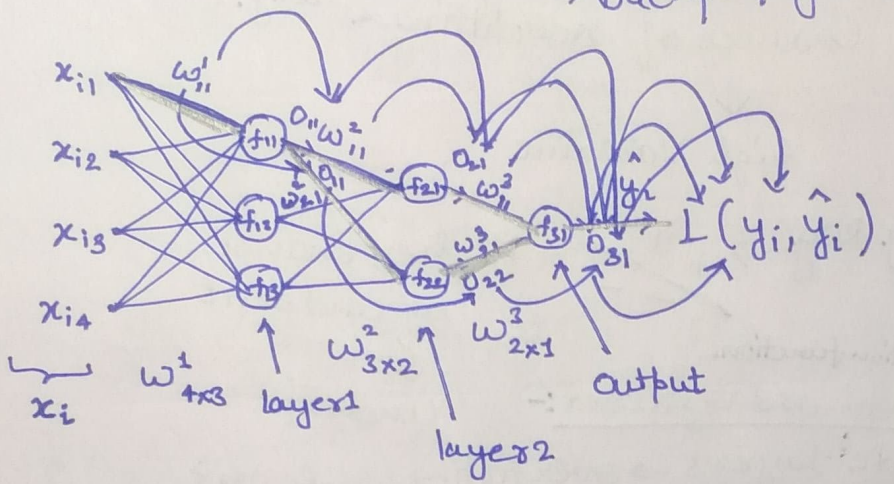
(1) λ → weight of regularizer.

(2) No. of layers. (we dont want less or more layers).

(14) Playground · tensorflow.org

(8) Training an MLP chain Rule.

↳ Backpropogation Algorithm



Now we have find $4\times3 + 3\times2 + 2\times1 = 20$ weights.

(1) $L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + reg$

↓ Define loss?

Sq. Loss

$\sum_{i,j,k} (\omega^k_{ij})^2$ ← this is something like L2 regulizer.

$\sum_{i,j,k} |\omega^k_{ij}|$ ← this is L1 regulizer

(2) Optimization problem:-

$\min_{\omega^1, \omega^2, \omega^3} (L)$ ← Sq. loss + reg.

(2) SGD or GD (solving optimization problem)

(a) initialize ω^k_{ij} randomly } → smart initialisation lot of technique.

(b) $(\omega^k_{ij})_{new} = (\omega^k_{ij})_{old} - \eta \left[\frac{\partial L}{\partial \omega^k_{ij}}\right]_{(\omega^k_{ij})_{old}}$ ← learning rate

Converge ω^k_{ij} untill $(\omega^k_{ij})_{new} \approx (\omega^k_{ij})_{old}$.

ex → $\frac{\partial L}{\partial \omega^3_{11}} = \frac{\partial L}{\partial O_{31}} \times \frac{\partial O_{31}}{\partial \omega^3_{11}}$

$$\frac{\partial L}{\partial w^3_{11}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w^3_{11}} \neq \frac{\partial L}{\partial w^2_{11}}$$

$$\frac{\partial L}{\partial w^3_{21}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w^3_{21}}$$

w^3's derivatives wrt. L are found.
(2) weights.

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

$$\frac{\partial L}{\partial w^2_{21}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial (w^2_{21})}$$

$$\frac{\partial L}{\partial w^2_{31}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial (w^2_{31})}$$

$$\frac{\partial L}{\partial w^2_{12}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{22}} \cdot \frac{\partial O_{22}}{\partial w^2_{12}}$$

$$\frac{\partial L}{\partial w^2_{22}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{22}} \cdot \frac{\partial O_{22}}{\partial w^2_{22}}$$

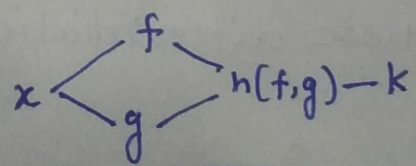
$$\frac{\partial L}{\partial w^2_{32}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{22}} \cdot \frac{\partial O_{22}}{\partial w^2_{32}}$$

w^2's derivatives wrt L are found.
(6) weights.

Now we have 2 path rather than just 1 → Now this way we can get other w's derivatives.

$$\frac{\partial L}{\partial w^1_{11}} = \frac{\partial L}{\partial O_{31}} \cdot \boxed{\frac{\partial O_{31}}{\partial w^1_{11}}} \leftarrow ??$$

$$\frac{\partial O_{31}}{\partial w^1_{11}} = \frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial O_{11}} \cdot \frac{\partial O_{11}}{\partial w^1_{11}} + \frac{\partial O_{31}}{\partial O_{22}} \cdot \frac{\partial O_{22}}{\partial O_{11}} \cdot \frac{\partial O_{11}}{\partial w^1_{11}}$$



$$\frac{\partial k}{\partial x} = \frac{\partial K}{\partial h} \boxed{\frac{\partial h}{\partial x}} \rightarrow ?$$

$$\boxed{\frac{\partial h}{\partial x} = \frac{\partial h}{\partial f} \cdot \frac{\partial f}{\partial x} + \frac{\partial h}{\partial g} \cdot \frac{\partial g}{\partial x}}$$

(18) Back propogation.

$D = \{x_i, y_i\}$ ← Dataset

(1) initialize w^k_{ij}'s

(2) for each x_i in D

- (a) pass x_i forward through the network
- (b) Compute $L(\hat{y}_i, y_i)$
- (c) Compute all the derivate chain-rule & memorization.
- (d) Update weights for end of the network to the start. $\left(w^k_{ij}\right)_{new} \leftarrow \left(w^k_{ij}\right)_{old} - \eta \left[\frac{dL}{dw^k_{ij}}\right]$

{GD: (one by one point)}

back propogation.

(3) repeat step 2 till convergence

if $\left(w^k_{ij}\right)_{new} \approx \left(w^k_{ij}\right)_{old}$.

when you pass all the datapoints for once we call it 1 epoch. we typically run for multiple epochs. & these x_i are choosen randomly in each epoch rather than one after other.

VV important Backpropogation algorithm only work when activation function is differentiable. and if these differntiation is fast & easy we can get fast NN.

#we will look mini-batch to optimise the result.

#we generally avoid GD when all the data backpropogation is used because it require lot of memory.