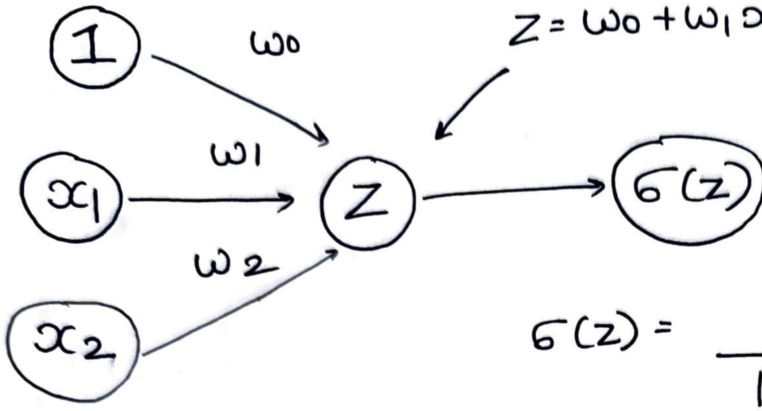
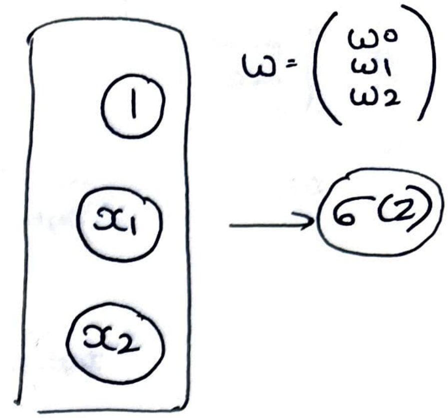
*ANNs → artificial neural networks :-

$$Z = w_0 + w_1 x_1 + w_2 x_2$$

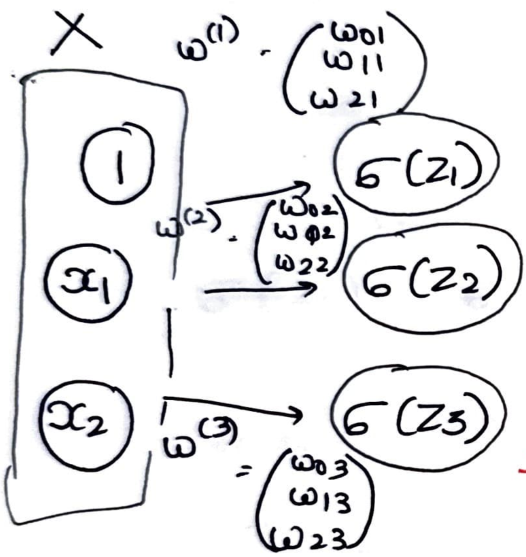


$$w = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$$



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

activation

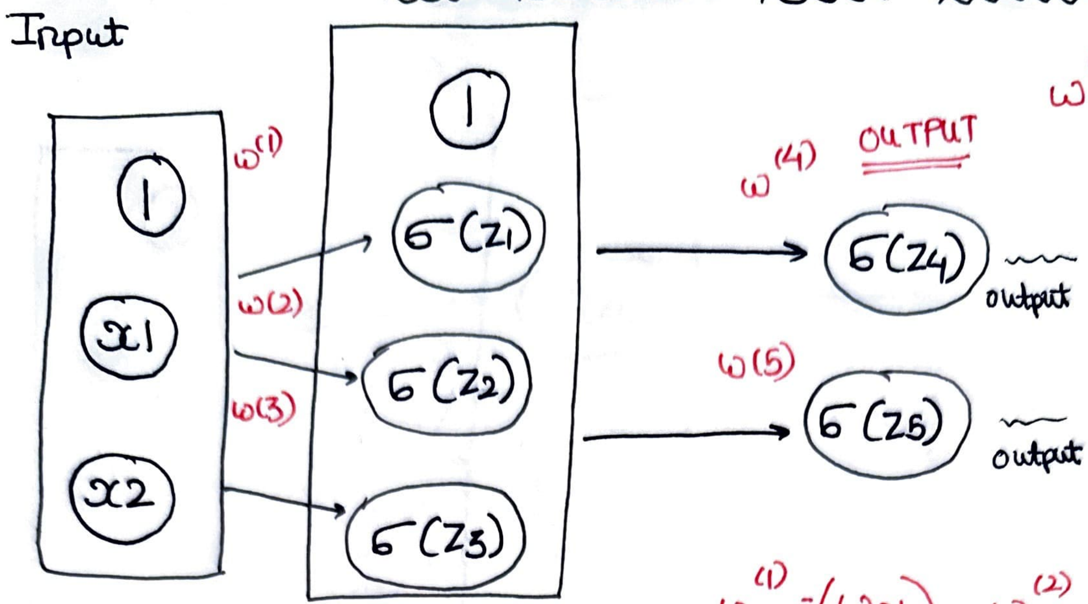


OVR → one vs rest

neurons work independently

Linear Classifier

* Connecting neurons together – 3 Layer networks



Hidden Layer

$$W^{(4)} = \begin{pmatrix} w_{04} \\ w_{14} \\ w_{24} \\ w_{34} \end{pmatrix}$$

4 weights

$$W^{(1)} = \begin{pmatrix} w_{01} \\ w_{11} \\ w_{21} \end{pmatrix} \qquad W^{(2)} = \begin{pmatrix} w_{02} \\ w_{12} \\ w_{22} \end{pmatrix}$$

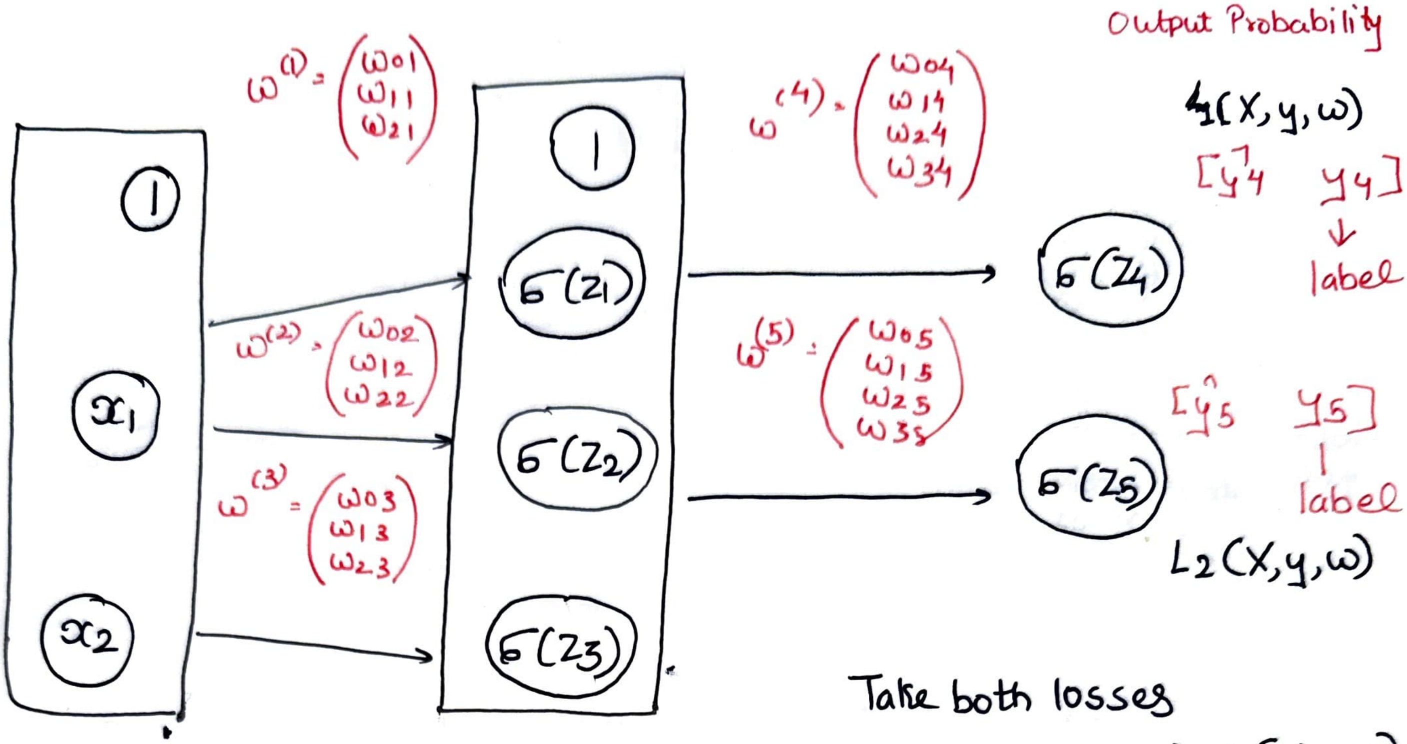
forward computation

$$\sigma(z_4) = \sigma\big(w_{04} + w_{14}\,\sigma\underbrace{(z_1)}_{\text{neuron 1}} + w_{24}\,\sigma\underbrace{(z_2)}_{\text{neuron2}} + w_{34}\,\sigma\underbrace{(z_3)}_{\text{neuron 3}}\big)$$

Each neuron has its own set of independent weights.

apply activation over this → that gives activation value for that layer

* Training 3 Layer Networks:



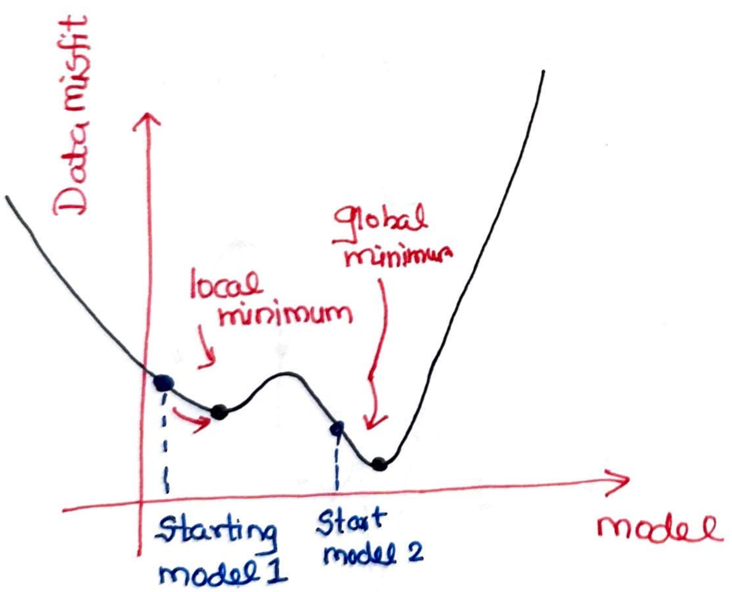
Take both losses

$L_1(X,y,w) + L_2(X,y,w) = \underline{L(X,y,w)}$

Updates weights over iterations via some gradient descent (To Reduce Loss)

* Complicated Landscape:-

(1-Dimension)

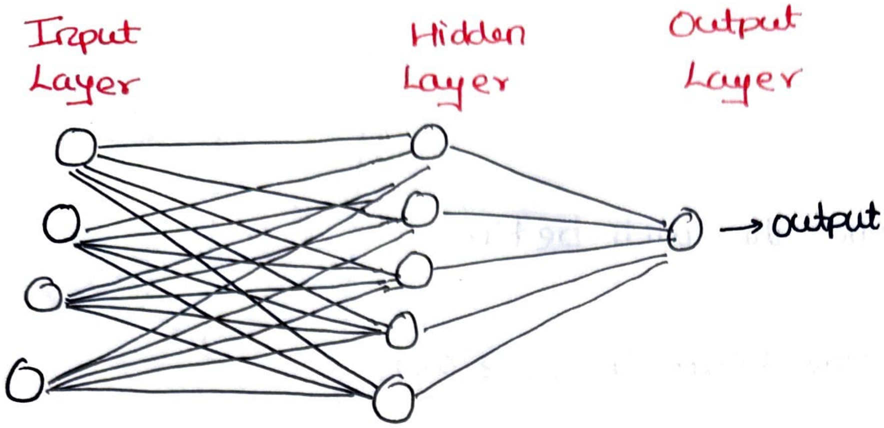


Local minimum → generalize

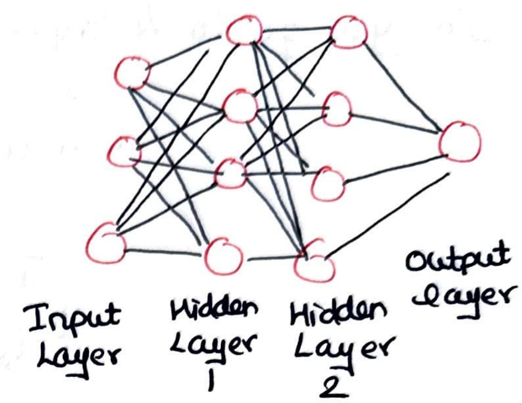
$L(X, y, w)$ → entire set of weights

Starting model 1 → (move forward based on step size/learning rate) → move to local minimum → not global minimum directly → However if we start at model 2 (global minimum can be found)

*Universal Approximation Theorem



(4-Layer)
More Powerful?



non linear

If the training dataset (X,y) comes from a 'reasonable' function $y = f(X)$ (where reasonable means almost any function you can imagine), then there exists a 3 layer neural network m such that m(X) approximates F(X) arbitrarily well ie the loss function of m can be driven to be very small on X.

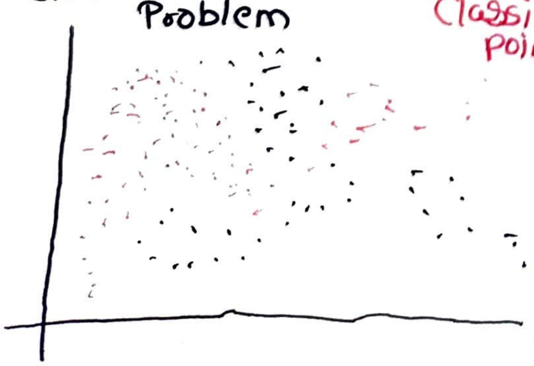
1

(X,y)

very similar

$m : m(X) \approx y$

2 Classification Problem



Classify points

Powerful 3 Layer Network

If you go to 4-layer

↓

you cannot do much better

↓

If you learn it on 4 layer

↳ some weights/setting exists on a 3 layer network that produces the same result.

So UAT is strongest for 3 Layer Networks.

It does not speak about how many nodes in hidden layer will be → so we don't know how big it will be

↓

it cannot be processed on a local computer

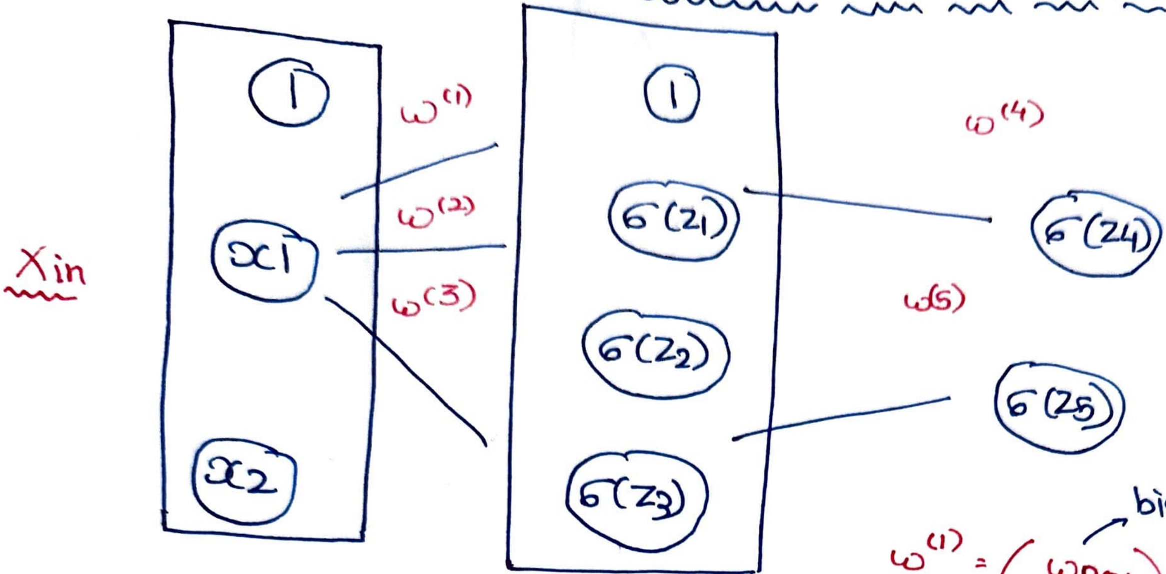
Combination of these issues force us to look at deeper networks

↓

3 Layer Networks → Theoretically better

Deeper Networks → Practically Better.

*Algebra of Forward Computation



$w^{(1)} = \begin{pmatrix} \underline{w_{01}} \\ w_{11} \\ w_{21} \end{pmatrix}$ → bias $w^{(2)} = \begin{pmatrix} \underline{w_{02}} \\ w_{12} \\ w_{22} \end{pmatrix}$ → bias

$w^{(4)} = \begin{pmatrix} w_{04} \\ w_{14} \\ w_{24} \\ w_{34} \end{pmatrix}$

other weights

$$W_{ih} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

weights ← $\underset{\sim\sim}{2 \times 3}$ → neurons

$$Z_h = \underbrace{x_{in}}_{\text{input point}} \cdot W_{ih} + \underbrace{[w_{01}, w_{02}, w_{03}]}_{\text{biases}}$$

The forward function (3 Layer)

nets $\quad Z_h = x_{in} \cdot W_{ih} + [w_{01}, w_{02}, w_{03}]$ → ih → input to hidden layer

$a_h = \sigma(Z_h)$ activations (hidden neuron) → become input for ne

nets $\quad Z_o = a_h \cdot W_{ho} + [w_{04}, w_{05}]$ → ho → hidden to output layer

$a_o = \sigma(Z_o)$ activations (output activation) → Final Output

*Take batch of points → $X_1, X_2, X_3, \ldots$ $m(X_1), m(X_2), m(X_3)$

send one by one to model / works on all parallel by making a matrix

$$X_{in} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

3×2

Parallism is important instead of single points (for GPU usage) efficient use of computation power.

$$Z_h = x_{in} \cdot W_{ih} + [w_{01}, w_{02}, w_{03}]$$

* Training with BackPropagation

$y(u(x))$

Chain rule

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

Example

y	u	x
TESLA	Bmw	

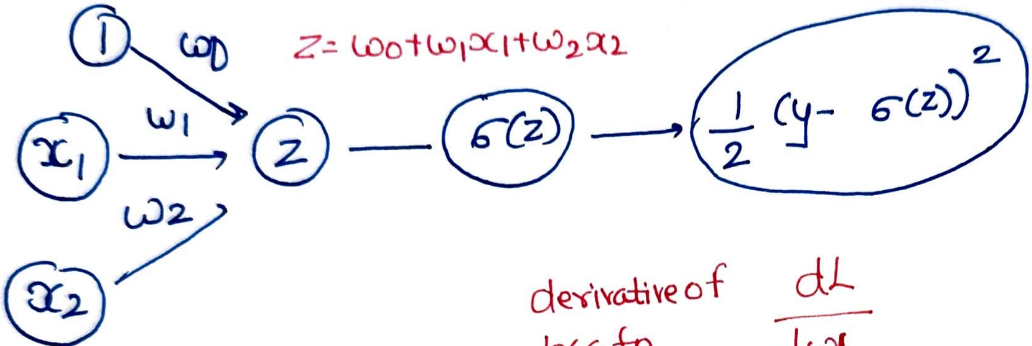
Speed of Tesla w.r.t Bmw 2x

$\frac{dy}{du} = 2$] 200m/h

Bmw (onground) $\frac{du}{dx} = 100$] chain rule

Tesla (ground) → find that

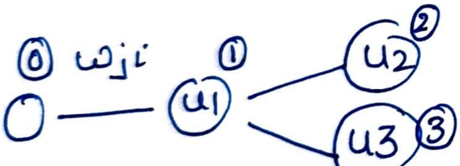
$Z = w_0 + w_1x_1 + w_2x_2$



derivative of loss fn w.r.t w_1 $\frac{dL}{dw_1}$

Lets focus on $\frac{dL}{\partial \sigma(z)}$

Loss w.r.t activation



$\frac{du_2}{du_1}$ $\frac{du_3}{du_2}$ $\frac{du_1}{dw_{ij}}$

③ ② ①.

① $\frac{dL}{\partial \sigma(z)} = (-y - \sigma(z))$ ↳ single variable

Back Propagation → go to ⓩ — σ(z)

② $\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$

Algorithm (Automatic Derivations)

then move to 1st & 2nd Layer

③ $\frac{dz}{dw_1} = \frac{\partial(w_0 + w_1x_1 + w_2x_2)}{dw_1} = x_1$

Similar to Tesla, Bmw, ground

Now apply chain rule = $\frac{dL}{dw_1} = \frac{dL}{d\sigma} \cdot \frac{d\sigma}{dz} \cdot \frac{dz}{dw_1}$

(3) (2) layer (1) layer