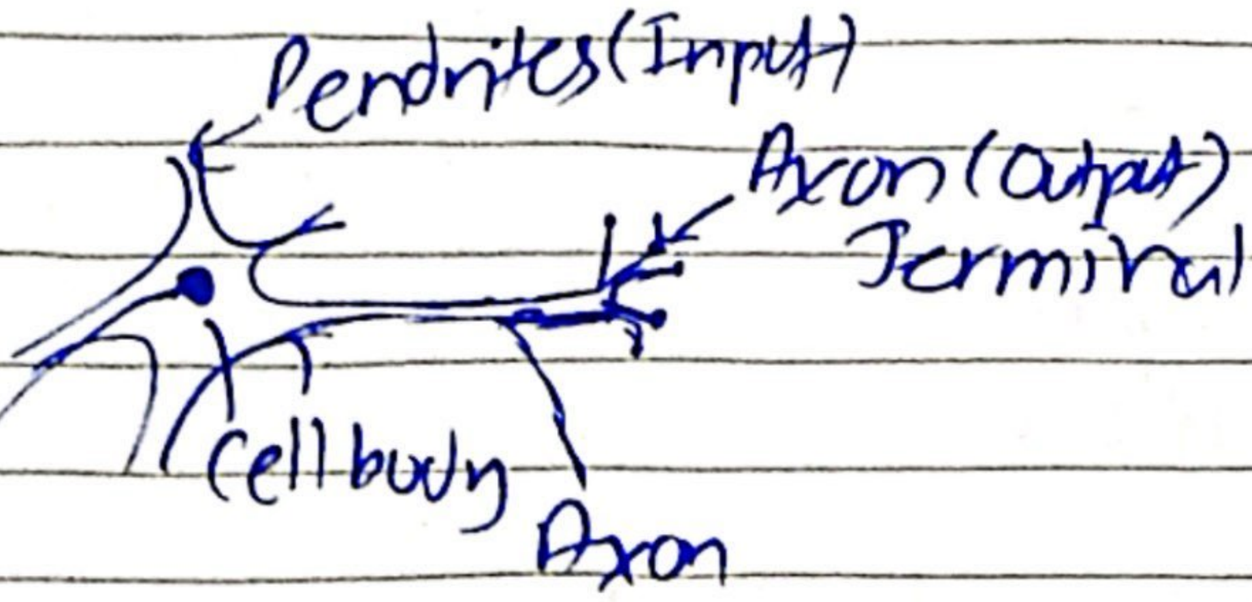
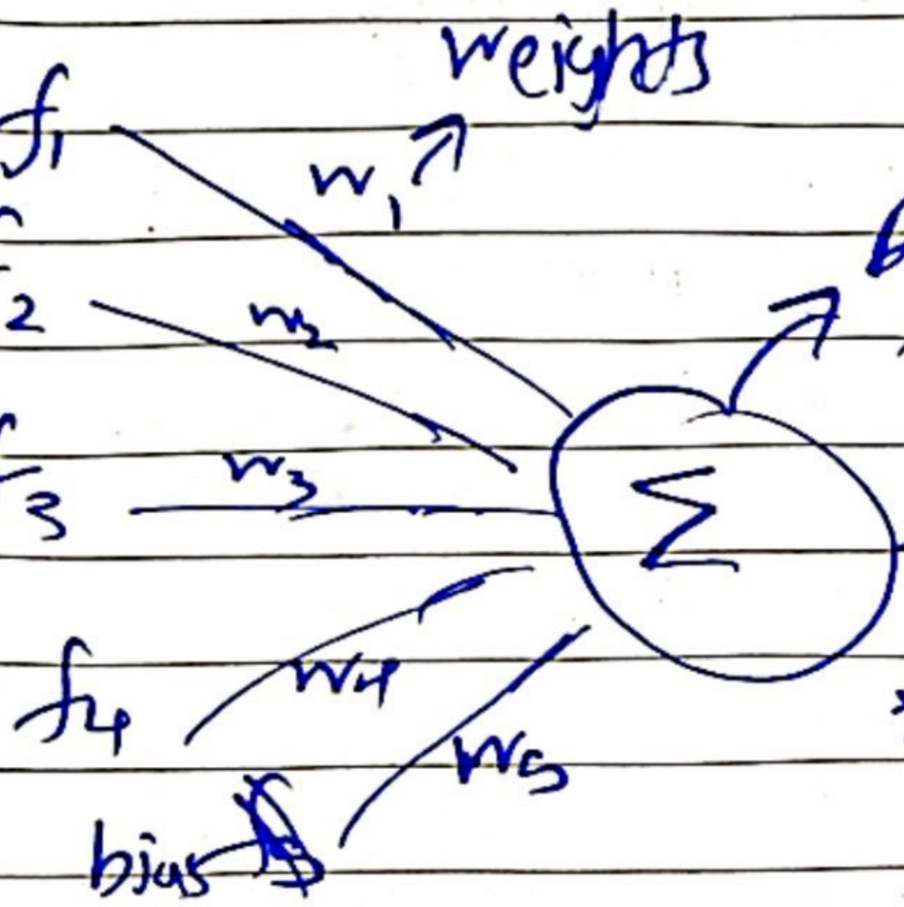
Content :-

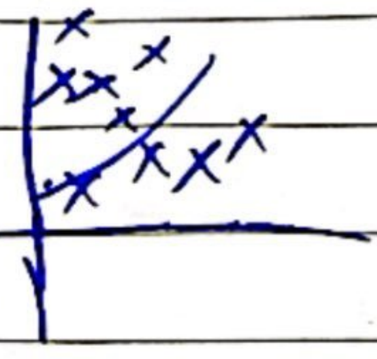
i) Neuron and Perceptron
ii) Perceptron Algorithm
iii) Learning Rate
iv) ~~Neural~~ ~~Artificial~~ Artificial Neural Network
v) Layers in deep learning
vi) Pooling Layer
vii) Margin and Padding
viii) Activation function types
ix) Optimisers in deep learning
x) Forward and Backward Propogation.
xi) Feature Extraction.



Neuron
* Input from Dendrites. Output
* Output to other neurons from axon

Counterpart :-
* Close to logistic regression





* Works well on linear tasks but not on non-linear tasks

Perceptron

f_1	f_2	f_3	f_4	Out

$\Sigma[(f_1 * w_1 + f_2 * w_2 + \dots + f_n * w_n) + bias]$

* Data inputs with different weights.
* Go through activation function.
* Gets an output.

Similarities :-

* Both have inputs, processing of inputs and resulting an output.
* Inputs in neuron are weighted based on strength of synapses while each input of a perceptron multiplied by it's weight representing the value.
* If neuron fires it transmits signals to other neurons. If activation function outputs, it is transmitted to the next layer in neural network.

Differences :-

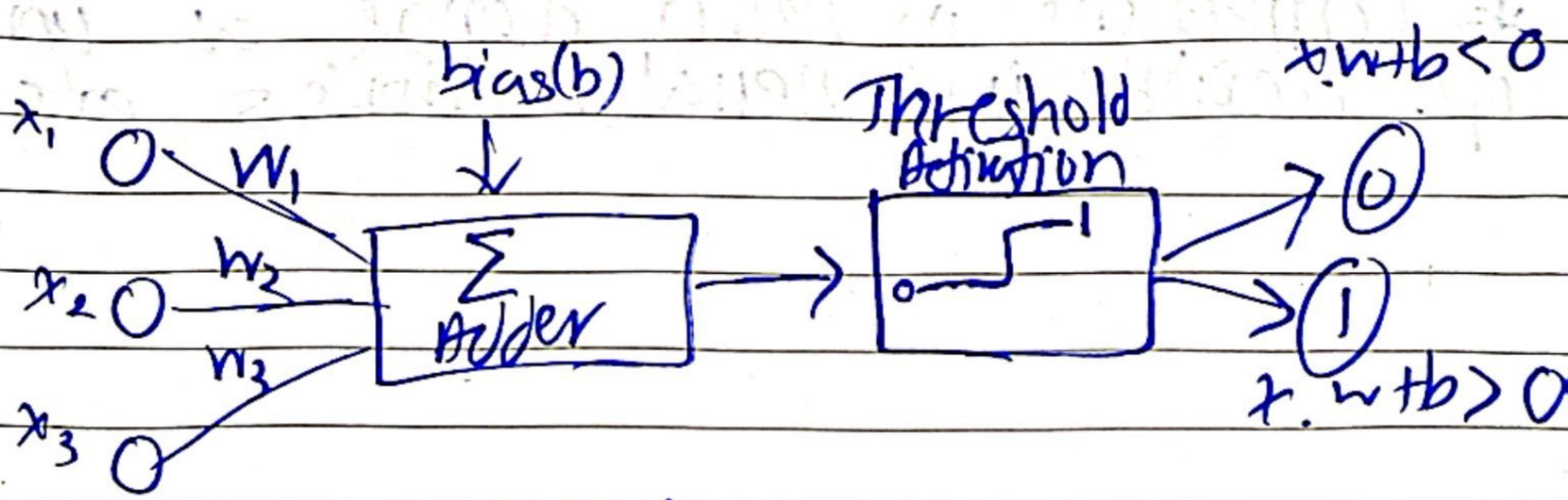
* Neuron is highly complex compared to perceptron.
* Lots of collections of inputs and neurons compared to inputs and output in perceptron.
* There is no mathematical function in neuron but there is an activation function in perceptron.
* Connections in neuron change as you grow up. Weights in neural networks are fixed.

07 Perceptron Algorithm:

- supervised learning algorithm
  - Binary classifiers

x	y

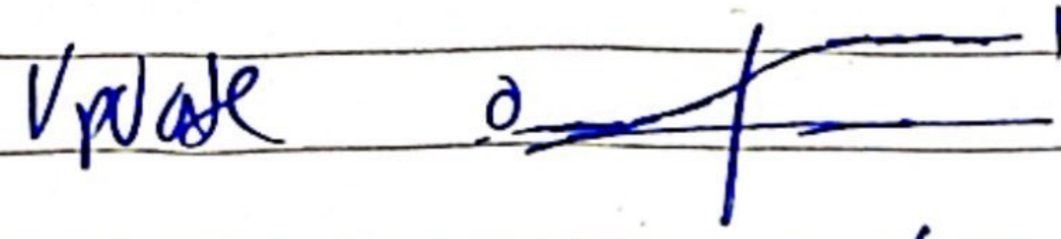
$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$



$$f(x) = \frac{1}{1+e^{x}}$$

$$r = x_1w_1 + x_2w_2 + \dots + x_nw_n + b$$
$$= \sum_{i=1} x_i w_i + b$$

$$r = xw + b$$



$$W_{new} = W_{old} + \alpha (y_i - \overline{y_i})$$

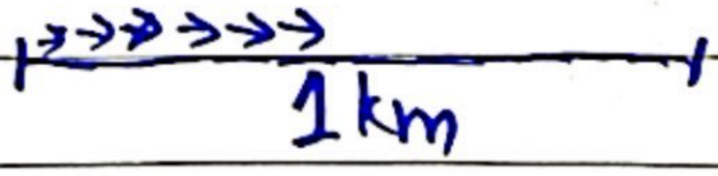
↑ high error

iii) Learning Rate :-

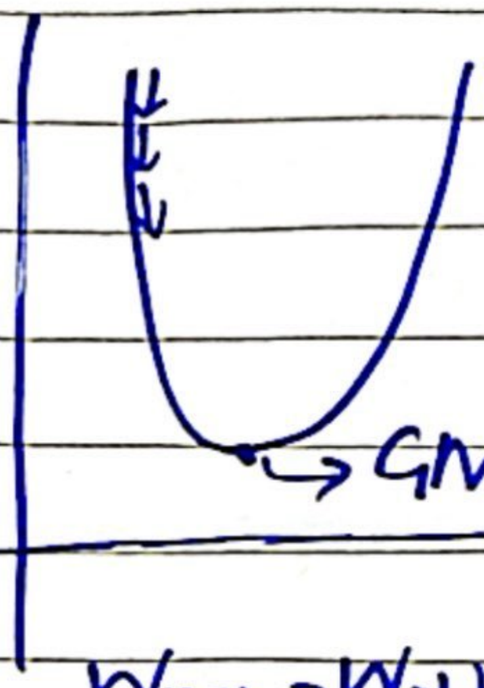
- We need weights where the loss is bare minimum. (Global minima).

~~Speed~~

Step Size :-



Small step size will take time to reach GM (Global Minima).
Bigger step size could cross past GM. Might stop at Local Minima (LM).



$W_{new} = W_{old} - (LR \times G$

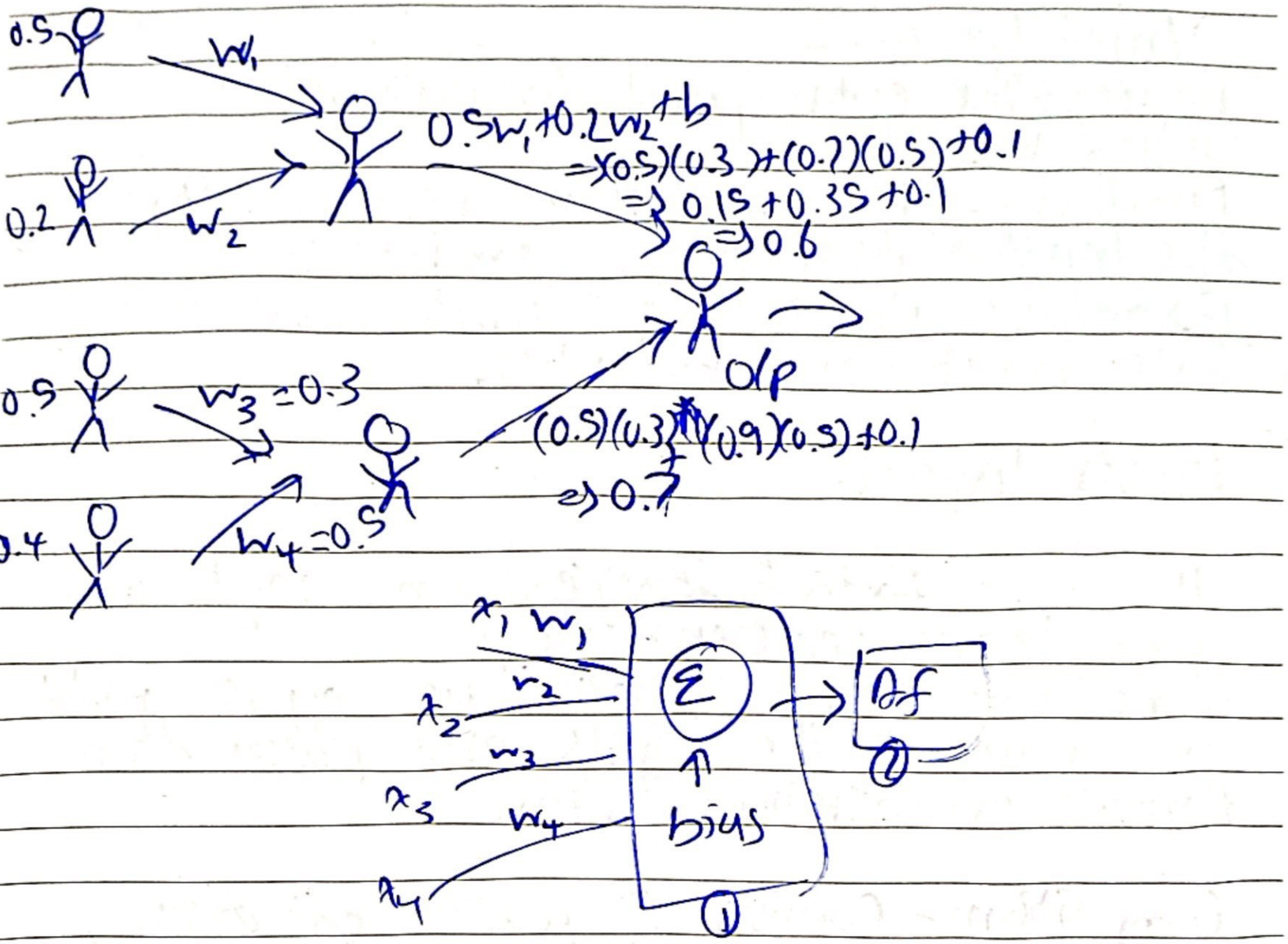
Gain :-
Is the loss increases after each step size, gain increases.

Speed :-
Speed of learning the model.

Rate :-
Rate of learning the model.

New weight (W_{new}) will determine if it corresponds better to the model

iv) Artificial Neural Network :-



- This is a neural network that is a computational model that mimics the way nerve cells work in a human brain.

Layers in deep Learning:-

Input Layer:-
Purpose:- The entry point for data into the neural network.
Function:- No computation is performed. It passes the input features to the next layer.
Example:- For image data, the input layer would consist of pixel values.

Hidden layers:-

Purpose:- Extract features from input data and learn representations.
Function:- Each hidden layer applies weights and biases to the inputs and passes them through an activation function.

Composition:- Consists of a series of dense, convolutional or recurrent layers.

Output Layer:-
Purpose:- Produce the final prediction or classification
Function:- Applies final transformation to map the network's features to the desired output.
Components:-
i) Activation function:-
» Sigmoid: Classification tasks
Linear: Regression tasks

ii) Output units: - Number of units corresponding to the number of classes or the number of outputs.

vi Pooling Layer :

- Crucial for convolutional neural networks
- Primarily used to reduce width and height of the input volumes, making computation more manageable and extracting dominant features.

ii) Max Pooling:

Purpose: To downsample the input while preserving the most important features (the strongest activations).

Function: For each patch of the input, the max. value is selected.

Operation:

- Input is divided into non-overlapping regions.
- The maximum value within each region is taken as the pooled value.

Advantages:-

Feature Preservation:- keeps most prominent feature

Computational Efficiency:- Reduces no. of parameters and computations.

Disadvantages:-

Information loss:- Discards less prominent features, which might be relevant.

Ex:- '[1,3), [2,4)]'

Max Pooling (2x2) : '4'

ii) Min Pooling:-

Purpose:- To downscale the input while preserving the least prominent features.

Function:- For each patch of the input, the minimum value is selected.

Operation:

- The input is divided into non-overlapping regions.
- Min value within each region is taken as pooled value.

Advantages:

- Background Features:- Preserves less prominent features that might be relevant for certain tasks.

Disadvantages:

Less common:- Rarely used compared to max pooling.

Information bias:- Could bias the network towards lower intensity features

Ex. '[[1,3],[2,4]]'

Min Pooling: '1'

viii: Margin and Padding :-

i) Margin :
Purpose :- . Provides spacing and seperation b/w image content and other elements.
. Ensures image content is not to close to the edges.

Types :
Inner Margin: Space b/w edge and visible border of the ~~paiag~~ image.
Outer Margin: Space b/w image border and surrounding elements in a layout.

Applications :-
- Document Layout
- Web Design
- Graphical User Interface (GUI).

Considerations :
Proportionality :- margin size should be proportional to the image size and the overall layout.
Consistency :- Maintaining consistent margin throughout a desig to ensure visual harmony

ii) Padding :-

Refers to adding pixels to the border of an image.

Purpose :-

- Preserve spatial dimensions after convolution operations.
- To handle edge cases where the convolutional filter extends beyond the image boundary.
- To avoid information loss at borders of the image.

Types :-

Zero Padding :-

~~Function~~ : Adds zeros to the borders of the image ~~size~~

Simple and effective for preserving dimensions.

Can introduce artificial edges, potentially affecting learning process.

Ex: [[1, 2], [3, 4]]

[[0, 0, 0, 0], [0, 1, 2, 0], [0, 3, 4, 0], [0, 0, 0, 0]]

Reflective Padding:-
Mirrors the border elements of the image.
Reflects edge ~~effect~~ by ~~increasi~~ extending actual image values.
~~AA~~ Computationally more intensive than zero padding.

Ex: [[1,2][3,4]]

⇒ [4,3,4,3],[2,1,2,1],[4,3,4,3],[2,1,2,1]

Replicate Padding :-
~~Extends~~ Extends border pixels by repeating the edge values
Simple and reduces edge artifacts.
Can create artificial flat regions.

Ex: [[1,2],[3,4]]

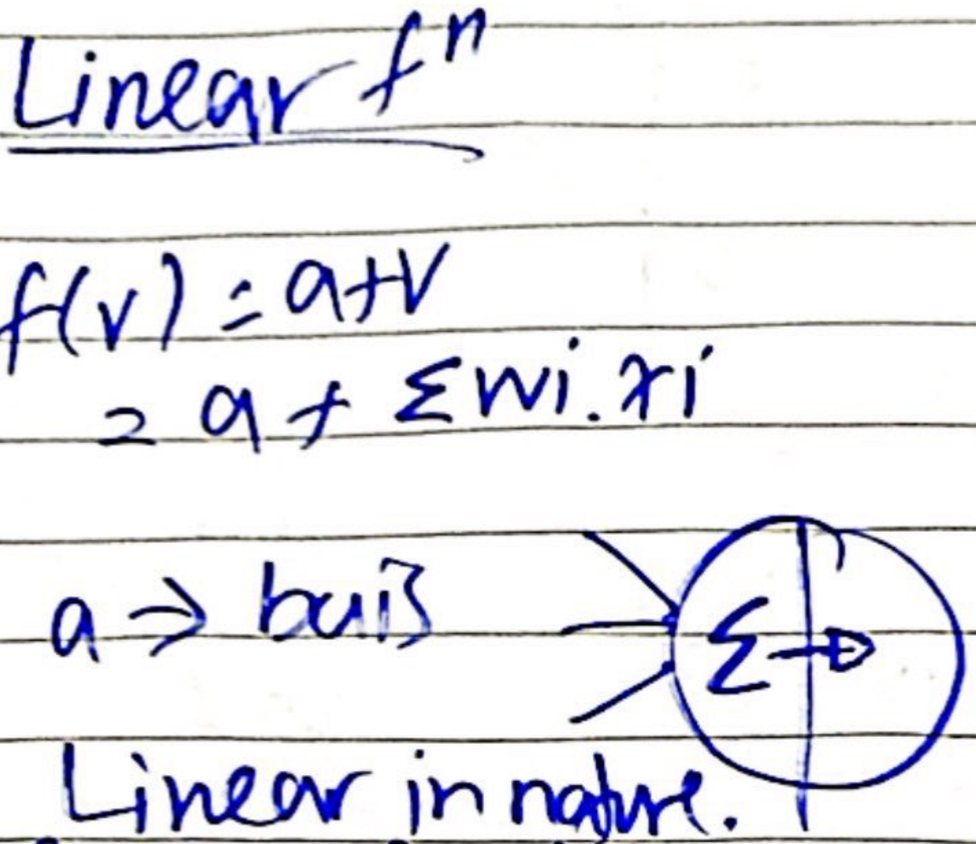
2)[1,1,2,2],[1,1,2,2],[3,3,4,4],[3,3,4,4]

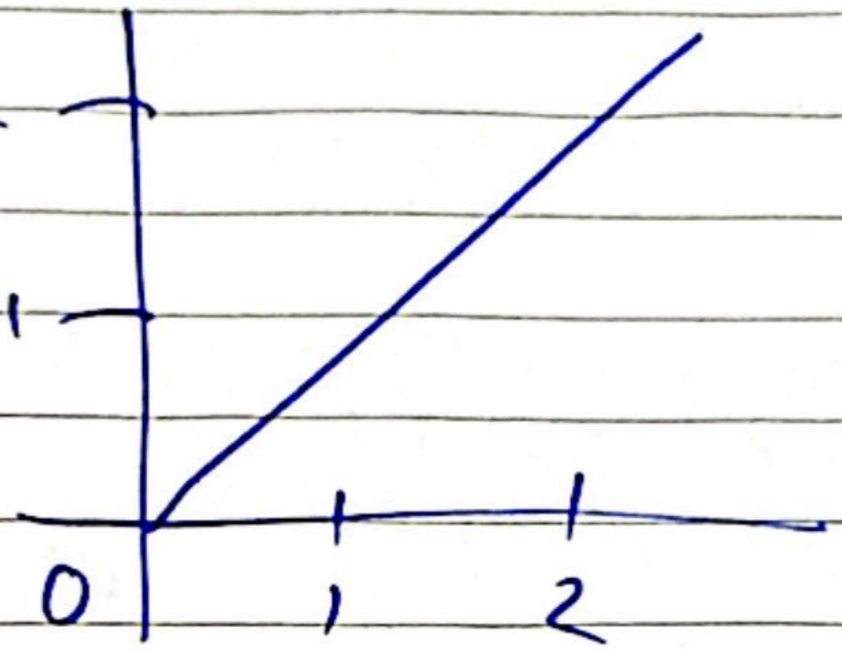
Applications:-
- Convolutional Neural Networks
- Edge detection and image filtering
- Image augmentation.

viii: Activation function types:-

i) Linear f^n

$f(v) = a + v$
$= a + \Sigma wi.xi$





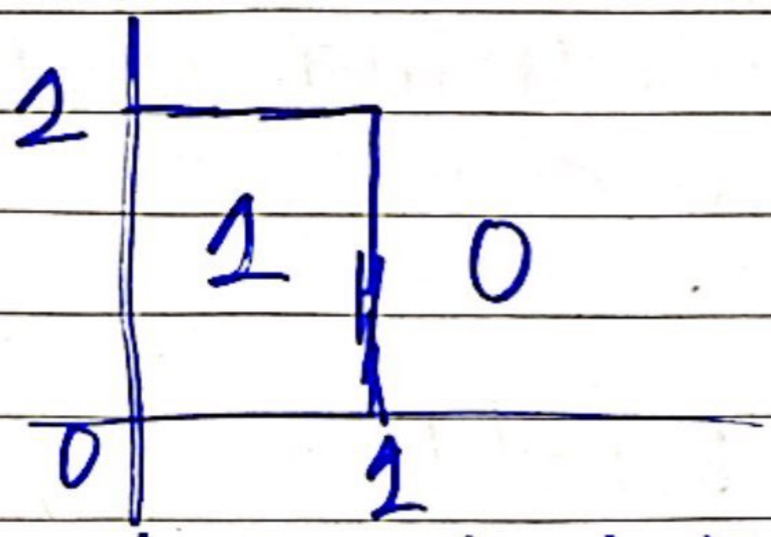
Linear in nature.
Simple function with no conditions.

iii)

ii) Heaviside step f^n

$f(v) = \begin{cases} 1 & \text{if } v \geq a \\ 0 & \text{otherwise} \end{cases}$

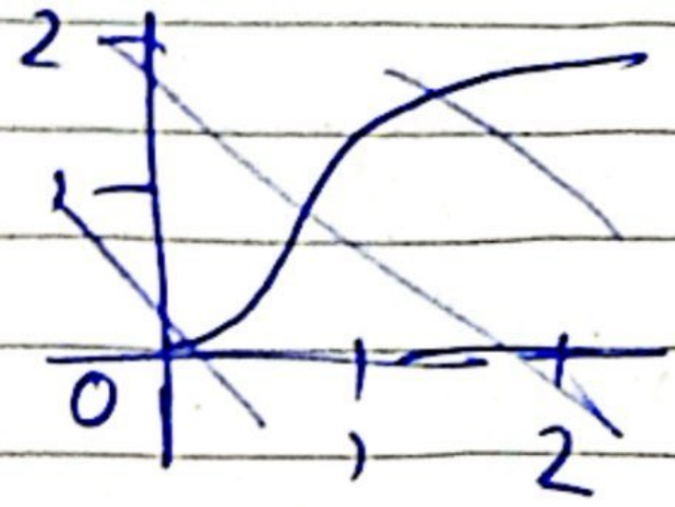
a → Threshold



Uses some conditions.
Only gives state 1 or state 0.
When threshold is crossed, signal is activated.

iii) Sigmoid f^n

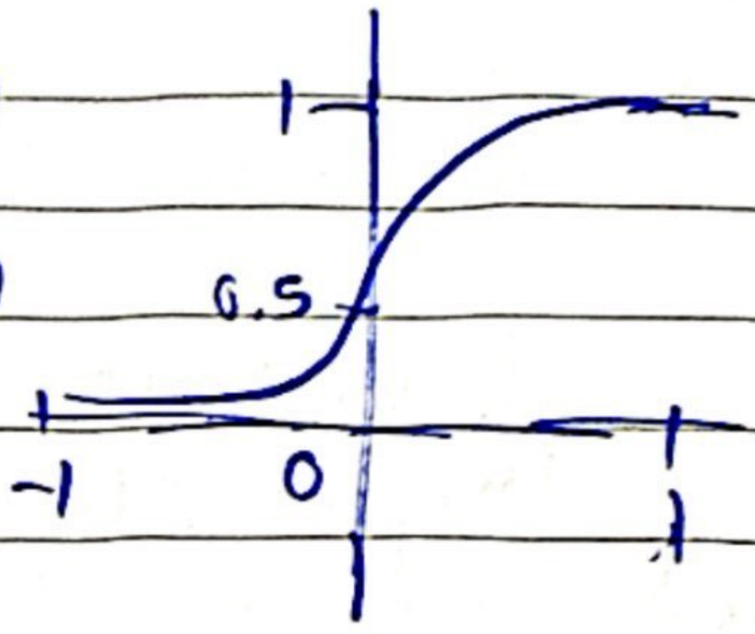
$$f(v) = \frac{1}{1+e^{-v}}$$



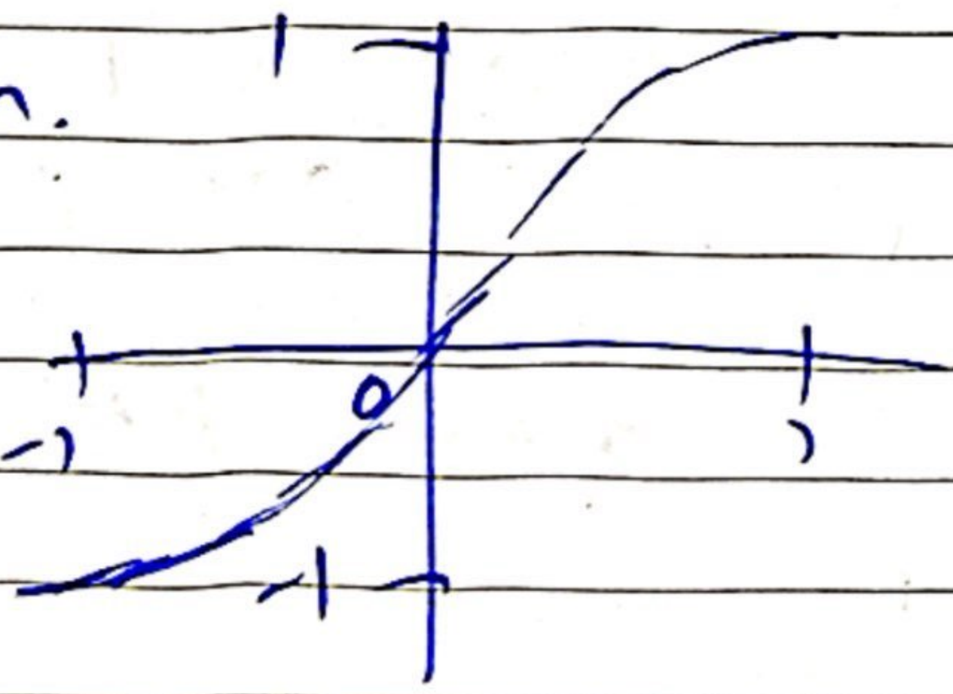
- Vanishing gradient function.
- More complex.
- Weighted sum is considered as '-v'
- ~~Curvature~~ forms a curvature from 0 to 1
- Non zero centered activation function

~~iv) tanh~~

iv) tanh



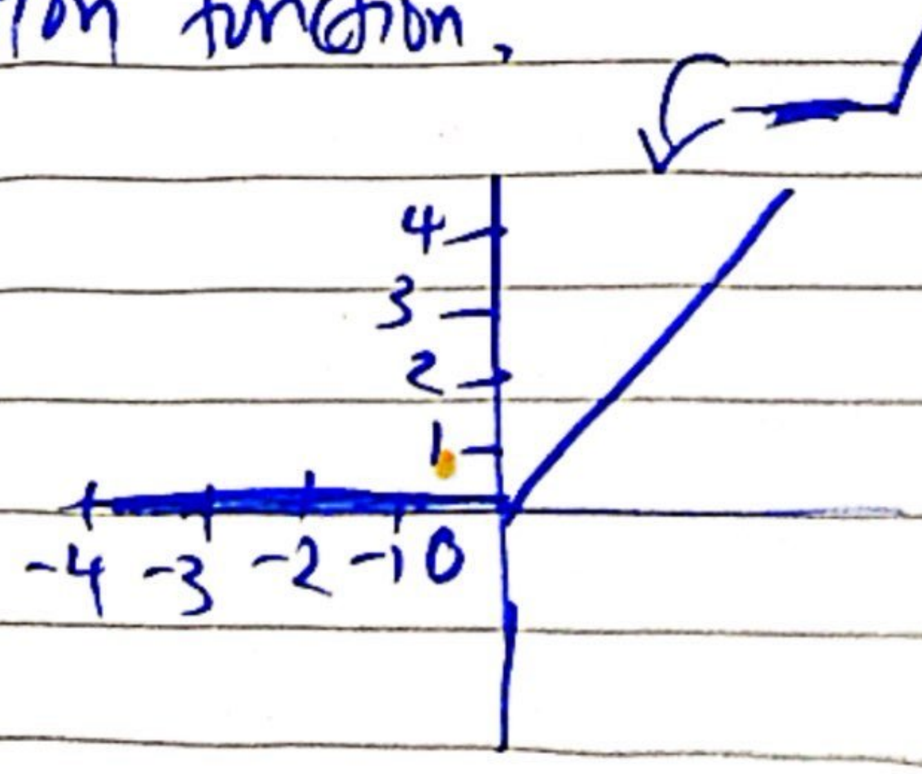
- Hyperbolic tangent function.
- zero centred function
- ~~Form~~ from -1 to +1.
- Vanishing gradient function.



v) RELU

- Rectified linear unit activation function.

$$f(x): \begin{cases} x_i & \text{if } x_i \geq 0 \\ 0 & \text{if } x_i < 0 \end{cases}$$



ix Optimisers in deep learning:-

Graddient descent:-

Batch GD:

* Prouses entire dataset in one go.
* ensures gradients are accurate
* ~~Motes~~ effective for large dataset.
* Computation resources should be sufficient.

Ex: Training on large well curated dataset like Image Net where full batch updates are feasible.

~~*~~ Stochastic GD:

* Updates weights after each training ex.
* introduces noise to help escape local minima - find ~~better~~ solutions.
* Suitable for realtime applications where immediate feedback is required.

Ex: Online learning for representation model, where user interactions are continuosly fed into the model.

Mini Batch GD:

* Balances efficiency of Batch GD
* balances robustness of SGD
* updates weights after processing small batches of data.
* Reduces noise and computation burden

Ex: General purpose training on minimum or large datasets, like training a CNN on CIFAR-10, where full batch processing is impractical but batch updates are efficient.

ii) Momentum:-
* Accelerates SGD.
* Adds fraction of prev. update to new update.
* helps to navigate along relevant direction.
* dampens oscillations.

Ex: Image classification where convergence speed is critical, like recognizing handwritten digits in a dataset.

iii) Nesterov Accelerated G:
* Looks the future position
* makes updates more informed.
* Faster convergence compared to momentum.

Ex: Training deep neural networks where precise adjustments to learning path are beneficial - depp CNN in image recognition.

iv) Adagrad:
* learning rate -
* Based on freq. of parameter updates.
* Effective for sparse data
*
Ex: Natural Language processing tasks where certain words occur infrequently but carry significant meaning.

v) RMSprop:
* Maintains moving avg. of sq. gradients
- adapts learning rate.
* Effective for non-stationary data.
* faster convergence.
Ex: RNN for time-series prediction where data distribution can change over time.

vi) Adaptive Moment Estimation:
* Combines benefits RMSprop and momentum - by using adaptive rates and momentum
* Robust and efficient - wide range of tasks.

Ex: General purpose optimiser for deep learning models in various domains; object detection in images, where efficiency and availability are crucial

vii) Adadelta:

* Extends Adagrad — by restricting the window of accumulated past gradients to a fixed size.
* Addresses diminishing learning rate issue.

Ex: training neural networks for speech recognition, where different layers might need different learning rates

viii) Nadam:

* More precise updates
* Faster convergence.
* Combines Adam and NAG.

Ex: ResNet models where overfitting can be a concern due to complexity of the model.

ix) AdamW:

* Incorporates weight decay in Adam
* helps in regularization
* Prevents overfitting adding penalty on large weights.

Ex: Faster convergence models where both speed and precision are critical to achieving high quality results.

x) ~~Feature~~

x) Forward and Baolcnard Propogation:-

i) Forward Propogation:

Input to thiden Layer:-

* Compute weighted sum of inputs.
* Add a bias term.
* Apply an activation function

hidden to Output Layer:-

* Compute weighted sum of hidden layer and add bias.
* Apply an activation function to produce a final output.

1) Hidden Layer:

$$h1 = f(w_1 . x1 + w_2 . x_2 + b1)$$

$$h2 = f(w_3 x1 + w_4 x_2 + b_2)$$

$$h_3 = f(w_5 . x_1 + w_6 . x_2 + b_3)$$

2) Output Layer:

$$O1 = f(w_7 . h_1 + w_8 . h_2 + w_9 h_3 + b_4)$$

ii) Backward Prop Propogation:-

1) Compute Loss:
Difference between predicted and actual output.

2) Calculate G:
Compute gradients of the loss with respect to each weight using the chain rule of calculus.

3) Update weights:
Adjust the weights and biases in the direction that reduces the loss

i) Output Error :

Loss = L(predicted output, actual output)

Eq Error term for outer layer = $\delta_o = \frac{\partial L}{\partial o1}$

ii) Hidden Layer Error :-

$\delta_{h1} = \delta_o . w_7 . f'(w_7 . h_1 + w_8 . h_2 + w_9 . h_3 + b_4)$

$\delta_{h2} = \delta_o . w_8 . f'(w_7 . h_1 + w_8 . h_2 + w_9 . h_3 + b_4)$

$\delta_{h3} = \delta_o . w_9 . f'(w_7 . h_1 + w_8 . h_2 + w_9 . h_3 + b_4)$

iii. Weight updates :

$w_7 = w_7 - \eta . \delta_o . h_1$

$w_8 = w_8 - \eta . \delta_o . h_2$

$w_9 = w_9 - \eta . \delta_o . h_3$

$w_{10} = w_{10} - \eta . \delta_{h1} . x_1$

$w_2 = w_2 - \eta . \delta_{h1} . x_2$

$w_3 = w_3 - \eta . \delta_{h2} . x_1$

$w_4 = w_4 - \eta . \delta_{h2} . x_2$

$w_5 = w_5 - \eta . \delta_{h3} . x_1$

$w_6 = w_6 - \eta . \delta_{h3} . x_2$

η = learning rate

δ = error terms (gradients)

xi) Feature Extraction :-

* Transforming raw data into a set of meaningful attributes.
  - used for learning machine learning tasks.
* Used to identify patterns, edges, textures an other important aspects of image.

Convolutional Layers :
* Most common type of feature extraction layer.
* Use convolution operations to scan image with set of learnable filters (kernels).

1. Convolution operation:
   - Small matrix slides over input image.
   - For each position, filter performs element-wise multiplication with the input image region it overlaps, and the results are summed up.
   - Sum is a single value in the output feature map.

2. Activation Function:
   - An activation function is applied to introduce non-linearity.

3. Pooling:
* Pooling layer might follow convolutional layer.
- reduces the spatial dimensions of the feature maps.
- retains most important information.

iii) Feature extraction and passing to neural network layer:

i) Stacking Convolutional Layers:
- Multiple layers are stacked to progressively extract more complex layers.
- Early layers might ~~extract~~ detect simple edges.
- Deeper layers capture intricate patterns and object parts.

ii) Flattening:
. Final set of maps is flattened into a on-dimensional vector.
. Fed into fully dense layers for further processing.

iii) Fully connected layers:
- Layers take the flattened vector as input.
- perform the final classification and regression task.

Ex:

i) Input image : 32x32x3 (RGB image)

ii) Convolutional Layer 1: 32 filters of size 3x3
- Output : 30x30x32 feature maps

iii) ReLU Activation

iv) Max Pooling Layer 2: 2x2 pooling
- Output : 15x15x32 feature maps

v) Convolutional Layer 2: 64 filters of size 3x3
- Output: 13x13x64 feature maps

vi) ReLU Activation

vii) Max Pooling Layer 2: 2x2 pooling
- Output : 6x6x64 feature maps

viii) Flattening
- Output : 2304-dimensional vector

ix) Fully Connected Layer: 128 neurons
- Output: 128-dimensional vector

x) Fully connected Layer (Output Layer): 10 neurons for 10 classes
- Output: Probability distribution over classes.