UNIT 2
Single Layer & Multilayer Perceptron (MLP)

CONTENTS

- Adaptive Filtering Problem
- Unconstrained Organization Techniques
- Linear Least Square Filters
- Least Mean Square (LMS) Algorithm
- Learning Curves
- Learning Rate Annealing Techniques
- Perceptron Convergence Theorem
- Relation Between Perceptron and Bayes Classifier (Gaussian Environment)

CONTENTS

- Relation Between Perceptron and Bayes Classifier (Gaussian Environment)
- Backpropagation Algorithm
- XOR Problem
- Heuristics
- Output Representation and Decision Rule
- Computer Experiment
- Feature Detection

Adaptive Filtering Problem

- **What is Adaptive Filtering?**
- **Definition:** Adaptive filtering involves creating a filter whose coefficients (weights) change dynamically based on input-output data to minimize an error criterion (usually mean squared error).
- **Goal:** Make the system learn the unknown relationship between input and output without prior knowledge of the system model

Adaptive Filtering Problem

- **Mathematical Formulation**
- Suppose we have:
  - Input vector: $x(i)=[x1(i),x2(i),\ldots,xm(i)]T$
  - Desired output: $d(i)$
  - Filter output: $y(i)=wTx(i)$
  - Error: $e(i)=d(i)-y(i)$

  Adjust weights w to minimize E[e(i)2], the mean squared error

Adaptive Filtering Problem

- **Techniques in ANN for Adaptive Filtering**
- **ADALINE (Adaptive Linear Neuron):**
  - Similar to perceptron but uses a linear transfer function.
  - Learns using the **Least Mean Squares (LMS)** rule.
- **LMS Algorithm (Widrow-Hoff Rule):**
  - Iteratively updates weights:
- $w_{k+1}=w_k+\eta \cdot e_k \cdot x_k$
- Simple, efficient, widely used in adaptive signal processing

Adaptive Filtering Problem

- **Applications**
- **Noise cancellation:** Removing unwanted background signals.
- **Channel equalization:** Correcting distortions in communication channels.
- **System identification:** Modeling unknown systems using input-output data.
- **Signal prediction:** Forecasting future values of signals

Adaptive Filtering Problem

- **Key Challenges**
- **Convergence speed:** Depends on learning rate.
- **Stability:** Too high learning rate can cause divergence.
- **Nonlinear environments:** Single-layer adaptive filters may fail; multilayer perceptrons or advanced algorithms are needed.

COMPARISION

Aspect	Perceptron	ADALINE (Adaptive Filter)
Output	Binary (0/1)	Continuous (linear)
Learning Rule	Perceptron rule	LMS rule
Problem Solved	Linearly separable classification	Adaptive filtering, signal processing
Flexibility	Limited	More powerful, minimizes MSE

Unconstrained Organization Techniques

- **Definition:** These are optimization methods used in ANN where the objective function (loss/cost) is minimized without restrictions on the values of weights or biases.
- **Contrast with Constrained Optimization:**
  - *Constrained*: Parameters must satisfy certain conditions (e.g., non-negativity, bounded ranges).
  - *Unconstrained*: No restrictions; weights can take any real values to achieve the best fit.

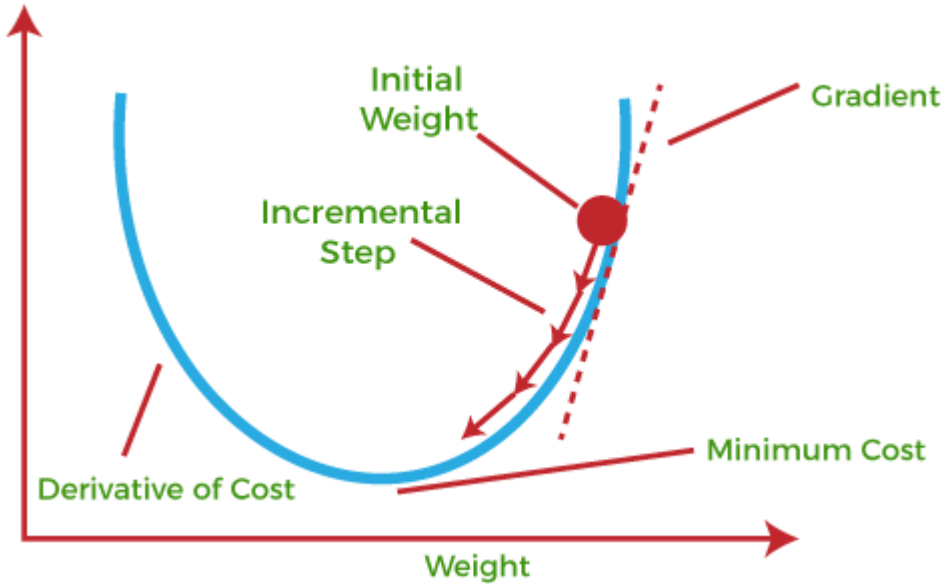
Unconstrained Organization Techniques

Role in ANN Training

- Neural networks are trained by minimizing a **loss function** (e.g., mean squared error, cross-entropy).
- In unconstrained optimization, the algorithm searches the entire parameter space for the best solution.
- Commonly used in **gradient-based learning** methods such as:
  - **Gradient Descent (GD)**
  - **Stochastic Gradient Descent (SGD)**
  - **Momentum-based methods**
  - **Adaptive optimizers** (Adam, RMSProp, Adagrad)

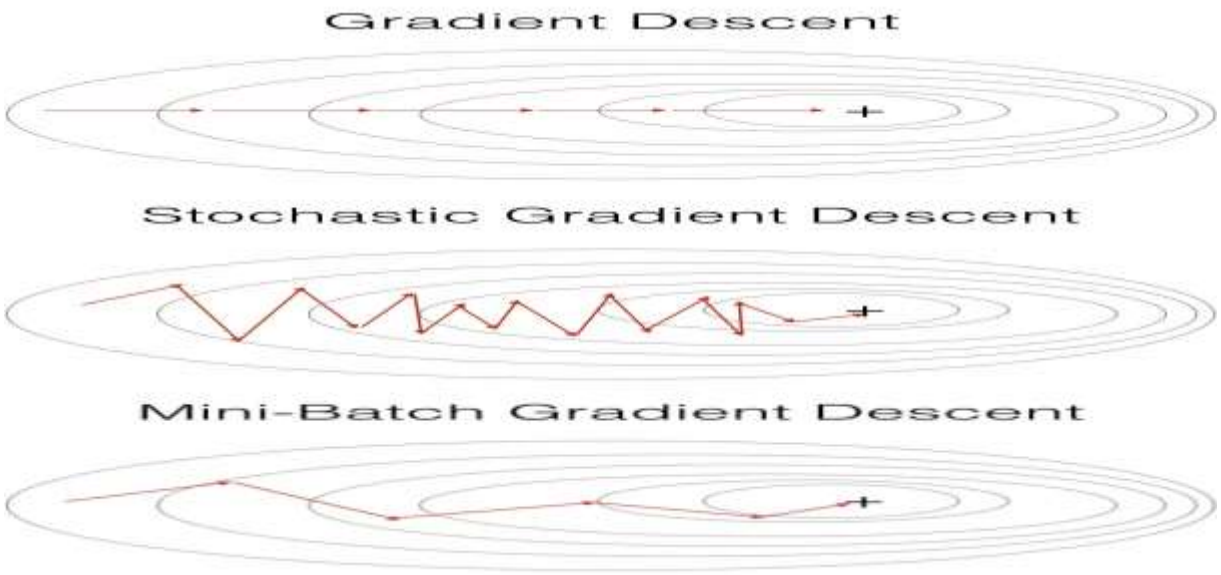
Unconstrained Organization Techniques

- **Gradient Descent**
  - Iteratively updates weights in the opposite direction of the gradient.
  - Formula:
- $w_{k+1} = w_k - \eta \cdot \nabla J(w_k)$
- where J(w) is the loss function.



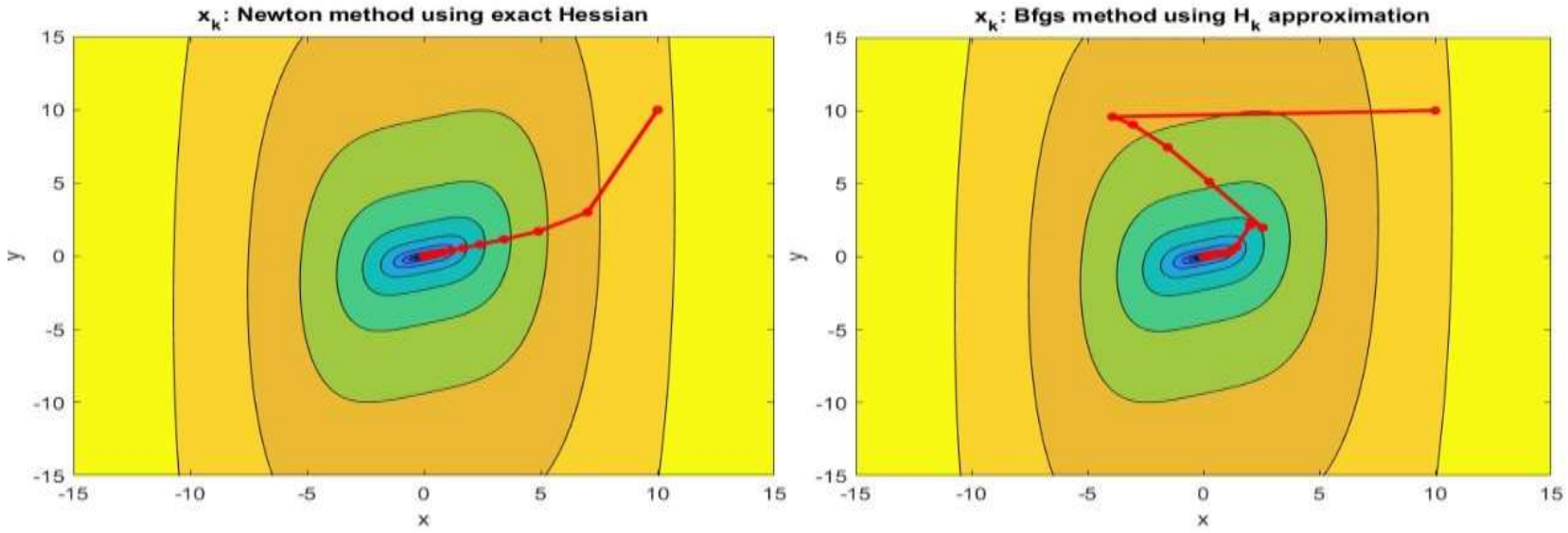
Unconstrained Organization Techniques

- Stochastic Gradient Descent (SGD)
- Uses a single sample or mini-batch per update.
- Faster and widely used in deep learning.



Unconstrained Organization Techniques

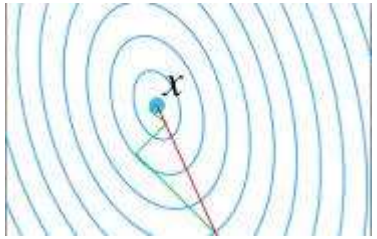
- **Newton’s Method / Quasi-Newton Methods**
- Use second-order derivatives (Hessian matrix) for faster convergence.
- More computationally expensive.



Unconstrained Organization Techniques

Conjugate Gradient Method

- Avoids direct computation of Hessian.
- Efficient for large-scale problems.



comparision of unconstrained organization techniques

Technique	Nature	Pros	Cons
Gradient Descent	First-order	Simple, widely used	Slow convergence
SGD	First-order	Fast, scalable	Noisy updates
Newton's Method	Second-order	Faster convergence	High computation
Conjugate Gradient	Second-order approx	Efficient for large problems	Complex implementation

Linear Least Square Filters

- **Definition**
- A **Linear Least Square Filter** is a filter that minimizes the **sum of squared errors** between the desired output and the actual filter output.
- It’s based on the **least squares criterion**, a fundamental optimization method in statistics and machine learning

Linear Least Square Filters

- **Mathematical Formulation**
- Input vector: x(i)=[x1(i),x2(i),…,xm(i)]T
- Desired output: d(i)
- Filter output: y(i)=wTx(i)

Objective:

$$J(w) = \sum_{i=1}^{N} (d(i) - w^T x(i))^2$$

- Minimize $J(w)$ with respect to weights w.
- Solution (Normal Equation):

$$w = (X^T X)^{-1} X^T d$$

where X is the input data matrix and d is the desired output vector.

Linear Least Square Filters

Applications

- **Signal prediction**
- **System identification**
- **Noise cancellation**
- **Channel equalization**
- **Data fitting/regression problems** in machine learning.

Advantages

- Simple and mathematically elegant.
- Provides optimal solution under Gaussian noise assumptions.
- Foundation for more advanced adaptive filtering techniques.

Limitations

- Requires matrix inversion → computationally expensive for large datasets.
- Sensitive to ill-conditioned matrices (when XTX is nearly singular).
- Not suitable for real-time adaptation (hence LMS algorithm is preferred).

Least Mean Square (LMS) Algorithm

- The **LMS algorithm** is an **iterative optimization technique** used to adjust the weights of a filter or neuron to minimize the **mean squared error (MSE)** between the desired output and the actual output.
- It’s a practical alternative to the **Linear Least Squares** method, which requires matrix inversion

Least Mean Square (LMS) Algorithm

- Input vector: $x(i) = [x_1(i), x_2(i), \ldots, x_m(i)]^T$
- Desired output: $d(i)$
- Filter output: $y(i) = w^T x(i)$
- Error: $e(i) = d(i) - y(i)$

Weight update rule:

$$w_{k+1} = w_k + \eta \cdot e_k \cdot x_k$$

Where:

- w_k = weight vector at iteration k
- η = learning rate (step size)
- e_k = error at iteration k
- x_k = input vector at iteration k

Least Mean Square (LMS) Algorithm

Key Characteristics

- **Adaptive:** Weights change dynamically with each new input.
- **Simple:** Requires only multiplication and addition (no matrix inversion).
- **Real-time capable:** Suitable for online learning and streaming data.
- **Applications**
- **Noise cancellation** (removing unwanted background signals).
- **Channel equalization** (correcting distortions in communication systems).
- **System identification** □ (modeling unknown systems).
- **Prediction tasks** (forecasting signals, time series).

Advantages

- Easy to implement.
- Low computational complexity.
- Works well in real-time adaptive systems.

Least Mean Square (LMS) Algorithm

- **Limitations**
- Convergence speed depends heavily on learning rate η.
- Too large η → instability/divergence.
- Too small η → slow convergence.
- Sensitive to non-stationary environments (where data distribution changes).
- COMPARISION

Method	Nature	Pros	Cons
Linear Least Squares	Closed-form	Exact solution, optimal	Expensive, not adaptive
LMS Algorithm	Iterative	Simple, adaptive, real-time	Approximate, slower convergence

Learning Curves in ANN

Definition

- A **learning curve** is a plot that shows how the performance of a learning algorithm improves (or changes) with experience.
- Typically, it represents **error (or accuracy)** versus **number of iterations, epochs, or training samples**.

Typical Axes

- **X-axis:** Training iterations / epochs / number of samples.
- **Y-axis:** Error (Mean Squared Error, Cross-Entropy) or Accuracy.

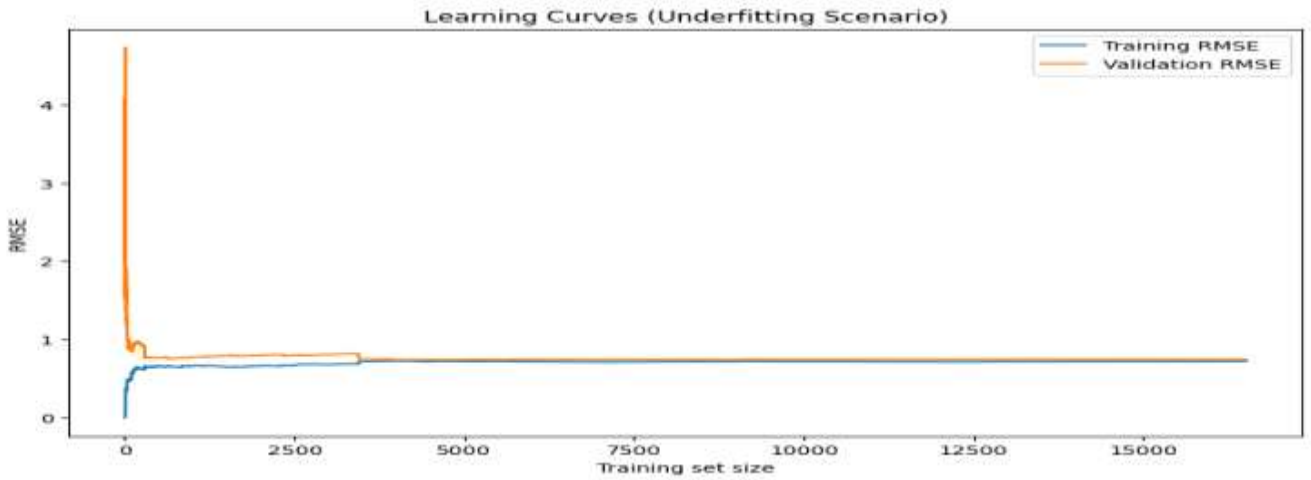
Learning Curves in ANN

Training Error Curve

Shows how error decreases on the training set as learning progresses.

Validation Error Curve

- Shows how error behaves on unseen data (validation set).



Learning Curves in ANN

- **Interpretation**
- **Steady decline in error:** Algorithm is learning effectively.
- **Plateau:** Learning has saturated; weights are no longer improving.
- **Gap between training and validation curves:** Indicates **overfitting** (model memorizes training data but fails on new data).
- **Slow convergence:** Learning rate may be too small.
- **Oscillations:** Learning rate may be too high.

Learning Curves in ANN

Interpretation

- **Steady decline in error:** Algorithm is learning effectively.
- **Plateau:** Learning has saturated; weights are no longer improving.
- **Gap between training and validation curves:** Indicates **overfitting** (model memorizes training data but fails on new data).
- **Slow convergence:** Learning rate may be too small.
- **Oscillations:** Learning rate may be too high.
- **Applications**
- **Monitoring training progress** in neural networks.
- **Comparing algorithms** (which converges faster, which generalizes better).
- **Hyperparameter tuning** (learning rate, batch size, network depth).
- **Detecting underfitting/overfitting**.

Learning Curves in ANN

Example Curve Behaviors

- **Good learning:** Training and validation errors both decrease and stabilize at low values.
- **Overfitting:** Training error decreases, but validation error increases after some point.
- **Underfitting:** Both errors remain high, indicating the model is too simple.

Curve Behavior	Meaning
Both training & validation errors ↓	Good learning
Training ↓, validation ↑	Overfitting
Both high	Underfitting
Oscillations	Learning rate too high
Very slow decrease	Learning rate too low

Learning Rate Annealing Techniques

What Is Learning Rate Annealing?

- **Learning rate annealing** refers to strategies that **gradually reduce the learning rate** during training to improve convergence and stability.
- Inspired by **annealing in metallurgy**, where materials are slowly cooled to remove defects.
- **Why Use Annealing?**
- A **high learning rate** helps the model learn quickly at the start.
- A **lower learning rate** later helps fine-tune weights and avoid overshooting the minimum.
- Prevents oscillations and improves generalization.

Learning Rate Annealing Techniques

Technique	Description	Example
Step Decay	Reduce learning rate at fixed intervals	Every 10 epochs: $\eta = \eta/2$
Exponential Decay	Reduce rate exponentially over time	$\eta_t = \eta_0 \cdot e^{-kt}$
Time-Based Decay	Reduce based on epoch count	$\eta_t = \eta_0/(1 + kt)$
Performance-Based Decay	Reduce when validation loss plateaus	If no improvement in 5 epochs, reduce η
Cosine Annealing	Use cosine function to vary rate	Smooth cyclical decay pattern

Perceptron Convergence Theorem

- **Statement:** If a training dataset is linearly separable, the perceptron learning algorithm will converge to a set of weights that correctly classifies all training examples in a finite number of iterations.
- **Linearly separable data:** There exists a hyperplane that separates the two classes without error.
- **Perceptron update rule:**
- $w_{k+1} = w_k + \eta \cdot y_i \cdot x_i$
- where:
- w_k = weight vector at iteration k
- η = learning rate
- y_i = true label ($+1$ or -1)
- x_i = input vector
- **Convergence condition:** The algorithm stops updating once all points are correctly classified.

Perceptron Convergence Theorem

- **Mathematical Insight**
- The number of updates is bounded by:

$$\frac{R^2 \cdot \|w^*\|^2}{\gamma^2}$$

- where:
- R = maximum norm of input vectors
- γ = margin (minimum distance from any point to the decision boundary)
- $w*$ = optimal weight vector
- This shows that convergence is faster when data is well-separated (large margin).

Perceptron Convergence Theorem

Implications

- **Guarantees success** for linearly separable problems.
- **Fails to converge** if data is not linearly separable (e.g., XOR problem).
- Forms the basis for more advanced models like **Support Vector Machines (SVMs)** and **Multilayer Perceptrons (MLPs)**.

Practical Example

- Imagine trying to classify cats vs dogs based on features like ear shape and tail length. If these features allow a clean separation (no overlap), the perceptron will eventually learn the correct boundary.

Perceptron Convergence Theorem Summary

Feature	Description
Applies to	Linearly separable data
Guarantees	Finite convergence
Fails when	Data is not linearly separable
Leads to	Foundations of SVMs and MLPs
Proven by	Novikoff (1962)

Bayees classifier

- The **Bayes classifier** is the optimal decision rule that minimizes the probability of misclassification.
- For two-class problems, it assigns a sample x to the class with the highest posterior probability:
- Class(x)=argmax P(Ci|x)
- Using Bayes' theorem:
- $P(Ci|x) \propto P(x|Ci) \cdot P(Ci)$

Bayees classifier

Gaussian Environment

- Assume each class distribution is **Gaussian** with mean μi and covariance Σ.
- If both classes share the **same covariance matrix**, the Bayes decision boundary becomes **linear**.
- Decision rule reduces to:
- wTx+b=0
- where w and b depend on class means and priors.

Bayees Classifier

- **Perceptron Connection**
- The **perceptron** also produces a **linear decision boundary**:
- y=sign(wTx+b)
- When data is linearly separable (as in Gaussian distributions with equal covariance), the perceptron's learned boundary **approximates the Bayes optimal boundary**.
- Thus:
  - **Perceptron ≈ Bayes classifier** in Gaussian environments with equal covariance.
  - Both yield linear hyperplanes separating the classes.

Relation Between Perceptron and Bayes Classifier

Key Insights

- **Perceptron is a discriminative model**: learns directly from class labels.
- **Bayes classifier is a generative model**: uses probability distributions of data.
- In Gaussian environments with equal covariance:
  - Their decision boundaries coincide.
- If covariances differ:
  - Bayes classifier produces a **quadratic boundary** (Quadratic Discriminant Analysis).
  - Perceptron cannot capture this — it remains linear.

Summary Table

Aspect	Perceptron	Bayes Classifier
Approach	Discriminative	Generative
Decision Boundary	Linear	Linear (equal covariance), Quadratic (unequal covariance)
Optimality	Converges if separable	Minimizes misclassification probability
Gaussian Environment	Approximates Bayes boundary	Exact optimal boundary

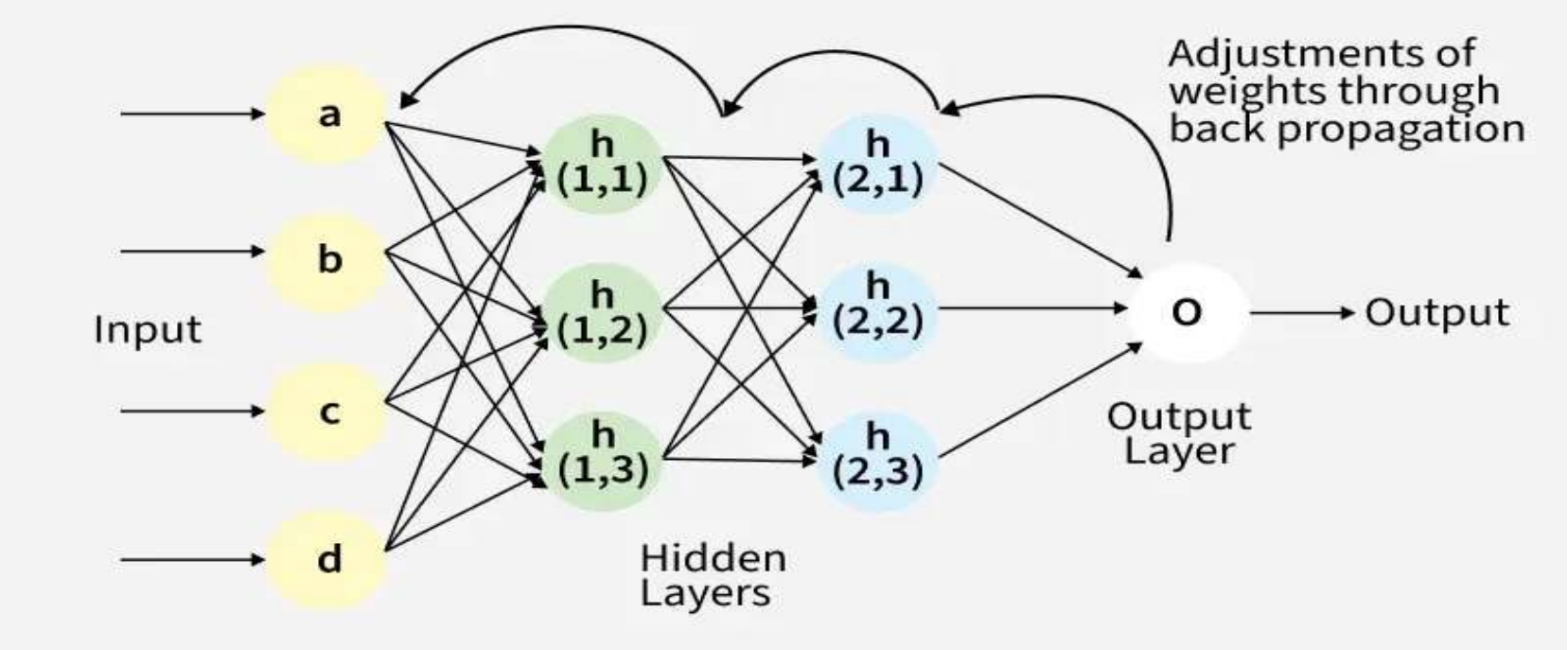
Multilayer Perceptron (MLP)

- **Definition**
- A **Multilayer Perceptron (MLP)** is a type of **feedforward artificial neural network** consisting of:
  - **Input layer** → receives raw data.
  - **Hidden layers** → perform nonlinear transformations.
  - **Output layer** → produces final predictions.
- Unlike the single-layer perceptron, MLPs can solve **nonlinear problems** (e.g., XOR problem).

Multilayer Perceptron (MLP)

- **Structure**
- **Nodes (neurons):** Each performs a weighted sum of inputs followed by an activation function.
- **Connections (weights):** Adjustable parameters learned during training.
- **Activation functions:** Introduce nonlinearity (e.g., sigmoid, tanh, ReLU).

Multilayer Perceptron (MLP)- Backpropagation Algorithm



Multilayer Perceptron (MLP)

- **Training: Backpropagation Algorithm**
- **Forward pass:** Input flows through layers → output is computed.
- **Error calculation:** Compare output with target (loss function).
- **Backward pass:** Use chain rule to compute gradients layer by layer.
- **Weight update:** Adjust weights using gradient descent (or variants).

Multilayer Perceptron (MLP)

Key Features

- **Universal Approximation Theorem:** MLPs can approximate any continuous function with enough hidden units.
- **Nonlinear decision boundaries:** Unlike single-layer perceptrons, MLPs can classify complex datasets.
- **Flexibility:** Can be used for regression, classification, and feature extraction.
- **Applications**
- **Image recognition**
- **Speech recognition**
- **Natural language processing**
- **Financial forecasting**
- **Medical diagnosis**

Multilayer Perceptron (MLP)

Example: XOR Problem

- Single-layer perceptron fails because XOR is **not linearly separable**.
- MLP with one hidden layer can solve XOR by combining nonlinear activations.
- Summary Table

Feature	Single-Layer Perceptron	Multilayer Perceptron
Layers	Input + Output	Input + Hidden + Output
Decision Boundary	Linear	Nonlinear
Solves XOR?	❌ No	✅ Yes
Training	Perceptron rule	Backpropagation
Applications	Simple classification	Complex tasks (vision, NLP, etc.)

XOR Problem

- **Definition**
- XOR is a logical operation with two binary inputs and one binary output.
- Rule: Output is **1 if inputs are different**, and **0 if inputs are the same**.

Input A	Input B	XOR Output
0	0	0
0	1	1
1	0	1
1	1	0

XOR Problem

Why XOR Matters

- A **single-layer perceptron** can only solve **linearly separable problems**.
- XOR is **not linearly separable** — you cannot draw a single straight line to separate the 1s and 0s in the input-output space.
- **Solution: Multilayer Perceptron (MLP)**
- By adding **hidden layers** with nonlinear activation functions, an MLP can solve XOR.
- Example architecture:
  - **Input layer:** 2 neurons (A, B).
  - **Hidden layer:** 2 neurons (with sigmoid/tanh/ReLU).
  - **Output layer:** 1 neuron (final XOR result).

XOR Problem

How MLP Solves XOR

- Hidden neurons learn intermediate features:
  - One neuron detects **A OR B**.
  - Another detects **A AND B**.
- Output neuron combines these features to produce XOR.
- So the nonlinear hidden layer transforms the input space into one where XOR becomes linearly separable.

XOR Problem

- XOR problem proved that **single-layer perceptrons are insufficient** for complex tasks.
- Motivated the development of **backpropagation** and **deep learning**.
- Demonstrates the power of **nonlinear transformations** in neural networks.
- Summary

Aspect	Single-Layer Perceptron	Multilayer Perceptron
Can solve XOR?	❌ No	✅ Yes
Decision boundary	Linear	Nonlinear
Training	Perceptron rule	Backpropagation
Importance	Shows limitation	Demonstrates power

Heuristics in Neural Network Training

- **Weight Initialization**
- **Problem:** Poor initialization can cause slow convergence or vanishing/exploding gradients.
- **Heuristic:**
  - Start with small random values (not all zeros).
  - Use specialized schemes:
    - **Xavier Initialization:** Keeps variance of activations stable across layers.
    - **He Initialization:** Designed for ReLU activations, prevents dying neurons.
- **Benefit:** Speeds up training and improves stability.

Heuristics in Neural Network Training

Momentum

- **Problem:** Gradient descent can be slow and oscillatory.
- **Heuristic:** Add a “momentum” term to smooth updates.
- **Update rule:**
- $v_{t+1}=\beta v_t+\eta \nabla J(w_t)$
- $w_{t+1}=w_t-v_{t+1}$
- where:
- β = momentum coefficient (0.9 typical)
- η = learning rate
- **Benefit:** Accelerates convergence, reduces oscillations, helps escape shallow local minima.

Heuristics in Neural Network Training

- **Early Stopping**
- **Problem:** Overfitting occurs when the model memorizes training data but fails on unseen data.
- **Heuristic:** Stop training when validation error stops improving.
- **Implementation:**
  - Monitor validation loss during training.
  - If no improvement for *N* epochs, halt training.
- **Benefit:** Prevents overfitting, saves computation time, improves generalization.

Summary Table

Heuristic	Purpose	Benefit
Weight Initialization	Start with good weights	Faster, stable convergence
Momentum	Smooth gradient updates	Faster training, less oscillation
Early Stopping	Prevent overfitting	Better generalization, saves time

Output Representation and Decision Rule in ANN

Output Representation

- **Binary Output (Classification):**
  - Used in simple perceptrons or binary classifiers.
  - Output is either **0 or 1** (or −1 and +1).
  - Example: Spam vs. Not Spam.
- **Probability Output (Softmax):**
  - Used in **multiclass classification**.
  - Softmax function converts raw scores (logits) into probabilities:

$$P(y = i|x) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Output Representation

- Each class gets a probability between 0 and 1.
- Example: Digit recognition (0–9).

Decision Rule

- **Binary case:**
  - If output ≥ threshold (e.g., 0.5), classify as **Class 1**.
  - Otherwise, classify as **Class 0**.
- **Multiclass case (Softmax):**
  - Choose the class with the **highest probability**.
  - Decision rule:

$$\hat{y} = \underset{i}{\operatorname{argmax}} P(y = i|x)$$

Summary Table

Output Type	Representation	Decision Rule
Binary	0/1 or -1/+1	Threshold-based (≥0.5 → Class 1)
Probability (Softmax)	Probabilities across classes	Pick class with highest probability

Computer experiment: Simulating learning with an MLP on XOR

- **Objective**
- **Goal:** Test whether an MLP can learn the XOR function and observe convergence behavior.
- **Focus:** Training dynamics, convergence speed, and generalization. **Experimental setup**
- **Dataset:** XOR truth table with inputs (0,0),(0,1),(1,0),(1,1) and labels (0,1,1,0).
- **Model:** MLP with:
  - **Input layer:** 2 neurons
  - **Hidden layer:** 2 neurons, nonlinear activation (ReLU or tanh)
  - **Output layer:** 1 neuron, sigmoid activation

Computer experiment: Simulating learning with an MLP on XOR

- **Loss: Binary cross-entropy**
- **Optimizer: SGD** with **momentum** or **Adam**
- **Initialization: He** (for ReLU) or **Xavier** (for tanh)
- **Training:** Batch size 4, epochs 500–2000

Computer experiment: Simulating learning with an MLP on XOR

Metrics to observe

- **Training loss:** Should decrease smoothly; plateaus indicate learning saturation.
- **Accuracy:** Should reach **100%** on the 4-point XOR dataset.
- **Learning curve shape:** Watch for oscillations (too high learning rate) or slow descent (too low).
- **Decision boundary:** Visualize how the hidden layer enables nonlinear separation.

Computer experiment: Simulating learning with an MLP on XOR

- **Expected results**
- **Single-layer perceptron:** Fails—cannot separate XOR.
- **MLP with hidden layer:** Converges to near-zero loss and perfect accuracy.
- **Effect of heuristics:**
  - **Momentum:** Faster, smoother convergence.
  - **Early stopping:** Prevents overfitting when using noisy augmentations.
  - **Annealing:** Stabilizes late-stage training.

Computer experiment: Simulating learning with an MLP on XOR

PSEUDOCODE

```
# Define XOR data
X = [[0,0],[0,1],[1,0],[1,1]]
y = [0,1,1,0]

# Build MLP
model = MLP(
    layers=[
        Dense(2, activation='tanh', init='xavier'),
        Dense(1, activation='sigmoid')
    ]
)

# Train
optimizer = Adam(lr=0.01)
for epoch in range(1000):
    y_pred = model.forward(X)
    loss = binary_cross_entropy(y, y_pred)
    grads = model.backward(y, y_pred)
    optimizer.step(model.parameters, grads)
    log(loss)
```

Feature Detection in ANN

Definition

- **Feature detection** refers to the process by which a neural network automatically identifies and extracts **important patterns or characteristics** from raw input data.
- These features are then used for tasks like classification, recognition, or prediction.

Feature Detection in ANN

How It Works

- **Input layer:** Receives raw data (pixels, audio signals, text tokens).
- **Hidden layers:** Learn transformations that highlight useful features.
- **Output layer:** Uses these features to make decisions (classification, regression).
- Example:
- In image recognition, early layers detect **edges and corners**.
- Deeper layers detect **shapes, textures, and objects**.

Feature Detection in ANN

- **Techniques**
- **Convolutional Neural Networks (CNNs):**
  - Specialized for images.
  - Convolution filters detect edges, textures, and higher-level features.
- **Autoencoders:**
  - Learn compressed representations of data.
  - Useful for unsupervised feature learning.
- **Principal Component Analysis (PCA):**
  - Reduces dimensionality by extracting dominant features.
- **Adaptive Filters (LMS, RLS):**
  - Detect signal features in real-time (noise patterns, frequency components).

Feature Detection in ANN

Applications

- **Computer Vision:** Object detection, face recognition, medical imaging.
- **Speech Processing:** Detecting phonemes, speaker identity.
- **Natural Language Processing (NLP):** Extracting word embeddings, semantic features.
- **Signal Processing:** Identifying frequency components, noise patterns.

Example: Image Feature Detection

- **Low-level features:** Edges, corners, gradients.
- **Mid-level features:** Shapes, textures.
- **High-level features:** Objects (faces, cars, animals).

Feature Detection in ANN

Level	Features Detected	Example
Low-level	Edges, corners	Detecting outlines in an image
Mid-level	Shapes, textures	Identifying patterns like stripes
High-level	Objects	Recognizing a cat vs dog

Thank You..