### 🎯 Objective:

Develop a deep understanding of neural networks and how they power deep learning systems. Learn to build, train, and optimize models for computer vision, sequence processing, and foundational NLP.

## 🔍 Curriculum Breakdown

### 1. Neural Network Basics

#### 📌 1.1 Core Concepts

* What are artificial neurons and how they mimic biological ones
* Perceptrons and the building blocks of neural networks
* Activation functions:
  + ReLU (default in hidden layers)
  + Sigmoid (binary classification)
  + Softmax (multi-class output layers)

#### 📌 1.2 Forward & Backward Propagation

* Forward pass: weighted sum and activation
* Backpropagation and the chain rule
* Gradient descent and parameter updates

#### 📌 1.3 Loss Functions

* MSE (Mean Squared Error): for regression
* Cross-Entropy: for classification tasks
* Log loss interpretation and probabilities

📌 Hands-On Lab: Build a neural net from scratch using NumPy for a binary classification task

### 2. Deep Learning Architectures

#### 📌 2.1 Convolutional Neural Networks (CNNs)

* Convolutional layers, filters/kernels
* Feature maps and receptive fields
* Pooling (MaxPooling, AvgPooling)
* Flattening and fully connected layers
* Use-cases: image classification, object detection

#### 📌 2.2 Recurrent Neural Networks (RNNs) and LSTMs

* Time series and sequence modeling
* RNNs: vanishing gradients, sequential state
* LSTMs: forget/update gates, long-term memory
* Use-cases: stock prediction, text generation

#### 📌 2.3 Transformers (Intro Level)

* Limitations of RNNs → Need for attention
* Self-attention and positional encoding
* Encoder-decoder overview
* Used in NLP, vision, audio — foundation for GenAI

📌 Mini Project: Build a CNN for classifying CIFAR-10 or MNIST  
 📌 Optional: Sequence-to-sequence model for text input

### 3. Optimization & Regularization

#### 📌 3.1 Optimization Algorithms

* SGD: classic baseline
* Adam: adaptive learning rates
* RMSProp: smoothing with momentum

#### 📌 3.2 Regularization Techniques

* Dropout: randomly turn off neurons during training
* Batch Normalization: normalize activations within layers
* Weight Decay (L2 regularization): penalize large weights

#### 📌 3.3 Learning Rate Control

* Static vs. dynamic learning rates
* Learning rate schedulers
* Gradient clipping to avoid exploding gradients

📌 Lab: Train the same network with and without dropout, visualize differences in loss curves

### 4. Frameworks & Tools

#### ✅ Core Libraries

* TensorFlow + Keras:
  + High-level APIs, great for beginners
  + Deployment-ready
* PyTorch:
  + Dynamic graphs for flexible training loops
  + Preferred for research

#### ✅ Additional Ecosystem Tools

* PyTorch Lightning: modular code for cleaner experiments
* Hugging Face Transformers: use pre-trained models with ease
* Weights & Biases / TensorBoard: monitor experiments visually

📌 Activity: Build the same CNN in both TensorFlow and PyTorch

### 5. Training & Evaluation

#### 📌 5.1 Model Training Workflow

* Batches, epochs, and early stopping
* Visualizing training vs. validation accuracy/loss
* Managing overfitting with validation sets and dropout

#### 📌 5.2 Preprocessing for Deep Learning

* Image:
  + Resizing, normalization, channel adjustments
* Text:
  + Tokenization, padding, vocab building, embeddings
* Data generators and input pipelines (tf.data, PyTorch Datasets)

📌 Lab: Build a training loop with live loss/accuracy chart

### 6. Best Practices for Robust Training

#### 📌 6.1 Data Augmentation

* For images: flipping, rotation, cropping, brightness
* For text: synonym replacement, random deletion, back-translation

#### 📌 6.2 Transfer Learning

* Load pre-trained models: VGG, ResNet, BERT
* Fine-tune vs. freeze layers
* Apply to small datasets with fewer resources

#### 📌 6.3 Checkpointing & Callbacks

* Save best model weights
* Auto-adjust learning rate on plateaus
* Stop early to prevent overfitting

📌 Mini Project: Train a CNN with augmentations and evaluate improvements  
 📌 Bonus: Load a pre-trained model, fine-tune it on a custom dataset