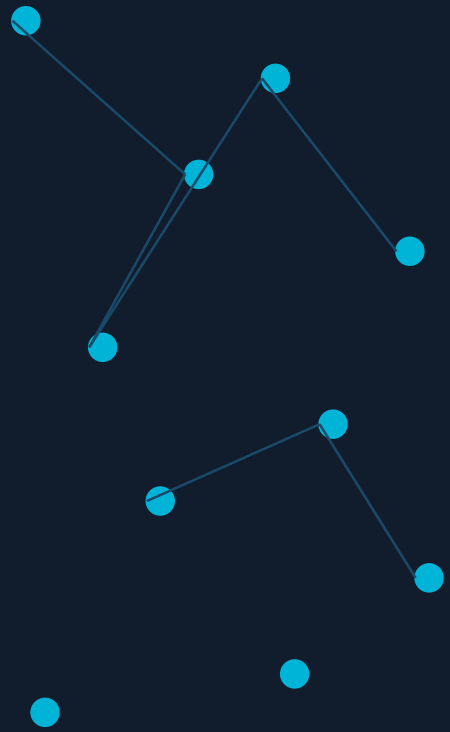
The Perceptron, Backpropagation & Its Variants

Neural Networks — Lecture Module

Topics Covered: Single Layer Perceptron · Perceptron Learning Algorithm · Convergence Theorem
Delta Rule · MLP Networks · Backpropagation · Learning Rate Analysis · BP Variants



Course Page: https://www.gcjana.in/courses/shardauniversity/2502/CSA203/

Lecture Outline

01 **Single Layer Perceptron Network**

Architecture, inputs, weights, activation function

02 **Perceptron Learning Algorithm**

Weight update rule and convergence theorem

03 **Delta Rule**

Gradient-based learning strategy

04 **Limitations of Single Layer Perceptron**

XOR problem & linear inseparability

05 **Multilayer Perceptron (MLP) Network**

Hidden layers, universal approximation

06 **Backpropagation Learning**

Forward pass, error, backward pass & applications

07 **Effect of Learning Rate**

Underfitting, oscillation, adaptive methods

08 **Variants of Backpropagation**

Momentum, Adam, RMSProp, and more

01

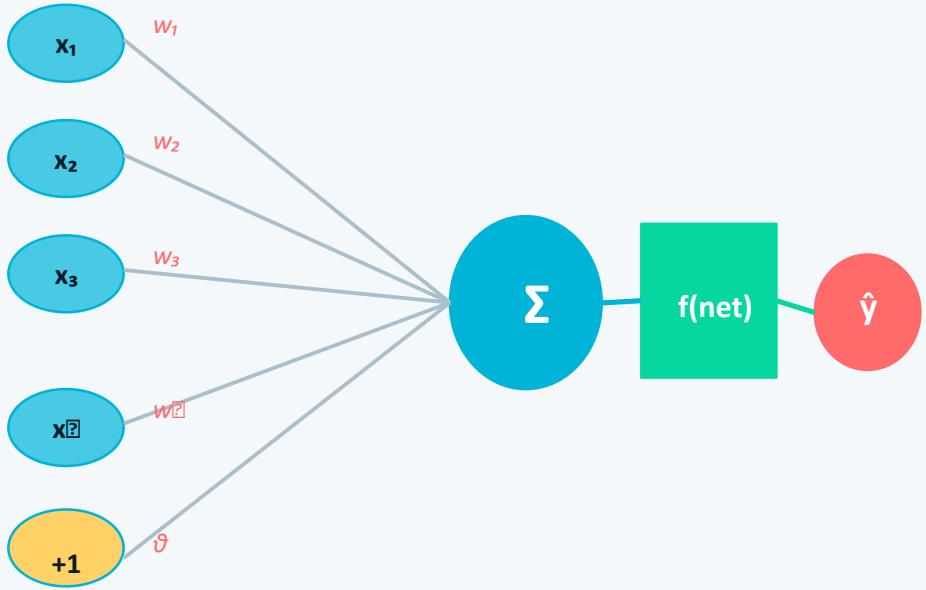
Single Layer Perceptron Network

Architecture · Biological inspiration · Mathematical model

Single Layer Perceptron — Architecture

Biological Inspiration

- Inspired by biological neurons in the brain
- Dendrites → Input signals (x_1, x_2, …, x_n)
- Synaptic weights → Connection strengths (w_1, w_2, …, w_n)
- Cell body → Summation of weighted inputs
- Axon → Output based on activation function



Mathematical Model:

net = Σ w_i x_i + θ | Output y = f(net) | Step activation: y = 1 if net ≥ 0, else y = 0

Activation Functions in Perceptron

Step Function

y = 1 if net≥0, else 0

Classic Perceptron activation.
Outputs hard binary 0 or 1.
Not differentiable at 0.

Sign Function

y = +1 if net≥0, else −1

Bipolar variant of step.
Used in Hebb-style learning.
Outputs −1 or +1.

Linear Function

y = net (identity)

Used in delta rule.
Differentiable everywhere.
Gradient is constant = 1.

Sigmoid Function

$y = 1 / (1 + e^{-net})$

Smooth & differentiable.
Output in (0,1).
Key for backpropagation.

02

Perceptron Learning Algorithm

Weight update rule · Convergence theorem

Perceptron Learning Algorithm

1 Initialize

Set all weights $w_i = 0$ (or small random values)
Set bias $\theta = 0$ and choose learning rate η $(0 < \eta \leq 1)$

2 Present Input

Feed input pattern $x = (x_1, x_2, \ldots, x_n)$ with desired output d
Compute net $= \Sigma\, w_i x_i + \theta$, then $\hat{y} = f(\text{net})$

3 Compute Error

error $e = d - \hat{y}$
If $e = 0$: no update needed. If $e \neq 0$: update weights

4 Update Weights

$w_i(\text{new}) = w_i(\text{old}) + \eta \cdot e \cdot x_i$
$\theta(\text{new}) = \theta(\text{old}) + \eta \cdot e$

5 Repeat

Present all training patterns (one epoch)
Repeat until all patterns correctly classified

Key Rule: $w_i(\text{new}) = w_i(\text{old}) + \eta \cdot (d - \hat{y}) \cdot x_i$ where η = learning rate, d = desired output, $\hat{y}$ = actual output

Perceptron Convergence Theorem

Statement

"If a set of training patterns is linearly separable, the Perceptron Learning Algorithm is guaranteed to converge to a solution in a finite number of steps."

— Rosenblatt (1962)

✓ Key Conditions

1. Patterns must be linearly separable
2. Learning rate η must be positive and finite
3. Training set must be finite

⚠ Proof Sketch

Bound the number of mistakes using the margin δ (min distance to decision boundary). Misclassification count ≤ $(R/\delta)^2$ where R = max ||x||

✕ Limitation

Theorem DOES NOT guarantee convergence for non-linearly separable patterns — the algorithm may cycle indefinitely

03

Delta Rule

Gradient-based learning strategy · Least Mean Square

Delta Rule — Gradient-Based Learning

What is the Delta Rule?

- Also called the Widrow-Hoff rule or LMS (Least Mean Square) rule
- Minimises Mean Squared Error (MSE) using gradient descent
- Works with continuous/linear activation (unlike step fn)
- Forms the mathematical foundation of backpropagation
- Error surface is a paraboloid → single global minimum

Derivation

Error: $E = \frac{1}{2} (d - \hat{y})^2$

Gradient: $\partial E / \partial w_i = -(d - \hat{y}) \cdot x_i$

Gradient descent: $\Delta w_i = -\eta \cdot \partial E / \partial w_i$

Delta Rule: $\Delta w_i = \eta \cdot (d - \hat{y}) \cdot x_i$

$= \eta \cdot \delta \cdot x_i$ where $\delta = (d - \hat{y})$

Comparison: Perceptron Rule vs Delta Rule

	Perceptron Rule	Delta Rule
Activation	Step / threshold	Linear (continuous)
Convergence	Only if linearly separable	Always (gradient descent)
Error minimised	Classification error	Mean Squared Error

04

Limitations of Single Layer Perceptron

Linear separability · XOR problem

Limitations of Single Layer Perceptron

Core Limitation: Linear Separability

- A Single Layer Perceptron can only learn linearly separable patterns
- The decision boundary is always a hyperplane: $\Sigma w_i x_i + \theta = 0$
- Cannot solve problems where classes cannot be separated by a straight line

XOR Truth Table

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

Cannot Solve XOR

XOR is not linearly separable — no single line can separate 0s and 1s. Proved by Minsky & Papert (1969).

No Hidden Layers

Lacks intermediate representations. Cannot learn complex/hierarchical features present in real-world data.

Step Fn Not Differentiable

Step activation has zero or undefined gradient, preventing gradient-based optimization strategies.

Only Binary Output

Produces hard 0/1 decisions. Cannot model probabilities or continuous outputs needed in regression.

05

Multilayer Perceptron (MLP) Network

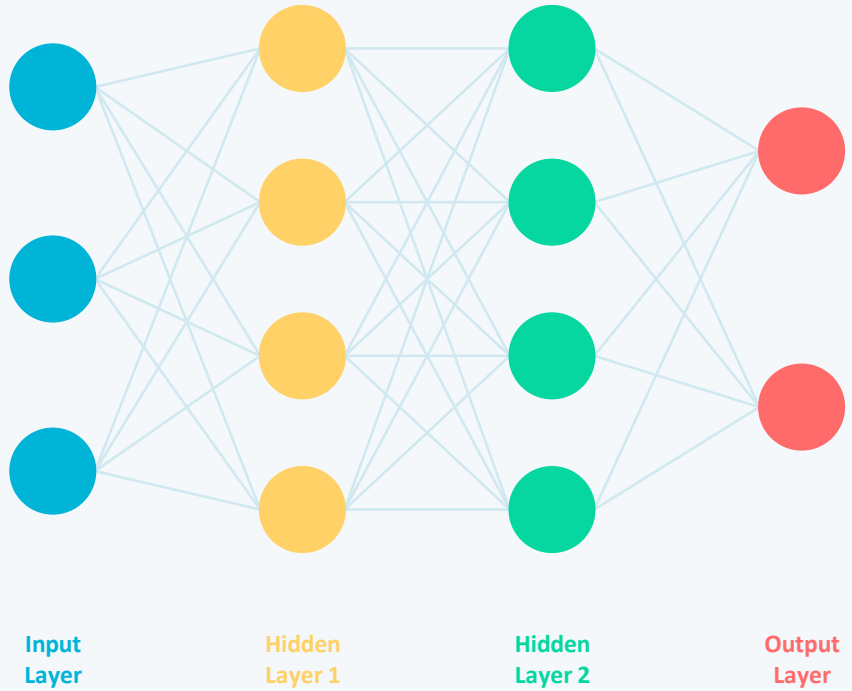
Hidden layers · Universal approximation theorem

Multilayer Perceptron — Architecture

- Input Layer: receives raw features (no computation)
- Hidden Layer(s): transform inputs non-linearly; number & size are hyperparameters
- Output Layer: produces final predictions
- Each neuron connected to all neurons in adjacent layers (fully connected)
- Non-linear activation enables learning of complex boundaries

Universal Approximation Theorem (Hornik, 1989):

A single hidden layer MLP with enough neurons can approximate any continuous function to arbitrary accuracy.



06

Backpropagation Learning

Forward pass · Error computation · Backward pass · Applications

Backpropagation Algorithm

Phase 1: Forward Pass

- Present input x and propagate forward through all layers
- Compute activations: $a^l = f(W^l \cdot a^{l-1} + b^l)$ at each layer l
- Record output $\hat{y} = a^L$ at the output layer

Phase 2: Error Computation

- Compute loss $L = \frac{1}{2} \Sigma (d_i - \hat{y}_i)^2$ (MSE) over output neurons
- Compute output layer delta: $\delta^L = (d - \hat{y}) \cdot f'(net^L)$

Phase 3: Backward Pass

- Propagate deltas backward: $\delta^l = (W^{l+1T} \cdot \delta^{l+1}) \odot f'(net^l)$
- Compute weight gradients: $\partial L/\partial W^l = \delta^l \cdot (a^{l-1})^T$

Phase 4: Weight Update

- Update all weights: $W^l \leftarrow W^l - \eta \cdot \partial L/\partial W^l$
- Repeat for all training samples until convergence

Core idea: Chain Rule — $\partial L/\partial w_{ij} = \partial L/\partial a_j \cdot \partial a_j/\partial net_j \cdot \partial net_j/\partial w_{ij} = \delta_j \cdot a_i$

Backpropagation — Applications

Image Recognition

CNNs trained via BP classify objects, detect faces, perform medical imaging diagnosis (X-rays, MRI).

Natural Language Processing

Text classification, sentiment analysis, machine translation trained using BP on RNNs/Transformers.

Game Playing & RL

Deep Q-Networks and policy gradients use BP to train agents that master chess, Go, and video games.

Time Series Forecasting

LSTMs trained via BP predict stock prices, weather, energy demand, and network traffic.

Speech Recognition

Deep neural networks convert audio waveforms to text; used in Siri, Alexa, Google Assistant.

Robotics & Control

Neural controllers trained via BP enable adaptive manipulation, locomotion, and path planning.

07

Effect of Learning Rate

Convergence speed · Stability · Adaptive methods

Analysing Effect of Learning Rate (η)

The learning rate η controls the size of weight updates at each step. Its value critically affects training dynamics.

η too small (η → 0)

Very slow convergence
May get stuck in local minima
Needs many epochs
High computational cost

✓ η optimal (η ≈ ideal)

Fast and stable convergence
Reaches global/good minimum
Smooth loss curve decrease

η too large (η >> 1)

Overshoots minimum
Oscillates around optimum
May diverge entirely
Unstable training

Adaptive Learning Rate Methods:

Learning Rate Decay: $\eta(t) = \eta_0 / (1 + \text{decay} \cdot t)$ | **Cyclical LR:** oscillate between min/max values | **Warmup:** start low, increase, then decay

08

Variants of Backpropagation Algorithm

Momentum · Adaptive methods · Modern optimisers

Backpropagation Variants — Gradient Descent Types

Batch Gradient Descent

$$\Delta W = -\eta \cdot (1/N) \Sigma \nabla L(x_i, y_i)$$

✓ Stable convergence, accurate gradient estimate

✕ Very slow for large datasets; needs full pass per update

Stochastic Gradient Descent (SGD)

$$\Delta W = -\eta \cdot \nabla L(x_i, y_i) \quad \text{[per sample]}$$

✓ Fast updates; can escape local minima due to noise

✕ Noisy, oscillatory convergence; high variance

Mini-Batch Gradient Descent

$$\Delta W = -\eta \cdot (1/m) \Sigma \nabla L \quad \text{[batch of m]}$$

✓ Balances speed and stability; enables GPU parallelism

✕ Batch size is a hyperparameter; slightly noisier than batch

Backpropagation Variants — Advanced Optimisers

Momentum

$$v = \beta \cdot v + \eta \cdot \nabla L$$
$$W \leftarrow W - v$$

Accumulates velocity in gradient direction. Typical $\beta = 0.9$. Reduces oscillation, accelerates convergence in flat regions.

Nesterov Accelerated Gradient (NAG)

$$v = \beta \cdot v + \eta \cdot \nabla L(W - \beta v)$$
$$W \leftarrow W - v$$

Look-ahead variant of Momentum. Evaluates gradient at anticipated future position. Faster convergence than vanilla momentum.

AdaGrad

$$G \mathrel{+}= (\nabla L)^2$$
$$W \leftarrow W - (\eta / \sqrt{G + \varepsilon}) \cdot \nabla L$$

Per-parameter adaptive LR. Larger LR for rare features. G accumulates, causing LR → 0 over time (limitation).

RMSProp

$$G = \rho G + (1 - \rho)(\nabla L)^2$$
$$W \leftarrow W - (\eta / \sqrt{G + \varepsilon}) \cdot \nabla L$$

Fixes AdaGrad's decay by using exponential moving average of squared gradients. $\rho \approx 0.9$. Effective for RNNs.

Adam (Adaptive Moment Estimation)

$$m = \beta_1 m + (1 - \beta_1) \nabla L$$
$$v = \beta_2 v + (1 - \beta_2)(\nabla L)^2$$
$$W \leftarrow W - \eta \cdot \hat{m} / \sqrt{\hat{v} + \varepsilon}$$

Combines Momentum + RMSProp. Uses bias-corrected 1st & 2nd moment estimates. β_1=0.9, β_2=0.999. Most widely used optimiser.

AdamW / Nadam

AdamW: adds L2 weight decay
Nadam: Adam + Nesterov

AdamW decouples weight decay from gradient update (better regularisation). Nadam combines Adam with look-ahead gradient. State of the art.

Optimiser Comparison Summary

Quick-reference guide to choosing an optimiser for your task:

Optimiser	Adaptive LR	Momentum	Memory Cost	Best For
SGD	No	No	Low	Simple/linear models
SGD+Momentum	No	Yes	Low	CV tasks with careful tuning
AdaGrad	Yes	No	Medium	Sparse features / NLP
RMSProp	Yes	No	Medium	RNNs, non-stationary problems
Adam	Yes	Yes	High	General-purpose; most tasks
AdamW	Yes	Yes	High	Transformers, LLMs

Key Takeaways — Course Summary

1. Single Layer Perceptron

Simple binary classifier; weights updated via Perceptron rule; convergence guaranteed for linearly separable data.

2. Delta Rule

Gradient descent on MSE with linear activation; foundation of modern backpropagation; always converges.

3. Limitations

SLP cannot solve XOR or any non-linearly separable problem; motivated the need for multilayer networks.

4. Multilayer Perceptron

Hidden layers + non-linear activations enable learning of arbitrary complex functions (universal approximator).

5. Backpropagation

Efficient chain-rule computation of gradients across all layers; enables training deep networks end-to-end.

6. Learning Rate & Variants

η critically controls training; advanced optimisers (Adam, RMSProp) use adaptive LR for faster, stable training.