The Human Brain

Artificial Neural Networks (ANN) are inspired by the biological human brain.

🔷 Structure of the Human Brain

- The brain contains **~86 billion neurons**.
- Neurons are connected through **synapses**.
- Signals are transmitted using **electrochemical impulses**.

🔷 Structure of a Biological Neuron

A biological neuron has:

1. **Dendrites** – Receive signals
2. **Cell Body (Soma)** – Processes signals
3. **Axon** – Sends signals
4. **Synapse** – Connection between neurons

🔷 Working

- Input signals come through dendrites.
- If total signal exceeds a threshold → neuron fires.
- Signal passes to next neuron.

👉 ANN mimics this structure mathematically.

---

2 Models of a Neuron (Artificial Neuron)

An artificial neuron is a mathematical model of a biological neuron.

🔷 Components of Artificial Neuron

1. **Inputs (x_1, x_2, x_3, ...)**
2. **Weights (w_1, w_2, w_3, ...)**
3. **Summation Function**
4. **Bias (b)**
5. **Activation Function**

🔷 Mathematical Model

$$y = f(\sum w_i x_i + b)$$

Where:

- w_i= weight
- x_i= input
- b= bias
- f= activation function
- y= output

🔷 Activation Functions

1. **Step Function**
2. **Sigmoid Function**

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

3. **Tanh Function**
4. **ReLU Function**

$$f(x) = max(0, x)$$

---

3 Perceptron

The **Perceptron** was introduced by **Frank Rosenblatt** in 1958.

🔷 Definition

A perceptron is a **single-layer neural network** used for binary classification.

🔷 Structure

- Input Layer
- Output Layer
- No hidden layer

🔷 Algorithm Steps

1. Initialize weights randomly.

2. For each training example:
    - Compute output
    - Compare with target
    - Update weights

🔷 **Weight Update Rule**

$$w_i = w_i + \eta(t - y)x_i$$

Where:

- η= learning rate
- t= target
- y= output

🔷 **Limitation of Perceptron**

It can only solve **linearly separable problems**.

Example:
✓ AND
✓ OR
❌ XOR

4️⃣ **Multilayer Perceptrons (MLP)**

To overcome perceptron limitations, **Multilayer Perceptron (MLP)** was introduced.

🔷 **Structure**

- Input Layer
- One or more Hidden Layers
- Output Layer

🔷 **Features**

- Uses nonlinear activation functions
- Can solve complex problems
- Can solve XOR problem

🔷 Why Hidden Layer?

Hidden layer allows:

- Nonlinear mapping
- Feature extraction
- Complex decision boundaries

---

5 The Back-Propagation Algorithm

Backpropagation is a supervised learning algorithm used to train MLP.

It was popularized by:

- **Geoffrey Hinton**
- **David Rumelhart**
- **Ronald Williams**

🔷 Purpose

To minimize error by updating weights using **gradient descent**.

🔷 Steps of Backpropagation

Step 1: Forward Pass

- Input → Hidden → Output
- Compute output

Step 2: Compute Error

$$Error = Target - Output$$

Step 3: Backward Pass

- Calculate gradients
- Propagate error backward

Step 4: Update Weights

$$w = w - \eta \frac{\partial E}{\partial w}$$

6 XOR Problem

◆ XOR Truth Table

x1	x2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

◆ Why Perceptron Fails?

Because XOR is **not linearly separable**.

No single straight line can separate output classes.

This limitation was highlighted by:

- **Marvin Minsky**
- **Seymour Papert**
  in the book **Perceptrons**

◆ Solution

Using:
✓ Hidden Layer
✓ Nonlinear Activation

MLP can solve XOR easily.

---

7 Heuristics for Making Backpropagation Perform Better

Backpropagation sometimes suffers from:

- Slow convergence
- Local minima

- Vanishing gradients

🔷 Important Heuristics

1 Proper Weight Initialization

- Small random values
- Xavier or He initialization

2 Learning Rate Tuning

- Too large → divergence
- Too small → slow learning

3 Momentum

$$v = \beta v - \eta \nabla E$$
$$w = w + v$$

Helps:

- Avoid local minima
- Faster convergence

4 Adaptive Learning Rate

- Adam
- RMSProp

5 Normalize Inputs

- Improves convergence speed

6 Use Better Activation Functions

- ReLU instead of Sigmoid

---

8 Backpropagation and Differentiation

Backpropagation is based on:

🔷 Chain Rule of Calculus

If:

$$E = f(g(h(x)))$$

Then:

$$\frac{dE}{dx} = \frac{dE}{df} \cdot \frac{df}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{dx}$$

This allows:

- Error to propagate backward layer by layer
- Efficient computation of gradients

🔷 Why Differentiation is Important?

Because:

- We need gradient to update weights.
- Activation function must be differentiable.
- Enables gradient descent optimization.

Topic	Key Idea
Human Brain	Inspiration for ANN
Neuron Model	Weighted sum + activation
Perceptron	Single layer classifier
MLP	Multi-layer nonlinear network
Backpropagation	Gradient-based learning
XOR Problem	Shows limitation of perceptron
Heuristics	Improve convergence
Differentiation	Chain rule for gradient computation

UNIT–2 Optimization & Hyperparameter Tuning

1 Optimization in Deep Learning

What is Optimization?

Optimization means:

Find parameters (weights) W that minimize loss $J(W)$

Loss function examples:

- Mean Squared Error (MSE)
- Cross Entropy Loss

Goal:

$$W^* = \arg\min J(W)$$

---

2 Gradient Descent (GD)

Gradient Descent is the basic optimization algorithm.

Idea

Move weights in the **opposite direction of gradient**.

$$W = W - \eta \nabla J(W)$$

Where:

- η= learning rate
- $\nabla J(W)$= gradient of loss

Working Steps

1. Initialize weights randomly.
2. Compute gradient of loss.
3. Update weights.
4. Repeat until convergence.

🔷 Problem

- Slow for large datasets.
- Can get stuck in local minima.
- Sensitive to learning rate.

---

3 Stochastic Gradient Descent (SGD)

Instead of using full dataset, update weights using **one training example**.

$$W = W - \eta \nabla J(W_i)$$

🔷 Advantages

✓ Faster
✓ Escapes local minima

🔷 Disadvantages

❌ Noisy updates
❌ High variance

---

4 Momentum

SGD oscillates in narrow valleys. Momentum helps smooth updates.

Inspired by physics.

$$v = \beta v - \eta \nabla J(W)$$
$$W = W + v$$

Where:

- β = momentum term (0.9 common)

🔷 Advantages

✓ Faster convergence
✓ Reduces oscillation

---

5 RMSProp (Root Mean Square Propagation)

Problem:

- Learning rate same for all parameters.

RMSProp adapts learning rate individually.

$$s = \beta s + (1-\beta)(\nabla J(W))^2$$
$$W = W - \frac{\eta}{\sqrt{s+\epsilon}}\nabla J(W)$$

🔷 Advantages

✓ Handles vanishing gradients
✓ Good for non-stationary problems

---

6 ADAM (Adaptive Moment Estimation)

ADAM combines:

- Momentum
- RMSProp

Developed by:
Diederik P. Kingma
and
Jimmy Ba

🔷 Equations

Momentum estimate:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

RMS estimate:

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Update rule:

$$W = W - \frac{\eta m_t}{\sqrt{v_t}+\epsilon}$$

🔷 Why ADAM is Popular?

✓ Fast convergence
✓ Less tuning required
✓ Works well in deep networks

---

7 Hyperparameter Tuning

◆ What are Hyperparameters?

Parameters NOT learned automatically:

- Learning rate
- Batch size
- Number of hidden layers
- Number of neurons
- Activation function
- Regularization parameter

◆ Methods of Tuning

1 Grid Search

Try all combinations.

2 Random Search

Randomly select values.

3 Bayesian Optimization

Smart search using probability models.

---

8 Regularization for Deep Learning

Deep networks easily overfit.

Regularization reduces overfitting.

◆ 8.1 Parameter Norm Penalties

Add penalty term to loss:

$$J(W) = J_{original}(W) + \lambda\Omega(W)$$

Where:

- λ= regularization parameter

🔷 8.2 L2 Parameter Regularization (Weight Decay)

$$\Omega(W) = || W ||^2$$

New loss:

$$J(W) = J(W) + \lambda\sum W^2$$

Update rule becomes:

$$W = (1 - \eta\lambda)W - \eta\nabla J(W)$$

🔷 Effects

✓ Prevents large weights
✓ Smooth model
✓ Reduces overfitting

🔷 L1 vs L2

L1	L2
Produces sparse weights	Produces small weights
Feature selection	Weight shrinking

---

9 Dataset Augmentation

Artificially increase dataset size.

Example in image processing:

- Rotation
- Flipping
- Cropping

- Noise addition

Benefits:
✓ Reduces overfitting
✓ Improves generalization

Used in CNN models like:

- **ResNet**
- **VGGNet**

🔟 Semi-Supervised Learning

Used when:

- Few labeled data
- Large unlabeled data

🔷 Methods

1. Self-training
2. Pseudo labeling
3. Consistency regularization

Used in modern systems like:

- **OpenAI** models
- Large language models

1️⃣1️⃣ Optimization for Training Deep Models

Deep models suffer from:

🔷 1. Vanishing Gradient Problem

Gradient becomes very small (sigmoid/tanh).

🔷 2. Exploding Gradient Problem

Gradient becomes very large.

🔷 Solutions

✓ ReLU activation
✓ Proper initialization (He/Xavier)
✓ Batch Normalization
✓ Gradient Clipping
✓ Use ADAM optimizer

🔥 Comparison of Optimizers

Optimizer	Speed	Stability	Adaptive
GD	Slow	Stable	No
SGD	Fast	Noisy	No
Momentum	Faster	Smooth	No
RMSProp	Fast	Stable	Yes
ADAM	Very Fast	Very Stable	Yes

📌 Final Summary

Unit–2 focuses on:

- Gradient-based optimization
- Advanced optimizers (Momentum, RMSProp, ADAM)
- Hyperparameter tuning
- Regularization techniques
- Handling deep network training challenges

UNIT–3: Convolutional Neural Networks (CNN)

CNNs are specialized neural networks used mainly for **image processing, computer vision, and pattern recognition**.

They automatically learn:

- Edges
- Shapes
- Textures
- Objects

---

1 Convolution Operation

🔷 What is Convolution?

Convolution is a mathematical operation between:

- Input image (matrix)
- Filter (kernel)

It produces a **feature map**.

🔷 Mathematical Representation

If:

- Input size = $N \times N$
- Filter size = $F \times F$
- Stride = S
- Padding = P

Output size:

$$\frac{N - F + 2P}{S} + 1$$

🔷 How Convolution Works

1. Place filter on image.

2. Multiply element-wise.
3. Sum all values.
4. Move filter (stride).
5. Repeat.

🔷 Why Convolution?

✓ Captures spatial features
✓ Reduces parameters
✓ Local connectivity

---

2 Filters (Kernels)

A filter is a small matrix (e.g., 3×3, 5×5).

Different filters detect:

- Vertical edges
- Horizontal edges
- Corners
- Textures

Example:
Edge detection filter:

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

CNN learns these filters automatically.

---

3 Padding

Padding means adding zeros around the input.

🔷 Types

1. Valid Padding (No padding)
2. Same Padding (Output size = Input size)

🔷 Why Padding?

✓ Prevents shrinking
✓ Preserves border information
✓ Controls output size

4️⃣ Stride

Stride = number of pixels filter moves.

- Stride = 1 → detailed output
- Stride = 2 → reduced output

Larger stride → smaller output feature map.

5️⃣ Pooling

Pooling reduces spatial dimensions.

🔷 Types

1️⃣ Max Pooling

Takes maximum value.

2️⃣ Average Pooling

Takes average value.

🔷 Advantages

✓ Reduces computation
✓ Controls overfitting
✓ Makes model translation invariant

6️⃣ Normalization

Normalization stabilizes training.

🔷 Batch Normalization

Introduced by:
Sergey Ioffe
and
Christian Szegedy

It:

- Normalizes activations
- Reduces internal covariate shift
- Speeds up training

---

7 Modern CNN Architectures

🔷 7.1 LeNet

LeNet

Developed by:
Yann LeCun

Features:

- 2 Convolution layers
- 2 Pooling layers
- Fully connected layers
- Used for digit recognition (MNIST)

🔷 7.2 AlexNet

AlexNet

Developed by:
Alex Krizhevsky

Innovations:

✓ ReLU activation
✓ Dropout
✓ GPU training

Won ImageNet 2012 competition.

🔷 7.3 VGG

VGGNet

Features:

✓ 3×3 filters only
✓ Very deep (16 or 19 layers)
✓ Simple architecture

Problem:
❌ Large parameters

🔷 7.4 GoogLeNet (Inception v1)

GoogLeNet

Introduced Inception module.

✓ Multiple filter sizes (1×1, 3×3, 5×5)
✓ Parallel convolution
✓ Fewer parameters

🔷 7.5 ResNet

ResNet

Developed by:
Microsoft Research

Key Innovation:

✓ Residual connections (Skip connections)

Solves:

- Vanishing gradient problem

Formula:

$$H(x) = F(x) + x$$

🔷 7.6 Inception (Later Versions)

Inception

Improvements:
✓ Factorized convolutions
✓ Efficient computation
✓ Auxiliary classifiers

🔷 7.7 EfficientNet

EfficientNet

Introduced compound scaling:

Simultaneously scales:

- Depth
- Width
- Resolution

More accurate with fewer parameters.

---

8 Transfer Learning

Transfer learning means:

Using a **pretrained model** and adapting it to new task.

Example:

- Use pretrained ResNet on ImageNet
- Replace last layer
- Train on small dataset

🔷 Advantages

✓ Saves time
✓ Works with small dataset
✓ High accuracy

---

9 Fine-Tuning

Two approaches:

1️⃣ Freeze base layers
2️⃣ Train entire network with small learning rate

Fine-tuning improves performance.

🔟 Image Augmentation

Artificially increase dataset size.

🔷 Techniques

- Rotation
- Flipping
- Zooming
- Cropping
- Color jittering
- Noise addition

Benefits:
✓ Reduces overfitting
✓ Improves generalization

🔥 CNN Layer Flow Summary

Input Image
↓
Convolution
↓
ReLU
↓
Pooling
↓
Repeat (Conv + Pool)
↓
Flatten
↓
Fully Connected Layer
↓
Softmax Output

📌 Comparison of Architectures

Model	Depth	Innovation
LeNet	Shallow	First CNN
AlexNet	Medium	ReLU + GPU
VGG	Deep	3×3 filters
GoogLeNet	Deep	Inception module
ResNet	Very Deep	Skip connections
EfficientNet	Efficient	Compound scaling

🎯 Important Exam Points

✓ Formula of convolution output size
✓ Difference between padding and stride
✓ Max vs Average pooling
✓ ResNet residual connection formula
✓ Transfer learning steps

🔷 UNIT–4: Sequence Modeling

Sequence modeling deals with **ordered data**, where previous inputs influence future outputs.

Examples:

- Text
- Speech
- Time-series data
- DNA sequences

Traditional neural networks cannot handle sequence dependencies effectively.
This led to the development of **Recurrent Neural Networks (RNNs).**

---

1 Recurrent Neural Networks (RNN)

🔷 What is RNN?

RNN is a neural network designed to handle **sequential data**.

Unlike feedforward networks:

- RNN has **memory**
- Output depends on previous inputs

🔷 Basic RNN Structure

At time step t:

$$h_t = f(W_h h_{t-1} + W_x x_t + b)$$
$$y_t = g(W_y h_t)$$

Where:

- x_t= input at time t
- h_t= hidden state
- y_t= output
- h_{t-1}= previous memory

🔷 Key Idea

The hidden state acts as memory:

$$\text{Current Output} = f(\text{Current Input + Previous Memory})$$

🔷 Applications

✓ Language modeling
✓ Speech recognition
✓ Time-series forecasting

---

2 Vanishing Gradient Problem in RNN

🔷 Why It Happens?

During Backpropagation Through Time (BPTT):

Gradients are multiplied repeatedly.

If:

- Weights < 1 → gradients shrink → **vanish**
- Weights > 1 → gradients grow → **explode**

🔷 Problem

Vanishing gradient prevents learning long-term dependencies.

Example:
In sentence:

"I grew up in France... I speak fluent ___"

To predict "French", model must remember "France".

Basic RNN fails for long sequences.

---

3 LSTM (Long Short-Term Memory)

To solve vanishing gradient problem, **Sepp Hochreiter** and **Jürgen Schmidhuber** introduced LSTM (1997).

🔷 Structure of LSTM

LSTM has:

1. Forget Gate
2. Input Gate
3. Output Gate
4. Cell State (Long-term memory)

🔷 LSTM Equations

Forget Gate:

$$f_t = \sigma(W_f[h_{t-1}, x_t])$$

Input Gate:

$$i_t = \sigma(W_i[h_{t-1}, x_t])$$

Candidate Value:

$$\tilde{C}_t = tanh(W_c[h_{t-1}, x_t])$$

Cell State Update:

$$C_t = f_t C_{t-1} + i_t \tilde{C}_t$$

Output:

$$h_t = o_t \tanh(C_t)$$

🔷 Why LSTM Works?

✓ Maintains long-term memory
✓ Prevents vanishing gradients
✓ Selectively forgets irrelevant information

---

4 GRU (Gated Recurrent Unit)

Introduced by:
Kyunghyun Cho

GRU is a simplified version of LSTM.

🔷 GRU Structure

Only two gates:

1 Update Gate
2 Reset Gate

🔷 GRU Equations

Update gate:

$$z_t = \sigma(W_z[h_{t-1}, x_t])$$

Reset gate:

$$r_t = \sigma(W_r[h_{t-1}, x_t])$$

Hidden state:

$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t$$

🔷 LSTM vs GRU

Feature	LSTM	GRU
Gates	3	2
Parameters	More	Fewer
Speed	Slower	Faster
Performance	Slightly better	Comparable

---

5 Bidirectional RNN (Bi-RNN)

Standard RNN reads sequence only forward.

Bidirectional RNN reads:

- Forward
- Backward

Introduced by:
Mike Schuster

🔷 Structure

$$h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}]$$

Final output combines both directions.

🔷 Advantage

Uses both:

- Past context
- Future context

Example:
Sentence:

"The movie was not ___"

Future word helps determine sentiment.

---

6 Applications of Sequence Modeling

🔷 6.1 Text Generation

Model predicts next word in sequence.

Example:

- Chatbots
- Story generation
- Code completion

Used in large models developed by:
OpenAI

🔷 6.2 Sentiment Analysis

Classify text as:

- Positive
- Negative
- Neutral

Steps:

1. Convert text to embeddings.

2. Pass through RNN/LSTM.
3. Output classification.

Example:
"I love this product" → Positive

🔷 6.3 Time-Series Forecasting

Predict future values using past values.

Examples:

- Stock price prediction
- Weather forecasting
- Sales prediction

Input:

$$[x_1, x_2, x_3, \ldots]$$

Output:

$$x_{future}$$

LSTM works well because it remembers long-term trends.

🔥 Comparison Summary

Model	Handles Long-Term Memory?	Complexity
RNN	No	Simple
LSTM	Yes	Complex
GRU	Yes	Moderate
Bi-RNN	Yes (both directions)	Moderate

🎯 Important Exam Points

✓ Difference between RNN and LSTM
✓ Why vanishing gradient occurs

✓ LSTM gate equations
✓ GRU structure
✓ Bidirectional RNN concept
✓ Applications in NLP and time-series

📌 **Complete Flow of Sequence Model**

Input Sequence
↓
Embedding Layer
↓
RNN / LSTM / GRU
↓
Fully Connected Layer
↓
Softmax Output

UNIT–5: Attention, Transformers & Autoencoders

1 Attention Mechanism

🔷 Why Attention?

In RNN/LSTM:

- Long sequences lose information.
- All hidden states treated equally.

Attention allows model to:
✓ Focus on important words
✓ Assign different importance (weights)

🔷 Basic Idea

For each output word:

- Compute importance scores for all input words.
- Take weighted sum.

🔷 Mathematical Formula

Attention score:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where:

- Q = Query
- K = Key
- V = Value
- d_k= dimension scaling factor

---

2 Self-Attention

Self-attention means:
Each word attends to all other words in the same sentence.

Example:
Sentence:

"The animal didn't cross the street because it was too tired."

"It" refers to "animal" → self-attention captures this.

🔷 Steps in Self-Attention

1. Convert input to embeddings.
2. Create Q, K, V matrices.
3. Compute attention scores.
4. Multiply with values.
5. Get new representation.

---

3 Multi-Head Attention

Instead of one attention, use multiple attention heads.

Each head:

- Learns different relationships.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, \ldots, head_h)W^O$$

Advantages:
✓ Capture different semantic relations
✓ More powerful representation

---

4 Positional Encoding

Transformers have no recurrence or convolution.

So they don't know order.

Solution:
Add positional encoding.

$$PE(pos, 2i) = \sin(pos/10000^{2i/d})$$
$$PE(pos, 2i+1) = \cos(pos/10000^{2i/d})$$

This gives sequence information.

5️⃣ Transformer Architecture

Proposed in 2017 paper:
Attention Is All You Need

Developed by:
Google Brain

🔷 Transformer Components

Encoder Block:

- Multi-head attention
- Feedforward network
- Add & Layer Normalization

Decoder Block:

- Masked multi-head attention
- Encoder-decoder attention
- Feedforward network

🔷 Architecture Flow

Input → Embedding → Positional Encoding
↓
Multi-Head Attention
↓
Add & Norm
↓
Feed Forward
↓
Add & Norm
↓
Output

6️⃣ Encoder–Decoder Architecture

Used in:

- Machine translation
- Text summarization

Encoder:

- Converts input into context vector.

Decoder:

- Generates output sequence.

Example:
English → French translation

---

7 Autoencoders

Autoencoder is an unsupervised neural network.

Goal:
Learn compressed representation.

Structure:
Input → Encoder → Bottleneck → Decoder → Output

Loss:
Reconstruction loss.

🔷 7.1 Sparse Autoencoders

Constraint:
Most neurons inactive.

Add sparsity penalty.

Benefits:
✓ Feature learning
✓ Avoid overfitting

🔷 7.2 Denoising Autoencoders

Input:
Corrupted data

Target:
Original data

Model learns:
✓ Robust features
✓ Noise removal

🔷 7.3 Variational Autoencoders (VAE)

Proposed by:
Diederik P. Kingma

VAE is probabilistic.

Instead of encoding to fixed vector:
Encode to distribution:

$$z \sim N(\mu, \sigma)$$

Loss:

- Reconstruction loss
- KL divergence loss

Used in:
✓ Image generation
✓ Data generation

---

8 BERT

BERT

Developed by:
Google

Full form:
Bidirectional Encoder Representations from Transformers

🔷 Key Features

✓ Uses Transformer encoder only
✓ Bidirectional training
✓ Pretrained on large corpus

Training tasks:

- Masked Language Modeling (MLM)

- Next Sentence Prediction (NSP)

🔷 Applications

✓ Question answering
✓ Sentiment analysis
✓ Named entity recognition

9️⃣ GPT

GPT

Developed by:
OpenAI

🔷 Key Features

✓ Uses Transformer decoder only
✓ Autoregressive model
✓ Predicts next word

Training:
Left-to-right language modeling.

🔷 GPT vs BERT

Feature	BERT	GPT
Architecture	Encoder	Decoder
Direction	Bidirectional	Left-to-right
Task	Understanding	Generation

🔥 Complete Flow Summary

Sequence Input
↓
Embedding + Positional Encoding
↓
Multi-Head Self Attention

↓
Feed Forward Network
↓
Layer Normalization
↓
Output

📌 Important Exam Points

✓ Attention formula
✓ Difference between self-attention & multi-head
✓ Transformer block diagram
✓ Autoencoder types
✓ VAE concept
✓ BERT vs GPT comparison

🎯 Overall Unit–5 Summary

This unit connects:

RNN Limitations → Attention → Transformers →
Modern NLP models → BERT & GPT → Generative AI

🔷 UNIT–6: Generative Models & GANs

1️⃣ Basics of Generative Models

🔹 What is a Generative Model?

A generative model learns the **data distribution** $P(x)$and generates new samples similar to training data.

Example:

- Train on face images → generate new faces

- Train on handwriting → generate new digits

🔷 Types of Generative Models

1. Autoregressive models
2. Variational Autoencoders (VAE)
3. Generative Adversarial Networks (GANs)
4. Diffusion models (modern approach)

🔷 Generative vs Discriminative

Model Type	Learns	Example
Discriminative	(P(y	x))
Generative	$P(x)$or $P(x, y)$	GAN

---

2 Generative Adversarial Networks (GANs)

Proposed in 2014 by:
Ian Goodfellow

GAN consists of two neural networks:

1. Generator (G)
2. Discriminator (D)

They compete in a **minimax game**.

🔷 2.1 Generator

- Takes random noise $z \sim P(z)$
- Generates fake sample $G(z)$

Goal:
Fool the discriminator.

🔷 2.2 Discriminator

- Takes real and fake samples.
- Outputs probability: real or fake.

Goal:
Correctly classify.

🔷 GAN Objective Function

$$\min_G \max_D V(D,G) = E_{x\sim P_{data}}[\log D(x)] + E_{z\sim P_z}[\log(1 - D(G(z)))]$$

🔷 Training Process

Step 1: Train discriminator
Step 2: Train generator
Repeat until equilibrium.

At equilibrium:
Generator produces realistic samples.

---

3 DCGAN (Deep Convolutional GAN)

Proposed in 2015.

Uses:
✓ Convolutional layers
✓ No pooling layers
✓ Batch normalization
✓ ReLU activation in generator
✓ LeakyReLU in discriminator

🔷 Why DCGAN?

Traditional GAN used fully connected layers.

DCGAN:
✓ Better image quality
✓ Stable training
✓ Learns hierarchical features

Used for:

- Image generation
- Feature learning

---

4 CycleGAN

Used for **image-to-image translation without paired data.**

Example:

- Horse → Zebra
- Summer → Winter
- Sketch → Photo

🔷 Key Idea: Cycle Consistency

If:
Horse → Zebra → Horse

The final horse should match original.

Loss includes:

- Adversarial loss
- Cycle consistency loss

🔷 Advantages

✓ No paired dataset required
✓ Style transformation

Applications:

- Artistic style transfer
- Domain adaptation

---

5 StyleGAN

Developed by:
NVIDIA

Used for:
✓ High-quality face generation

🔷 Key Innovation

Instead of feeding noise directly:

Noise → Mapping network → Style vector → Adaptive Instance Normalization (AdaIN)

This allows:
✓ Control over facial features
✓ Smooth interpolation
✓ High-resolution images

🔷 StyleGAN Improvements

- StyleGAN2
- StyleGAN3 (better stability)

Used in:

- Deepfake generation
- AI avatars

---

6 Applications of GANs

🔷 1. Image Generation

- Human faces
- Artwork
- Landscapes

🔷 2. Super-Resolution

Low resolution → High resolution image

🔷 3. Image-to-Image Translation

- Day → Night
- Sketch → Real image

🔷 4. Data Augmentation

Generate synthetic medical images.

🔷 5. Video Generation

🔷 6. Text-to-Image Generation

(Though diffusion models are now dominant)

7️⃣ Advantages of GANs

✓ Produce highly realistic images
✓ No explicit probability estimation
✓ Creative generation

8️⃣ Challenges in GAN Training

❌ Mode collapse (limited variety)
❌ Training instability
❌ Vanishing gradients
❌ Sensitive to hyperparameters

9️⃣ GAN vs VAE

Feature	GAN	VAE
Output Quality	Sharp images	Slightly blurry
Stability	Hard to train	More stable
Probabilistic	No	Yes

🔟 Complete GAN Workflow

Noise z
↓
Generator
↓
Fake Image
↓
Discriminator
↓

Real/Fake Classification
↓
Backpropagation

📌 Important Exam Points

✓ GAN minimax loss
✓ Generator vs Discriminator
✓ DCGAN architecture rules
✓ CycleGAN cycle consistency
✓ StyleGAN innovation
✓ Applications

🎯 Final Summary of Unit–6

Generative models learn data distribution.
GAN introduced adversarial learning.
DCGAN improved stability.
CycleGAN enabled unpaired translation.
StyleGAN achieved high realism.

This unit connects to:

- Computer vision
- Deepfake technology
- Creative AI
- Modern generative AI systems