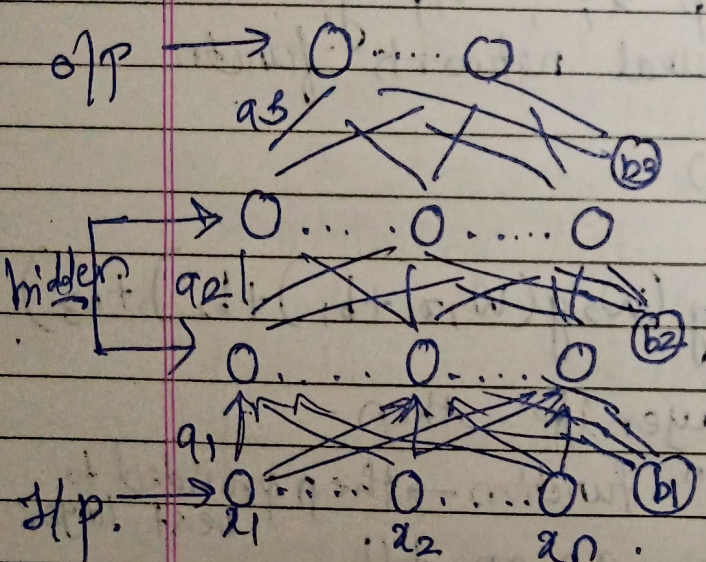
PPT 2

* Feedforward Neural Networks :- (multilayered NN).
→ feedforward :- connections between nodes, or 'neurons' do not form cycle. This means that inf. moves only in one direction : from input → hidden layers → output.

Key aspects :-
- i/p layer, hidden layer, o/p layer
- forward propagation
- Activation funcn
- Training.



- input to the network is an n-dimensional vector.
- contains L-1 hidden layers having n neurons each. (here 2 hidden layers)
- 1 o/p layer containing K neurons (corresponding K classes)
- Each neuron in hidden layer & o/p layer can be split into 2 parts :- activation, preactivation

(hi) activation, preactivation (ai)

i/p layer ⇒ 0th layer. o/p layer → Lth layer.
b1, b2, b3 are biases.

Preactivation at layer i is given by :-
$a_i(x) = b_i + W_i h_{i-1}(x)$.
(Wixi + bias).

activation at layer i :-

$h_i(x) = g(a_i(x))$

g is activation function. (Eg: tanh, linear, etc)

- The activation at o/p layer is given by $f(x) = h_L(x) = O(a_L(x))$

where O is o/p activation function (Eg :- softmax, linear, etc).

Model description :-

Data :- You have dataset with N samples, each sample consists of i/p x_i, o/p y_i.

model :- The model is neural network function

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

$$\hat{y}_i = f(x_i) = O(W_3 g(W_2 g(W_1 x_i + b_1) + b_2) + b_3)$$

↑ 1st layer i/p then activn function → then provided to next layer

→ then go on till o/p layer activation.

W_1, W_2, W_3 are weight matrices for each layer

b_1, b_2, b_3 are bias vectors.

g is activation function applied to o/p of each layer

O : final activation function or o/p layer transformation

* Parameters (θ) : $\theta = \{W_1, W_2, W_3, b_1, b_2, b_3\}$

* Total no. of parameters :—

$$P = (n \times n + n) \times (L-1) + (n \times K + K)$$

(n: bias, L: no. of layers, K: o/p layer has K neurons.)

* Algorithm used for this model :— Gradient Descent with Back Propogation.

◦ Objective / Loss function :— Goal of training the n/w is to find parameters θ that minimize the error betn predicted o/ps ($\hat{y}_i$) and true o/ps (y_i).

$$\min\left(\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K}(\hat{y}_{ij} - y_{ij})^2\right)$$

↪ mean square error funct^n :

◦ minimize loss function $L(\theta)$.

* The loss function $L(\theta)$ could take different forms depending on specific problem (Eg: mean sq. error for regression or cross-entropy for classification)

* Learning Parameters of Feedforward Neural Networks :—

◦ Gradient Descent Algorithm :—

GD is an optimization algorithm used to minimize the loss function of a model by iteratively adjusting its parameters.

Steps :-

1] Initialization :-

Set initial Parameters : initialize model parameters (weight & biases) randomly. This is θ_0.

$$\theta_0 = [w_1^0, w_2^0, \ldots w_L^0, b_1^0, b_2^0 \ldots b_L^0]$$

w_i^0 & b_i^0 are initial values for each layer.

2] Set hyperparameters :-

- Learning Rate (η): Determines step size for each iteration.
- maximum Iterations: A stopping criteria for algo (Eg: 1000 iterations).

3] Iteration Loop :-

1. compute gradients :-

forward pass :- compute predictions $\hat{y}_i$ using (θ_t).

Compute loss : Calculate loss $L(\theta_t)$ based on predicted & true values.

Backward pass :- Use backprop to compute gradient of loss funct w.r.t each parameter.

for a parameter θ_j in θ_t :-

$$\nabla_{\theta_j} L(\theta_t) = \frac{dL(\theta_t)}{d\theta_j^0}$$

These gradients include → gra. of weights w_i^0 and biases b_i for each layer.

2. Update Parameters :-

Adjust each parameter θ_j by moving it in the direction of negative gradient

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} L(\theta_t)$$

↳ vector of all gradients at iteration t

3 Increment counter.

Repeat until max no. of iterations is reached or changes in parameters become very small (convergence).

$$\nabla_\theta L(\theta) = \left[\frac{dL(\theta)}{d\theta_1}, \frac{dL(\theta)}{d\theta_2}, \ldots, \frac{dL(\theta)}{d\theta_n}\right]$$

- Algorithm :- gradient_descent(·)

$t \leftarrow 0$
max_iterations $\leftarrow$ 1000
Initialize $\theta_0 = [w_1^0, \ldots, w_L^0, b_1^0, b_2^0, \ldots b_L^0]$
While $t{+}{+} <$ max_iterations do
  $\theta_{t+1} \leftarrow \theta_t - \eta \nabla\theta_t$;
end

Where $\nabla\theta_t = \left[\frac{dL(\theta)}{dw_{1t}}, \ldots, \frac{\partial L(\theta)}{\partial w_{Lt}}, \frac{\partial L(\theta)}{\partial b_{1,t}}, \ldots \frac{\partial L(\theta)}{\partial b_{Lt}}\right]^T$

* Output functions and Loss functions :-

Loss functions for different types of problems :-

1. Regression Problem (Predicting ratings) :-
Ex: you want to predict ratings, where output $\hat{y}_i$ is a real number.
Loss fun^n: Squared error loss is suitable for this type of problem. It measures the squared deviation bet^n true & pred value.

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{3}(\hat{y}_{ij} - y_{ij})^2$$

2. Classification Problem :-
Ex: you want to classify image into one of K classes, so the true output y_i is a probability

distribution. Softmax funcn is used here:-

Output funcn:-

$$O(a_L)_j = \frac{e^{a_{Lj}}}{\sum_{i=1}^{K} e^{a_{Li}}}$$

This converts raw scores into probabilities.

Loss funcn:- cross entropy loss is suitable for classification problems. It measures how well predicted prob distribution matches true distribution.

$$L(\theta) = -\sum_{c=1}^{K} y_c \log \hat{y}_c$$

y_c is 1 for true class, 0 otherwise.

$$L(\theta) = -\log \hat{y}_{true}$$

where $\hat{y}_{true}$ is predicted probability for true class.

* Choice of o/p function:-

- for regression:- o/p funcn should be linear bec you want pred values $\hat{y}_i$ to be real numbers.
- for classifn:- o/p funcn should be softmax to ensure that o/ps are probabilities (they sum to 1 and are all betn 0 & 1).

- Log-Likelihood: for classification problem, the term $\log \hat{y}_{true}$ represents the log-likelihood of data given the model parameters.

Maximizing this log-likelihood helps in finding parameters θ that best fit the training data.

	o/p's	
	Real values	Prob.
o/p Activn	linear	Softmax
Loss fun^c	Squared error	Cross Entropy

* Back Propogation Algorithm :-

- Gradient descent :- Iteratively updates the network parameters to minimize loss by using gradients computed through forward & backward propogation.
- Steps seen earlier.

- Forward Prop. Algo :-

Purpose: To compute network's predictions & intermediate values during forward pass.

1. Layer by layer computation:
   For each layer k from 1 to $L-1$:
   Calculate a_k by adding bias to weighted sum of activations from previous layer.

$$a_k = b_k + W_k h_{k-1}$$

   Compute h_k by activating - applying activation function g to a_k :-

$$h_k = g(a_k)$$

2. O/p layer :- Compute final layer's pre-activation

$$a_L = b_L + W_L h_{L-1}$$

$$\hat{y} = o(a_L) \quad o \rightarrow \text{o/p activation function}$$

- Back Propogation :- Computes gradients of loss w.r.t each parameter by propagating errors backward from o/p to i/p layer.

Calculate gradient of loss function

$$\nabla_{a_L} L(o) = -(y - \hat{y})$$

Page No.
Date.

Propagate gradient backwards:-

- Gradient w.r.t weights

$$\nabla_{W_k} L(\theta) = \nabla_{a_k} L(\theta) h_{k-1}^T$$

(gradients of pre activation x activations from prev layer)

- Gradient w.r.t bias:-

$$\nabla_{b_k} L(\theta) = \nabla_{a_k} L(\theta)$$

grad of pre-activn a_k

- Gradient w.r.t prev layer's Activations:

$$\nabla_{h_{k-1}} L(\theta) = W_k^T \nabla_{a_k} L(\theta)$$

- " " Pre layer's Be activn:-

$$\nabla_{a_{k-1}} L(\theta) = \nabla_{h_{k-1}} L(\theta) \odot g'(a_{k-1})$$

Page No.
Date.

How to compute g' :—

1. Logistic funcn :—

$$g(z) = \sigma(z)$$

$$= \frac{1}{1+e^{-z}} \qquad \left(\frac{d}{dx}\frac{1}{x} = \frac{-1}{x^2}\right).$$

$$g'(z) = \frac{-1}{(1+e^{-z})^2} \frac{d}{dz}(1+e^{-z})$$

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

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

$$= g(z) \cdot (1-g(z)).$$

2. tanh

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

$$g'(z) = \frac{(e^z+e^{-z})\frac{d}{dz}(e^z-e^{-z}) - (e^z-e^{-z})\frac{d}{dz}(e^z+e^{-z})}{(e^z+e^{-z})^2}$$

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

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

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

$$= \underline{\underline{1-(g(z))^2}}$$