Date: __/__/20__ Day: _____

Deep learning Complete Notes

By

Mirza Yasir Abdullah Baig

Introduction to Neural Networks

Neural Networks

Biological Neurons vs Artifical Neurons

Single layer Perceptron

Multi-layer Perceptron

Artifical Neural Networks (ANNs)

Types of Neural Networks

Architecture and Learning in neural network

Basic Components of Neural Networks

Layers in Neural Networks

Weights and Biases

Forward Propagation

Activation Functions

Loss Functions

Backpropagation

Learning Rate

Optimization Algorithm in Deep Learning

Optimization algorithms in DL

Gradient Descent

Stochastic Gradient Descent (SGD)

Batch Normalization

Mini-Batch Gradient Descent

Adam (Adaptive Moment Estimation)

Momentum-based Gradient Optimizer

Date: __/__/20 Day: _____

Adagrad Optimizer
RMSProp Optimizer

Types of Deep learning Models
Convolutional Neural Networks (CNNs)
Deep Learning Algorithms
Convolutional Neural Network (CNNs)
Basics of Digital Image Processing
Importance for CNN
Padding
Convolutional layers
Pooling layers
Fully Connected Layers
Backpropagation in CNNs
CNN based Image Classification using PyTorch
CNN based Image Classification using TensorFlow
CNN Based Architectures
Convolutional Neural Network (CNN) Architectures
LeNet-5
AlexNet
VGGnet
VGG-16 Network
GoogleNet / Inception
ResNet (Residual Network)
MobileNet

Recurrent Neural Networks (RNNs)
Recurrent Neural Networks (RNNs)
How RNN Differs from Feedforward Neural Networks
Backpropagation Through Time (BPTT)
Vanishing Gradient & Exploding Gradient Problem

Date: __/__/20 Day: _____

Training of RNN in TensorFlow
Sentiment Analysis with RNN
Types of Recurrent Neural Networks
Types of Recurrent Neural Networks
Bidirectional RNNs
Long Short-Term Memory (LSTM)
Bidirectional Long Short-Term Memory (Bi-LSTM)
Gated Recurrent Units (GRU)

Generative Models in Deep Learning
Generative Adversarial Networks (GANs)
Autoencoders
GAN vs. Transformer Models
Types of Generative Adversarial Networks (GANs)
Deep Convolutional GAN (DCGAN)
Conditional GAN (cGAN)
Cycle-Consistent GAN (cycleGAN)
Super-Resolution GAN (SRGAN)
Style GAN
Types of Autoencoders
Types of Autoencoders
Sparse Autoencoder
Denoising Autoencoder
Convolutional Autoencoder
Variational Autoencoder

Deep learning Reinforcement (DRL)
Deep Reinforcement Learning
Reinforcement Learning
Markov Decision Processes

Date: __/__/20 Day: _____

Key Algorithms of DRL

Deep Q-Networks (DQN)

REINFORCE

Actor - Critic Methods

Proximal Policy Optimization (PPO)

Introduction to Neural Networks

Neural Networks

What is neural networks?

A neural network is a machine learning model inspired by the human brain

It has interconnected neurons (nodes) that process data, learn patterns, and make decisions - used in tasks like pattern recognition, NLP, & self driving

Key Components

- Neurons
  - Basic units that receive inputs and apply activation functions
- Connections
  - links B/w neurons that carry information
- Weights & Bias
  - Control the strength of neuron connections
- Propagation Function
  - Transfer data b/w layers
- Learning Rule
  - Update weights/biases for reduce errors

Learning Process

- Input Computation - Data enters the network
- Output Generation - Network predicts output

Date: __/__/20__ Day: _____

- Iterative Reinfinement - weights/bias adjust to improve accuracy

As the network trains, it adapts to data, improving its decisions over time

Importance of Neural Networks

- Find Complex patterns
  - Detect hidden structures in data
- Learn Automatically
  - Improves with experience
- Power AI Technologies
  - Used in NLP, self-driving cars, automation
- Increase Efficiency
  - Automates complex tasks
- Core of AI
  - Foundation of modern AI

Layers in Neural Networks

Input layer: Takes raw input data (features)
Hidden layer: Perform computation & transformation
Output layer: Produces final result (classification/Regression)

Working of Neural Networks

- Forward Propagation
  - Linear Transformation

$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$

  - Activation Function:

Adds non-linearity (e.g ReLU, Sigmoid, Tanh)

- Backpropagation
  - Loss Function: Measure prediction error (e.g MSE)

Date: __/__/20 Day: _____

- Gradient Calculation: use calculus chain Rule to find error contribution
- Weight Update: Adjusts weights using Gradient Descent with a learning rate
- Iteration

This forward → backward → update cycle repeats until errors become minimal

Learning Types

- Supervised : Learn from Labeled data
- Unsupervised: Learn from labeled data
- Reinforcement : learn by trial & error

Types of Neural Networks

- FeedForward NN
- Single-layer Perceptron
- MultiLayer Perceptron
- CNN
- RNN
- LSTM

Advantages

- Learns automatically & adapts to data
- Great at pattern & image recognition
- Can handle complex nonlinear relationships
- Supports parallel computation (fast processing)

Limitations

- Needs large Datasets for training
- Computationally expensive
- Work like a black box
- Can overfit if not regularized properly

Date: __/__/20__ Day: _____

Difference b/w ANN & BNN

Both Biological Neural Networks (BNNs) & Artifical Neural Networks (ANNs) work with neurons

- BNN: Natural networks in living beings
- ANN: Computer models inspired by brain

They differ in structure, learning, processing and adaptability

Artifical Neural Networks (ANNs)

Defination: Computational models inspired by human brain

Structure: Layers → Input → Hidden → output

Working: Data Flows Forward (Feed-forward) - no loops

Advantages

- Versatile learning: Works with both linear & non-linear data.
- Forcasting Power: Detects patterns for the prediction (e.g. stock prices)

Disadvantages

- Black Box: Hard to interpret inner decisions
- High Computation: Needs strong hardware (GPUs / TPUs)

Biological Neural Networks (BNNs)

Natural neuron networks in the human brain and animals

- Structure: Dendrites → Cell body (soma) → Axon → Synapse
  - Dentrites receive signals
  - Soma process them
  - Axon sends output

Date: __/__/20 Day: _____

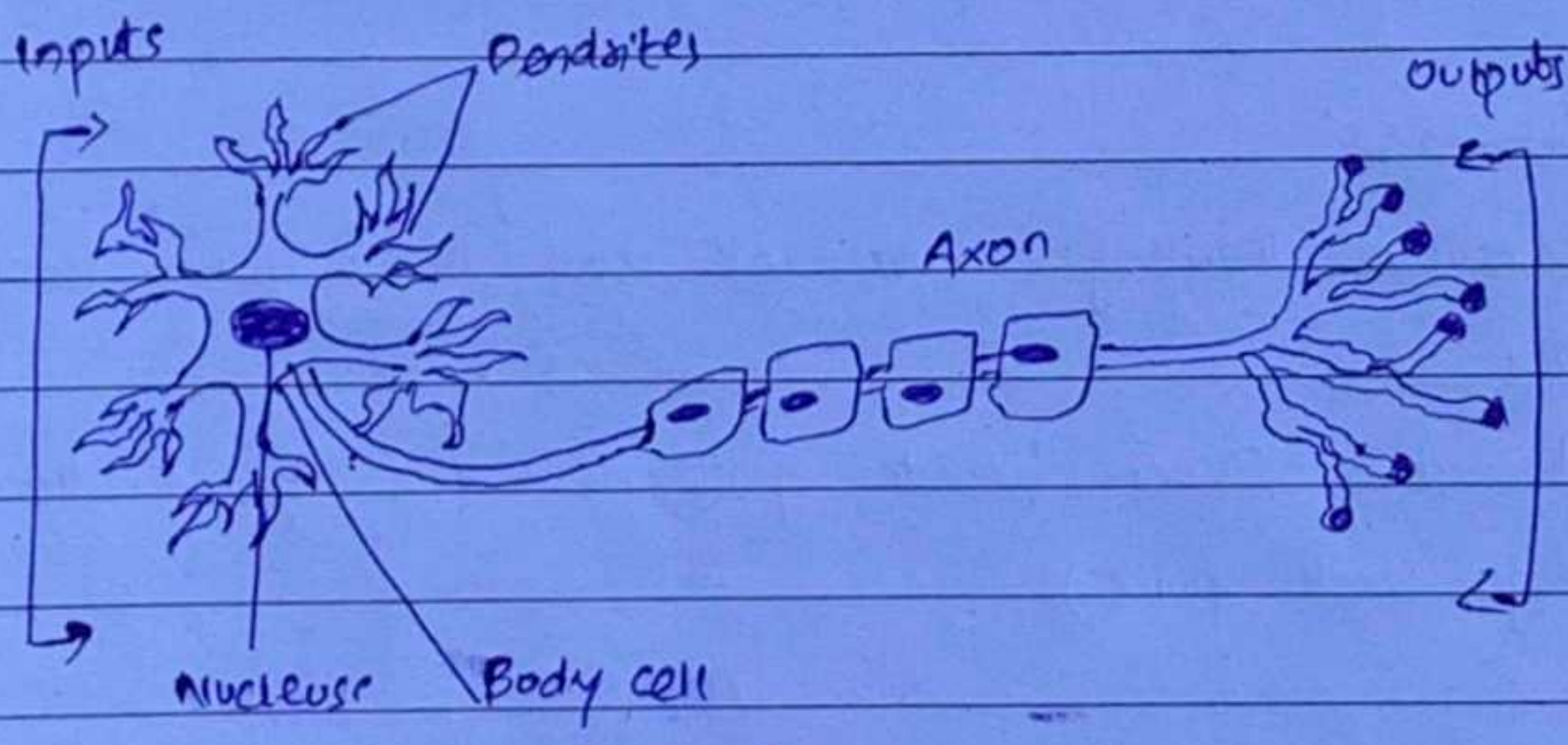
Advantages

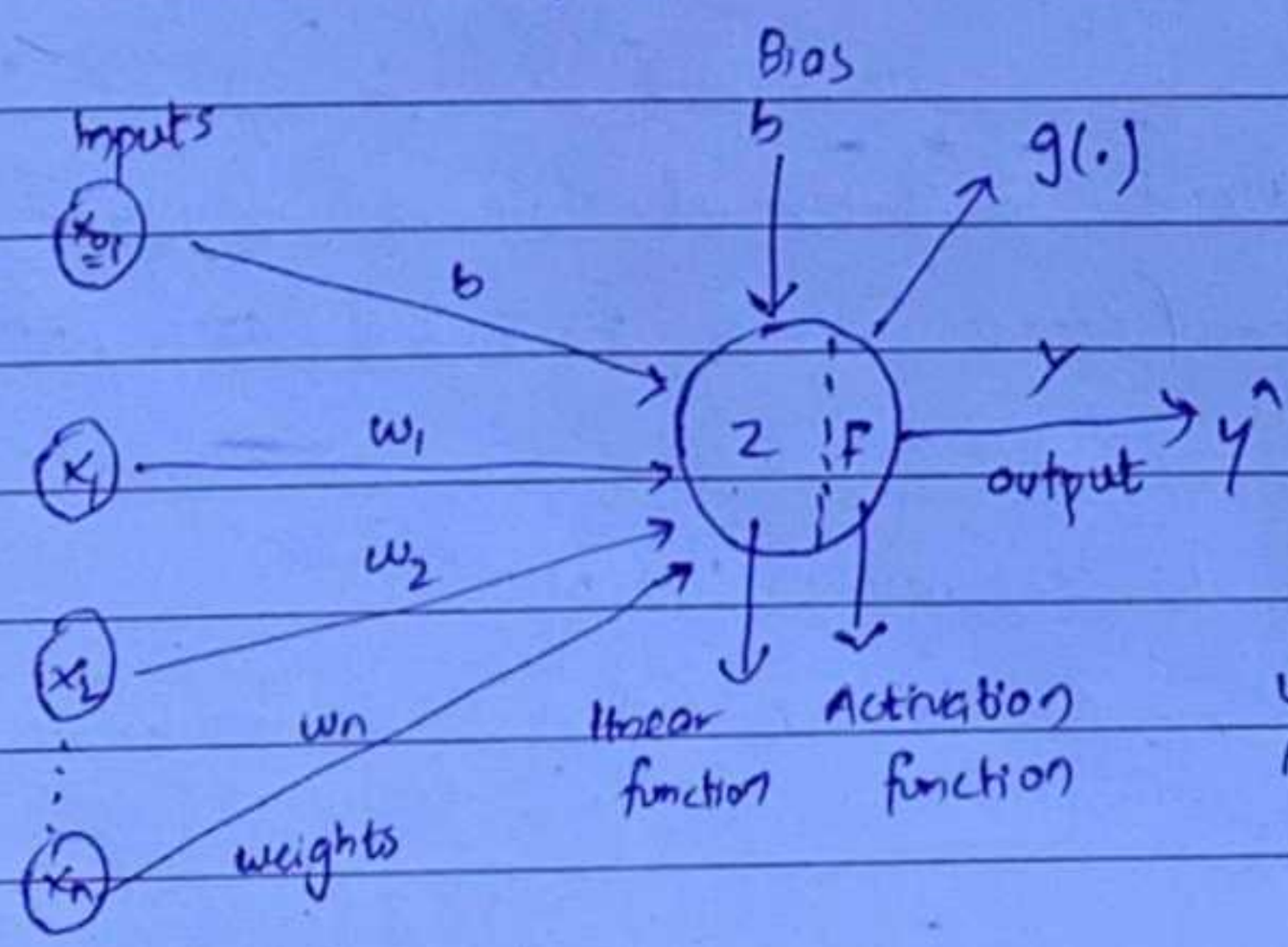
- Flexible Input Handling: Can Interpret noisy or complex signals
- Parallel Processing: Processes many signals at once
  → faster reactions

Disadvantages

- No Central Control: No single CPU: Control is distributed
- Slower Transmission: Electrochemical signals are slower than digital

Key Difference B/W ANNs & BNNs





$\hat{y} = g(w \cdot x + b)$

Date: __/__/20 Day: _____

Single Layer Perceptron

Single Layer Perceptron (SLP) is inspired by biological neurons

A neurons takes input, processes it, and produces an input

Steps:

- Receive signal (input)
- Process signal & decide if it should send output
- Send output to another neuron or cell

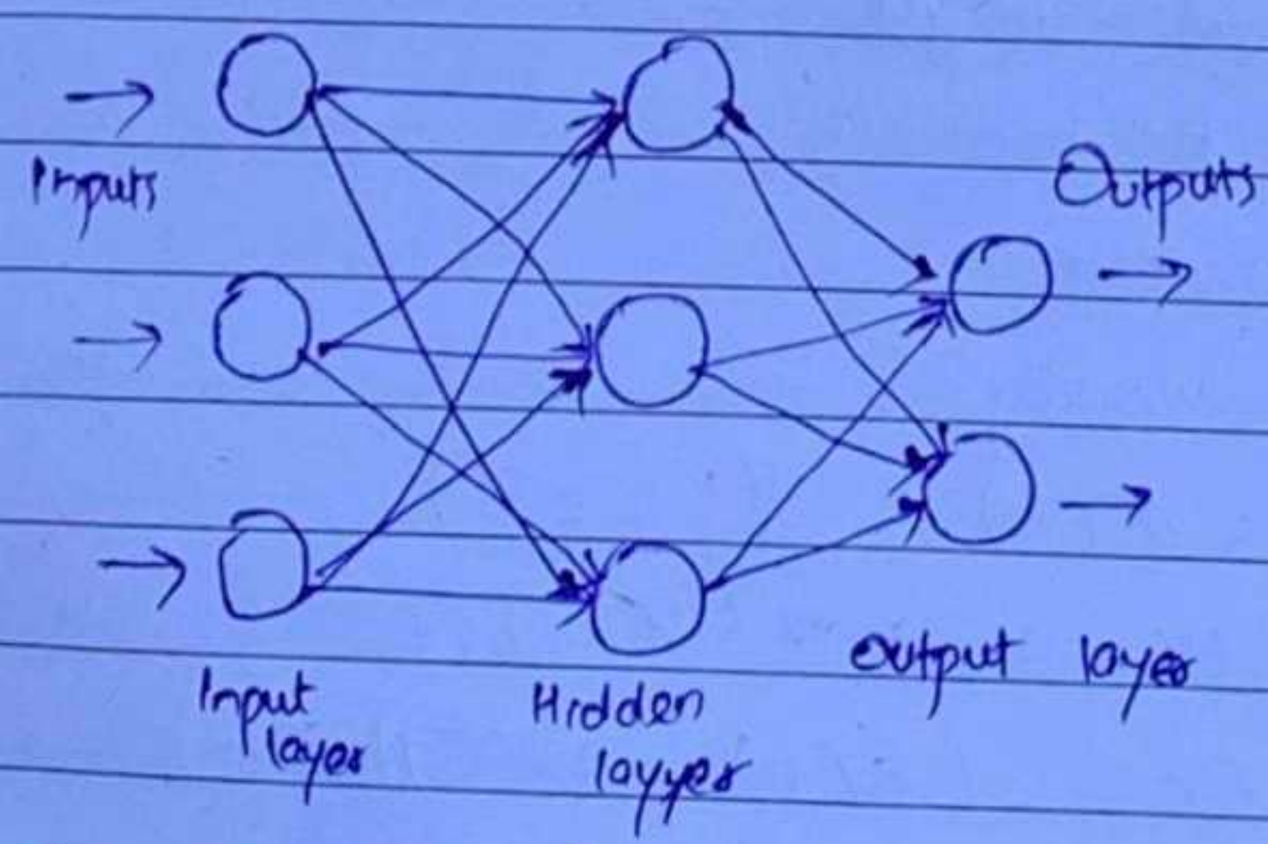
What is Single Layer Perceptron?

- Introduced by Frank Rosenblatt (1958)
- The simplest neural network, also called an artifical neuron
- Mainly used for binary logic problems (AND, OR, NOR)

Main Functions

- Take input
- Multiply by weights & sum them
- Pass sum through an activation function (sigmoid, ReLU)
- Produce output

Structure



Input layer → weight Sum → Activation → Output
(only one layer of neurons → no hidden layers)

Date: __/__/20__ Day: _____

Multi-layer Perceptron (MLP)

What is MLP?

- MLP: Multi-layer Perceptron → a fully connected neural network
- Has input layer, one or more hidden layers, & output layer
- Purpose: learn Complex input-output relationships

Components of MLP

Input layer

Hidden layer

Output layer

→ Fully Connected: Every neurons connects with all neurons in next layer

Working of MLP

- Forward Propogation

Data moves from input → hidden → output

- Weighted Sum

$z = \sum w_i x_i + b$

  - x_i : input
  - w_i : weight
  - b : bias

- Activation Funtion:

Adds non-linearity

- Sigmoid: $\sigma(z) = 1/(1 + e^{-z})$
- ReLU: $f(z) = \max(0, z)$
- Tanh: $(z) = (2/(1 + e^{-2z})) - 1$

- Backpropagation

Used to update weights to minimize loss

Date: __/__/20__ Day: _____

Steps

- Compute gradients of loss w.r.t each weight
- Send error backward layer by layer
- Update weights using Gradient Descent

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

- η : learning rate

- Optimization

Optimizers adjust weights efficiently

- SGD (Stochastic Gradient Descent):

  Updates weights after small batches

- Adam Optimizer

  Uses momentum + adaptive learning rate

Formula (Adam)

- $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
- $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

Loss Function

Measures how far predictions are from actual results

- For Classifications

  Binary Cross-Entropy

$$L = -\frac{1}{N} \sum [y \log(\hat{y}) + (1-y) \log(1-\hat{y})]$$

- For Regression

  Mean Squared Error [MSE]

$$MSE = \frac{1}{N} \sum (y - \hat{y})^2$$

Advantages	Disadvantages
Versatile: work for class/reg	Slow training on Big data
Handles Non-linearity	Can overfit without regularization
Fast Training with GPU	Needs normalization data for
parallel computation	best results

Date: __/__/20 Day: _____

Artifical Neural Networks (ANNs)

What are ANNs?

- Artifical Neural Networks are computer systems that mimic the human brain
- Like the brain neurons, artifical neurons process data, find patterns, and make predictions
- ANNs have layers of interconnected neurons that work to-gether to solve problems
- They can learn from data just as the brain learns from experience
- Used in: image recognition, speech, finance healthcare etc.

Key Components

- Input Layer
- Hidden layers
- Output layer

How ANNs work?

- Input layer: Takes data (image, text, etc)
- Hidden layers: Neurons perform calculations and pass results forward
- Output layer: Produces final prediction
- Backpropagation: Adjusts weights b/w neurons when an error occurs.
  - Reduces errors & improves accuracy

Training and Testing

- Training: Networks learns patterns from labeled data
- Testing: Checked on new data to evaluate accuracy
  - Good training: → better generalization on unseen data.

Date: __/__/20 Day: _____

How ANNs learn?

- ANNs learn from many examples
- Examples: To recognize cats → train on thousands of cat images
- Backpropagation: Adjusts weights after each wrong prediction
- Process repeats: model learns patterns & minimizes error
- End goal → accurate predictions on new data

Common Activation Functions

- Sigmoid : 0 → 1; used in binary classification
- ReLU: Returns positive input, else 0, prevents vanishing gradient
- Tanh: -1 → 1; used in hidden layers for wider range
- Softmax: Converts outputs into probabilities
- Leaky ReLU: Allows small negative values; avoids dead neurons

Types of ANNs

- Feed forward Neural Network (FNN):
  - Data flow one way: Input → output
  - No loops, no back connections
  - Used for simple classification / regression

- Convolutional Neural Network (CNN):
  - Works with grid-like data (images)
  - Extracts features like edges, textures
  - Used in image & speech recognition

- Radial Basic Function Network (RBFN)
  - Works with circular / radial data patterns
  - Used for classification / regression with trends

Date: __/__/20 Day: _____

- Recurrent Neural Network (RNN):
  - Handles sequential / time-based data
  - Has memory (feedback loops)
  - Used in text, speech & forcasting.

Optimization Algorithms

Used to minimize errors & improve training

- Gradient Decent:
  - Basic methods, updates weights using gradients
- Adam:
  - Adaptive learning rate + momentum (use in dL)
- RMSprop:
  - Adjusts learning rate based on recent gradients
- SGD:
  - Updates one sample at time → faster but noisier

Applications of ANNs

- Social Media
- Marketing & sales
- Healthcare
- Virtual Assistants
- Finance
- Customer Support

Challenges in ANNs

- Data Dependency
- High computation
- Overfitting
- Interpretability
- Training Time

Date: __/__/20__ Day: _____

Types of Neural Networks

What are Neural Networks?

Neural Networks are computer models inspired by the human being brain

They Process Input data through layers of neurons to produce outputs

Used for learning patterns, predictions & decision-making.

1- Feedforward Neural Network (FNN)

- Flow: Data move one way — Input → hidden → output
- Training: Backpropagation reduces prediction errors
- Use: Classification, regression
- Apps: Image/voice recognition, NLP, forecasting.
- Best When: Data is static (no sequence)

2- Convolutional Neural Network (CNN)

- Purpose: Works on grid data like Images & videos
- Layers: Convolution + Pooling + fully Connected.
- Apps: Image Classification, object detection, self driving
- Best When: Working with Image/video data

3- Recurrent Neural Network (RNN)

- Purpose: Handles sequential Data
- Structure: loops back to remember previous inputs
- Problem: Vanishing gradient
- Apps: language translation, text, speech, time series
- Best when: Data is sequential (time or text-based)

4- Long Short-term Memory (LSTM)

- Type: Advanced RNN with memory cells

Date: __/__/20 Day: _____

- Fixes: Solves vanishing gradients; remembers long dependencies
- Apps: Text translation, time series forecasting
- Best when: long-term memory is needed

5- Gated Recurrent Unit (GRU)
- Types Simplified LSTM with fewer parameters
- Advantages: Faster, handles vanishing gradient well
- Apps: Speech recognition, text monitoring
- Best when: Need LSTM performance with less computation

6. Radial Bias Function Network (RBFN)
- Idea: Uses radial basis functions for approximation
- Apps: Regression, pattern recognition, control systems
- Best When: Small / medium data

7- Self-Organizing MAP (SOM)
- Type: Unsupervised learning
- Goal: Reduce high-dimensional data → 2D map
- Apps: Data visualization, clustering, feature selection
- Best when: Doing data visualization or clustering

8- Deep Belief Network (DBN)
- Made of: Multiple layers of hidden (latent) variables
- Works for: Both supervised & unsupervised learning
- Apps: Image & voice recognition, NLP, recommendation
- Best When: Need unsupervised pre-training or feature extraction

9- Generative Adversarial Network (GAN)
- Made of: 2 networks → Generator + Discriminator
- Working: Both compete & improve over time
- Apps: Image generation, style transfer, data augmentation
- Best when: Need realistic synthetic data

Date: __/__/20 Day: _____

10. Autoencoder (AE)

- Type: ~~Feedback~~ Forward network that learns to compress & reconstruct data
- Parts: Encoder → compress; Decoder → rebuilt
- Variants: Undercomplete, Overcomplete, Variational Autoencoder
- Apps: Noise removal, dimensionality reduction, anomaly detection
- Best When: Doing unsupervised learning like compression/denoising

11. Transformer Network

- Core Idea: Uses self-attention for parallel processing of data
- Strength: Captures long-range dependencies better than RNN
- Apps: Translation, text generation, summarization etc
- Best when: Doing NLP or long-sequence tasks

12. Siamese Network

- Idea: Two identical networks compare inputs via similarity metrics
- Apps: Face verification, image similarity, signature verification
- Best when: Need to compare two inputs for similarity

13. Capsule Network (CapsNet)

- Idea: Captures part-to-whole relationships in images
- Better: Handles spatial hierarchies missed by CNNs
- Apps: Image classification, object detection
- Best when: Image relationships matter.

14. Spiking Neural Network (SNN)

- Inspired by: Biological neurons using electrical "spikes".
- Used in: Neuromorphic (brain-like) computing
- Apps: Brain modeling, cognitive simulations, etc
- Best When: Working on biologically inspired or neuromorphic systems

Date: __/__/20 Day: _____

Neural Network Architecture

Neural Network Architecture

A neural Network (ANN) is made of many connected artifical neurons that pass signals through weighted connections

They way neurons are connected defines the architecture.

Single-layer Feedforward Network

- Simplest form of ANN
- Has 2 layers.
  - Input layer (takes data, no processing)
  - Output layer (does calculations)
- Signals move only forward → from input → output
- Hence:
  - Single-layer : Only one layer does computation
  - Feedforward = one-way signal flow

Net input

$$y_{ink} = \sum_{i=1}^{m} x_i w_{ik}$$

output depends on the activation function

Multi-layer FeedForward Network

- Has input, one or more hidden layers & output layer
- Hidden layers add more computation power
- Example:

  Input (m neurons) → Hidden (n neurons) → output (r neurons)

Hidden layer

$$y_{ink} = \sum_{i=1}^{m} x_i w_{ik}$$

Output layer input

$$z_{ink} = \sum_{i=1}^{n} y_{out_i} w'_{ik}$$

Date: __/__/20__ Day: _____

Competitive Network

- Structure like single-layer feedback
- But output neurons are interconnected & compete with each other
- Used in supervised learning (like clustering)
- Only one neuron "wins" to represent the input

Recurrent Network (RNN)

- Unlike feedback forward, it has feedback loops
- Outputs can return as input to previous layers
- Used for sequential or time-based data

Learning Process in ANN

Learning depends on:

- Number of layers → Single or Multi-Layer
- Direction of signal flow → Feedback forward or Recurrent
- No. of Nodes:
  - Input nodes = no. of features
  - Output nodes = no. of classes
  - Hidden nodes = chosen by user (too many - overfitting)
- Weights B/w Neurons
  - Weights determine learning
  - Start with random value → adjust iteratively → stop when error < 1% or after max iterations

Goal: Find the best weights, no. of layers, & nodes for minimum error

Key Observation

A network with even one hidden layer can approximate any continuous function.

Usually trained using Backpropagation

Date: __/__/20 Day: _____

Backpropagation Algorithm

Idea:

- Introduced in 1986
- Used to train multi-layer feed~~back~~forward networks
- Works by sending errors backward to adjust weights

Objective: Minimize the diff b/w predicted output & actual target

Two Phases per Epoch:

- Forward Phase
  - Input → hidden → output
  - Signals pass through weights & activation function
  - Output is generated
- Backforward Phase
  - Output is compared to target
  - Error is sent backward
  - Weights are adjusted to reduce error

Key Formulas

Hidden layer input

$$y_{ink} = w_{0k} + \sum_{i=1}^{m} x_i w_{ik}$$

Hidden layer output

$$y_{out_k} = f_y(y_{ink})$$

Output layer input

$$z_{ink} = w'_{0k} + \sum_{i=1}^{n} y_{out_i} w'_{ik}$$

Output layer output

$$z_{out_k} = f_z(z_{in_k})$$

cost Function (Error)

$$E = \frac{1}{2} \sum_{k=1}^{n} (t_k - z_{out_k})^2$$

where;

- t_k = Target value
- z_{out_k} = Predicted output

Date: __/__/20 Day: _____

Gradient Descent (weight update)

we minimize E by adjusting weights

Derivative for weights (hidden → output)

$$\frac{\partial E}{\partial w'_{jk}} = -(t_k - z_{out_k}) \cdot f'_z(z_{in_k}) \cdot y_{out_i}$$

Define:

$$\delta_k = -(t_k - z_{out_k}) \cdot f'_z(z_{in_k})$$

Then:

$$\frac{\partial E}{\partial w'_{jk}} = \delta_k \cdot y_{out_i}$$

Weight & Bias Update Rules

For Hidden → output Weights

$$\Delta w'_{jk} = -\alpha \cdot \delta_k \cdot y_{out_i}$$

$$w'_{jk(new)} = w'_{jk(old)} + \Delta w'_{jk}$$

Bias Update:

$$\Delta w'_{0k} = -\alpha \cdot \delta_k$$

$$w'_{0k(new)} = w'_{0k(old)} + \Delta w'_{0k}$$

For Input → Hidden Weights

$$\Delta w_{ij} = -\alpha \cdot \delta_j \cdot x_{out_i}$$

$$w_{ij(new)} = w_{ij(old)} + \Delta w_{ij}$$

Bias Update:

$$\Delta w_{0j} = -\alpha \cdot \delta_j$$

$$w_{0j(new)} = w_{0j(old)} + \Delta w_{0j}$$

Learning Rate (α)

- Controls how Fast weights change
- Too high - may diverge
- Too slow - too slow to converge
- This is called learning rate

Date: __/__/20 Day: _____

Basic Components of Neural Networks

Layers in Neural Networks

In an Artificial Neural Network(ANN), data moves from the Input layer → Hidden layers → Output layers
Each layer has neurons that process data and pass results forward
Together, these layers extract features, transform data, & make predictions

Three Main Layers in ANN

- Input layer
  - The first layer – receives raw input data
  - Each neuron = one features (e.g pixel value in an image)
  - It doesn't do any computation – only passes data forward

Key points
- Role → Take raw data
- Function → Send it to hidden layers
- Example → Image input → each pixel = one neuron

- Hidden layer
  - These are intermediate layers b/w input & output
  - Main computation happens here
  - No. of hidden layers depends on problem complexity
  - Each layer applies weights + biases + activation function to introduce non-linearity

Role: Extract features & learn patterns
Function: Apply weights & activation to data

- Output layer
  - Final layer – gives predictions or final outputs
  - No. of neurons = no. of target outputs or classes

Date: __/__/20__ Day: _____

Activation Function depends on problem type:

- Softmax → Multi-class classification
- Sigmoid → Binary classification
- Linear → Regression problems

Types of Hidden Layers in ANN

- Dense (fully Connected layer)
  - Most common type
  - Even neuron connects to all neurons in previous & next layer
  - Performs weighted sum + activation (ReLU, Sigmoid)

Role: Learns strong representations from input

Function: Weighted Sum + non-linear activation

→ Used in → All types of ANNs (MLPs, CNNs, etc)

- Convolutional Layer (CNN)
  - Used in image processing tasks
  - Applies filters/Kernals to scan input & create feature maps
  - Detects edges, textures, & visual patterns

Role: Extracts spatial features

Function: Convolution using filters (Kernals)

→ Used in → CNNs for computer vision

- Recurrent Layer (RNN)
  - Used for sequential data (like text, speech, time Series)
  - Has loop connections → allows past information to influence current output
  - Keeps memory (context) across time steps

Role: Handles temporal dependencies

Function: Maintains State across sequences

→ Used in → RNN, LSTM, GRU models.

Date: __/__/20 Day: _____

- Dropout layer
  - Used to reduce overfitting during training
  - Randomly drops some neurons (with probability p) during training
  - Forces the network to learn robust, general features instead of memorizing

Role: Regularization

Function: Randomly deactivate neurons

→ Used in → All networks (especially deep ones)

- Pooling Layer
  - Reduces the size (spatial dimensions) of data
  - Keeps important features & removes unnecessary details
  - Common types:
    - Max Pooling: Keeps the largest value
    - Average Pooling: Takes the mean value

Role: Dimensionality Reduction

Function: Downsampling to reduce computation

→ Used in → CNNs after Convolutional layers

- Batch Normalization Layer
  - Normalizes data from previous layer using mean & standard deviation
  - Keeps training stable & faster by controlling ~~internet~~ internal changes in activations

Role: Stabilize & speech up training

Function: Normalize layer outputs

→ Used in → Deep CNNs, RNNs, & MLPs.

Date: __/__/20__ Day: _____

Weights & Bias in Neural Network

Neural Network learn from data to find patterns used in image recognition, NLP, automomous systems, Two key parts make this possible: Weights & Biases Together, they control how network learns & predicts

Weights

- Defination: Numbers that control how much influence each inputs has on the output
- Purpose: Inputs are multiplied by weights → higher weight = higher influence
- Learning: Updated during training using algorithms like Gradient Descent to reduce prediction error
- Generalization: Well-trained weights help the model perform well on new data too
- Example: House prediction, etc

Biases

- Defination: Extra parameters that shift the neuron's output (not linked to input)
- Purpose: Allow neurons to activate even if all inputs are zero
- Function: Move the activation function so the model fits data better
- Training: Updated with weights using backpropagation
- Example: land cost

How Neuron Networks Learn?

- Forward Propagation
- Backpropagation

Date: __/__/20__ Day: _____

Real - World Application

- Image Recognition
- NLP
- Autonomous Vehicles
- Healthcare

Advantages

- Learn from Data
- Flexibility
- Improved Accuracy
- Better Generalization
- Enhanced Learning

Challanges

- Poor Initialization
- Overfitting
- Complex Tuning
- Black Box Problem
- High Computational Cost

Forward Propagation

Forward propagation is the process where input data passes through all layers of a neural network to produce an output

It's the first step before training (backpropagation) and helps the model make predictions using weights & Biases

How it Works (Step - by - Step)

- Input layer
  - Data enters the network here

Date: __/__/20__ Day: _____

- Each feature = one neuron
- Inputs are often normalized or standardized for better results

- Hidden layers
  - Data passes through one or more hidden layers
  - Each neuron does:

$$Z = W \times X + b$$

where:

- $W \rightarrow$ weight
- $X \rightarrow$ Input
- $b \rightarrow$ bias

- Then, the activation function (Like ReLU or Sigmoid) is applied to make the output non-linear

- Output layer
  - Produces the final prediction
  - Activation depends on task:
    - Softmax $\rightarrow$ multi-class classification
    - Sigmoid $\rightarrow$ binary classification
    - Linear $\rightarrow$ regression

- Prediction
  - The output is compared with actual values & using a loss function to calculate the error

Math Behind Forward Propagation

For a 3-layer neural network

- First Hidden layer

$$A_1 = \sigma(W_1 X + b_1)$$

Date: __/__/20 Day: _____

- Second Hidden Layer

$A_2 = \sigma(W_2 A_1 + b_2)$

- Output layer

$Y = \sigma(W_3 A_2 + b_3)$

In Short:

Output = $\sigma(\sigma(\sigma(XW_1 + b_1)W_2 + b_2)W_3 + b_3)$

- Weights (W): control importance of inputs
- Bias (b): helps shift activation
- σ (Activation): adds non-linearity

Activation Functions

An activation function decides if a neuron should activate or not

It adds non-linearity to the network - helping it learn complex data patterns

Without it, even deep networks act like linear algebra

Why activation Functions Are Important?

- Real-world data is non-linear
- Non-linear activations let models form curved decision boundaries
- They help solve tasks like image recognition, NLP, & speech processing
- They make backpropagation possible by providing gradients

Simple Mathematical View

For a small network

$h1 = i1 * w1 + i2 * w3 + b1$

$h2 = i1 * w2 + i2 * w4 + b2$

output = $h1 * w5 + h2 * w6 + bias$

Date: __/__/20 Day: _____

Without activation → linear

With activation (e.g Sigmoid) → non-linear

Final output = σ(h1*w5 + h2*w6 + bias)

This non-linearity helps the network learn complex relationships

Types of Activation Function (Linear Activation Fun)

Formula : $y = x$

Range: $(-\infty, +\infty)$

- Gives linear output – limited learning power
- Used mostly in output layer for regression
- Not good for hidden layers (makes network behave linearly)

Non-Linear Activation Functions

(a) Sigmoid Function

Formula

A = 1/(1 + e^(-x))

Range (0, 1)

- Converts values into probabilities
- Best for binary classification
- Sensitive b/w -2 to +2
- May cause vanishing gradient in deep networks

(b) Tanh (Hyperbolic Tangent)

Formula

f(x) = tanh(x) = (2/(1 + e^(-2x))) -1

or

tanh(x) = 2 × sigmoid(2x) -1

Range (-1, +1)

- Zero-centered → easier learning
- Better than sigmoid for hidden layers
- Still can suffer from vanishing gradient

Date: __/__/20__ Day: _____

(c) ReLU (Rectified Linear Unit)

Formula

$A(x) = \max(0, x)$

Range : $[0, \infty)$

- Outputs x if positive, else 0
- Fast & efficient - simple computation
- Solves vanishing gradient issue
- Makes network sparse (not all neurons activate)
- Most common in hidden layers

(d) Leaky ReLU

Formula,

$f(x) = x$ (if $x > 0$), else αx ($\alpha \approx 0.01$)

Range : $(-\infty, \infty)$

- Fixes "dying ReLU" issue
- Allows small gradient for negative inputs
- Keeps better gradient flow during training

Exponential Linear Units (ELUs)

(a) Softmax Function

- Used for multi-class classification
- Converts raw scores → probabilities (sum = 1)
- Each class gets a probability → highest one = prediction
- Non-linear & ideal output-layer in the classification.

(b) Softplus Function

Formula :

$A(x) = \log(1 + e^x)$

Range: $(0, \infty)$

Date: __/__/20__ Day: _____

- Smooth version of ReLU (no sharp edges)
- Always positive & differentiable everywhere
- Avoids sudden jumps of ReLU
- Good for stable gradient updates

Impact on Model Performance

Factor	Effect of Activation Function
Convergence Speed	ReLU trains faster, Sigmoid/Tanh → Slower
Gradient Flow	ReLU keep gradients alive, Sig/Tanh shrink thru hidden layer
Model Complexity	Softmax enables multiclass outputs, ReLU good for
Learning Efficiency	Right activation = faster, more stable learning

Formula Tables

Function	Formula	Range	Use Case
Linear	$y = x$	$(-\infty, \infty)$	Regression output
Sigmoid	$1/(1+e^{-x})$	$(0,1)$	Binary classification
Tanh	$(2/(1+e^{-2x}))-1$	$(-1, 1)$	Hidden layers
ReLU	$\max(0, x)$	$[0, \infty)$	Hidden layers
Leaky ReLU	x if $x>0$ else αx	$(-\infty, \infty)$	Hidden layers
Softmax	$e^x / \Sigma e^x$	$(0,1)$	Multi-class output
Softplus	$\log(1+e^x)$	$(0, \infty)$	Smoth ReLU alternative

Loss Function

What is a Loss Function?

A loss function measures how wrong the model's predictions are compared to the real values.
It gives a single number - smaller no = better model
Used in training to improve the model.

Why its Important?

- Guides Training: Algorithms like Gradient Descent

Date: __/__/20 Day: _____

use loss to adjust weights & reduce error

- Measures Performance: Shows how close predictions are to actual results
- Affects Learning Behaviour: Different losses make the model learn differently

Types of Loss Functions

- Regression Loss Functions

used when output is a continuous number (e.g price, age)

Mean Squared Error (MSE)

Measures average of squared errors

Formula

$$MSE = \frac{1}{n} \sum (y_i - \hat{y}_i)^2$$

very common

Sensitive to Outliers (because errors are squared)

Mean Absolute Error (MAE)

Average of absolute differences

$$MAE = \frac{1}{n} \sum |y_i - \hat{y}_i|$$

Less sensitive to Outliers

Not Smooth at zero (can cause optimization issues)

Huber Loss

Mix of MSE & MAE

$$L = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & |y_i - \hat{y}_i| \leq \delta \\ \delta |y_i - \hat{y}_i| - \frac{1}{2}\delta^2 & |y_i - \hat{y}_i| > \delta \end{cases}$$

less Sensitive to Outliers than MSE

Smooth & differentiable

Date: __/__/20 Day: _____

- Classification Loss Function

Used when output is a label or class

- Binary Cross-Entropy (Log Loss)

For binary classification (0 or 1)

$$-\frac{1}{n}\sum\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

Output b/w 0 & 1 (probability)

- Categorical Cross-Entropy

For multi-class classification

$$-\sum y_{ij} \log(\hat{y}_{ij})$$

Uses one-hot encoded labels

- Sparse Categorical Cross-Entropy

Same as above, but labels are integers (not one-hot)

Efficient for many classes

- Kullback-Leibler (KL) Divergence

Measures how one probability distribution differs from another

$$\sum y_{ij} \log \frac{y_{ij}}{\hat{y}_{ij}}$$

Used in probabilistic models like VAEs

- Hinge Loss

Used for SVMs (Support Vector Machines)

$$\frac{1}{n}\sum \max(0, 1 - y_i \cdot \hat{y}_i)$$

for binary classification

works with labels -1 and $+1$

Date: __/__/20 Day: _____

- Ranking Loss Functions

Used when model predicts relative order (e.g recommendation)

- Contrastive Loss

Used in Siamese networks (learn similarity)

$$\frac{1}{2N} \sum \left[y_i d_i^2 + (1-y_i) \max(0, m - d_i)^2 \right]$$

$y_i = 1$ (similar), 0 (dissimilar)

- Triplets Loss

Used in embedding learning (anchor, positive, negative)

$$\frac{1}{N} \sum \left[\| f(a) - f(p) \|^2 - \| f(a) - f(n) \|^2 + \alpha \right]_+$$

Ensures positive is closer to anchor than negative

- Margin Ranking Loss

Keeps correct order with a margin

$$\frac{1}{N} \sum \max(0, -y_i (s_i^+ - s_i^-) + \text{margin})$$

- Image & Reconstruction Loss Functions

Used in image classification / segmentation

- Pixel-wise Cross-Entropy

Each pixel treated as classification

$$-\frac{1}{N} \sum y_{i,c} \log(\hat{y}_{i,c})$$

Used for image Segmentation

- Dice Loss

Measures overlap b/w predicted and true mask

$$1 - \frac{2 \sum y_i \hat{y}_i}{\sum y_i + \sum \hat{y}_i}$$

works well for imbalanced data

Date: __/__/20 Day: _____

- Jaccard (IoU) Loss

Similar to Dice, measures intersection over union

$$1 - \frac{\sum y_i \hat{y}_i}{\sum y_i + \sum \hat{y}_i - \sum y_i \hat{y}_i}$$

- Perceptual Loss

Compares features (not pixels) using pre-trained models
Used in image generation (e.g SRGANs)

- Total Variation Loss

Encourages smoothness in images by penalizing sharp pixel changes

$$\sum (y_{i,j+1} - y_{i,j})^2 + (y_{i+1,j} - y_{i,j})^2$$

- Adversarial Loss Functions (for GANs)

GAN Loss

$$\min_G \max_D \left[E_x[\log D(x)] + E_z[\log(1 - D(G(z)))] \right]$$

- D tries to detect fakes, G tries to fool D

- Least Squares GAN (LSGAN)

Makes GAN training stable using squared error

Discriminator

$$\frac{1}{2} E_x[(D(x) - 1)^2] + \frac{1}{2} E_z[D(G(z))^2]$$

Generator

$$\frac{1}{2} E_z[(D(G(z)) - 1)^2]$$

- Specialized Loss Functions
- CTC (Connectionist Temporal Classification)

For sequence prediction (speech, handwriting)

$$L = -\log(P(y|x))$$

Date: __/__/20__ Day: _____

- Poisson Loss

for count data (e.g, event counts)

$$\sum(\hat{y}_i - y_i \log(\hat{y}_i))$$

- Cosine Proximity Loss

Encourages similar direction b/w vectors

$$-\frac{1}{N} \frac{y_i \cdot \hat{y}_i}{\|y_i\| \, \|\hat{y}_i\|}$$

- Earth Mover's (Wasserstein) Loss

Used in Wasserstein GANs (WGAN) to improve stability

$$E_x[D(x)] - E_z[D(G(z))]$$

Backpropagation

Backpropagation (Backward Propagation of Errors) is a training algorithm for neural networks
It reduces the error b/w predicted and actual output by sending the error backward through the network and updating weights & biases using the chain rule of calculus

Why its important?

- Efficient Weight update:
  - Finds how much each weight affects the error and updates efficiently
- Scalable:
  - Works for deep & large networks
- Automated Learning:
  - Adjusts itself to improve prediction accuracy

Date: __/__/20 Day: _____

Working Steps

- Forward Pass
  - Input data ~~does~~ goes layer by layer (Input → hidden → output)
  - Each neurons:
    - Multiplies inputs by weights
    - Adds bias
    - Passes through activation (e.g:, ReLU, Sigmoid)
  - Final layer gives output (e.g softmax for classification)

- Backward Pass
  - Error = (Prediction − Actual)
  - Example loss: Mean Squared Error [MSE]

MSE = $(\text{Predicted} - \text{Actual})^2$

- Using the chain rule, gradients (∂loss/∂weights) are found
  - These gradients tell how to adjust weights to reduce error
  - Repeat until the loss becomes small (model learns)

Example (Manual Calculation)

Forward Propagation

Weight sum for each neuron

$a = \Sigma(w * x)$

Activation (Sigmoid)

$y = 1/(1 + e^{\wedge} - a)$

Example

$a1 = (0.2 \times 0.32) + (0.2 \times 0.7) = 0.21$

$y^3 = 1(1 + e^{\wedge} - 0.21) = 0.56$

Similary

$y^4 = 0.59$, $y^5 = 0.67$

Actual output = 0.5 Error = 0.5 − 0.67 = −0.17

Date: __/__/20 Day: _____

Backward Propagation

Output layer Error

$\delta 5 = y5(1-y5)(y_{target} - y5)$

$\delta 5 = 0.67(1-0.67)(0.17) = -0.0376$

Hidden Layer Error

$\delta 3 = y_3(1-y_3)(w13 \times \delta 5) = -0.0027$

$\delta 4 = y_4(1-y_4)(w23 \times \delta 5) = -0.0819$

Update Weights

$\Delta w = \eta \times \delta \times$ output

Example

$\Delta w23 = 1 \times (-0.0376) \times 0.59 = -0.022$

New $w23 = 0.9 - 0.022 = 0.8778$

After updating all weights → new forward pass gives

$y3 = 0.57$, $y4 = 0.56$, $y5 = 0.61$

Error reduces from $-0.17 \rightarrow -0.11 \rightarrow$ continues until accurate

Advantages

- Easy to Implement
- Works for Simple & deep networks
- Fast learning via gradient updates
- Improves generalization & accuracy
- Scales to large datasets

Challenges

- Vanishing Gradient: become too small (sigmoid/tanh)
- Exploding Gradient: Gradients become large (unstable training)
- Overfitting: Model memorizes training data instead of generalizing.

Date: __/__/20 Day: _____

Learning Rate

learning rate (α or η) is a hyperparameter that controls how fast or slow a neural network learns

It decides the step size for updating weights during training to minimize the loss function

Formula In Gradient Descent)

$$w = w - \alpha \times \nabla L(w)$$

Where

$w \rightarrow$ weight

α (alpha) $\rightarrow$ learning rate

$\nabla L(w) \rightarrow$ gradient of loss function

Impact on Model

low learning rate

Slow learning, takes more epochs, but may reach better accuracy

High Learning Rate

learn faster but may overshoot the minimum

$\rightarrow$ unstable or diverging loss

Optimal learning Rate

Balanced — learn faster, stable and accurate

$\therefore$ Goal: Find a learning rate that ensures fast + stable convergences

Best Practices

- Tune the learning rate carefully for each task
- Use learning rate schedules or adaptive methods for better control

Date: __/__/20__ Day: _____

- Avoid too small or too large values

Techniques For Adjusting learning Rate

- Fixed learning Rate
  - Constant value throughout training
  - Easy but not adaptive → may give suboptimal results

- Learning Rate Schedules
  - Learning rate decreases overtime to stabilize training
    - Step Decay: Reduce by a factor every few epochs
    - Exponential Decay: Gradually decreases exponentially
    - Polynominal Decay: Smooth decay using polynomial fun.

- Adaptive Learning Rate Methods

Adjusts learning rate automatically based on gradients

- Ada Grad:
  - Adapts learning rate per parameter
    - Works well for sparse data
    - Can decay too quickly
- RMSprop
  - Fixes AdaGrad's problem by using a moving average of squared gradients
    - Keeps learning rate stable
- Adam (Adaptive Moment Estimation)
  - Combines RMSprop + Momentum
    - Fast, stable, and most commonly used.

- Cyclic Learning Rate
  - Learning rate oscillates b/w a minimum & maximum value
  - Helps model explore loss surface → faster convergence.

Date: __/__/20 Day: _____

- Decaying Learning Rate
- Gradually decreases as training progresses
- Precise steps near minimum → stable training

Optimization Algorithm

- Optimization algorithms in deep learning
- Gradient Descent
- Stochastic Gradient Descent (SGD)
- Batch Normalization
- Mini batch Gradient Descent
- Adam (Adaptive Moment Estimation)
- Momentum-based Gradient Optimizer
- Adagrad Optimizer
- RMSProp Optimizer

What are Optimization Algorithms?
Optimization algorithms help adjust model parameters to minimize error and improve accuracy
They work by iteratively updating model parameters to find the best possible solution

1-First-Order Algorithms
Use first derivative (gradient) of the function
They move in the direction opposite to the gradient to minimize the loss.
∴ Used because they are simple and efficient for large problems

- Gradient Descent
find minimum of a function using gradients
Formula: $x_{new} = x - \text{learning rate} * \text{gradient}(x)$

Date: __/__/20 Day: _____

Variants of Gradient Descent

- SGD (Stochastic): Updates model for each traing sample. Fast for large data
- Mini-Batch: Updates in small batches - balance of speech & accuracy
- Momentum: Adds info from previous steps or to move faster through flat or noisy areas.

- Stochastic Optimization

Adds randomness to escape local minima

- Simulated Annealing: Starts with high randomness (exploration) and slowly cools (exploitation)
- Random Search: Tests random points - works well for high-dimensional problems

Tips:

- Needs many evaluations → can be slow
- Use based on problem type (e.g multiple minima → annealing)

2- Evolutionaly Algorithms

Inspired by natural selection.

Key terms:

- Population: Set of solutions
- Fitness Function: Measures quality of a solution
- Selection, Crossover, Mutation: Used to create better offspring
- Termination: Stops when best found or limit reached

- Genetic Algorithm (GA)

Uses Crossover & mutation to evolve population

- Differential Evolution (DE)

Improves solutions using vector addition & mutation

Date: __/__/20 Day: _____

3- Metaheuristic Optimization

High-level strategies for complex search problems

- Tabu Search: Avoids revisiting old solutions using memory (tabu list)
- Iterated Local Search (ILS): Repeated local search with small random changes to escape local minima

4- Swarm Intelligence Algorithms

Inspired by natural behaviors like flocks or ants

- Particle Swarm Optimization (PSO)

Each particle (solution) moves based on:

- Its best position (personal best)
- Best in Swarm (global best)

- Ant Colony Optimization (ACO)

Ants find shortest paths using pheromones

Steps:

- Each ant creates a path
- Shorter paths get stronger pheromones
- Over time, best paths dominate

5- Hyperparameter Optimization

Finding the best model hyperparameters

- Grid Search: Tries all combinations (accurate but slow)
- Random Search: Random combinations (faster, efficient in large spaces)

6- Deep Learning Optimization

Deep models need special optimizers.

Date: __/__/20

Day: _____

- Adam: Uses moving averages of gradients & squared gradients. Fast & adaptive.
- RMSProp: Adjusts learning rate per parameter using gradient magnitudes. Helps stabilize training

7. Second-Order Algorithms

Use gradient + Hession (curvature)

Faster convergence but expensive for large data

- Newton's Method

Uses both first & Second derivatives

- Quasi-Newton (BFGS, L-BFGS)
- BFGS: Approximates Hession from gradients. Fast like Newton, cheaper to compute
- L-BFGS: Memory-efficient version - works well for large-scale models

8. Constrained Optimization

- Lagrange Multipliers: Convert constrained → unconstrained problems
- KKT Conditions: Handle both equality and inequality constraints

9. Bayesian Optimization

Smart hyperparameter seach using probabilities and past results

Needs fewer iterations than random/grid search

Best for expensive or slow-to-train models

Date: __/__/20 Day: _____

- Optimization For ML Tasks
  - Classification (logistic Regression)
    - Uses cross-entropy Loss
    - Solvers: Gradient Descent, Netwton, etc
  - Regression (Linear Regression)
    - Goal: Minimine (MSE)
    - Solvers: Normal equation, Gradient descent etc

Challanges

- Non-convexity: Many local minima Hard to find global optimum
- High dimensionality: Expensive Search space
- Overfitting: Regularization needed to avoid memorizing training data

Gradient Descent Algorithm

Gradient Descent is an optimization algorithm used to minimize the cost (error) function of machine and deep learning models by iteratively updating model parameters (weights & biases) to make predictions more accurate

Key Idea

Gradient Descent helps models learn by finding parameters that reduce prediction errors — used in:

- Linear Regression
- Logistic Regression
- Support Vector Machine (SVMs)
- Neural Networks

Date: __/__/20 Day: _____

How Its Work

Step 1: Forward Propagation

Compute the model's output using current parameters

Step 2 Backward Propagation

Calculate the gradient (slope) of the loss function with respect to each parameter using chain rule

Step 3 Parameter Update

Use the update rule:

$$w = w - \alpha \frac{\partial J}{\partial w}, \quad b = b - \alpha \frac{\partial J}{\partial b}$$

where:

α = learning rate

J = cost / losts function

Roles in ML Models

- Linear Regression
  - loss: MSE
  - MSE is convex → guarantees a global minimum
  - Updates weights & bias to find the best-fit line minimizing prediction error

- Logistic Regression
  - loss: log loss (Cross - Entropy Loss)
  - Uses the sigmoid function to output probabilities
  - Updates weights to create the best decision boundary between classes

- Support Vector Machines (SVMs)
  - Loss: Hinge Loss
  - Gradient Descent optimizes the hyperplane that maximizes the margin b/w classes

Date: __/__/20 Day: _____

Mathematical Form

- For One example:

$$J(w, b) = \frac{1}{n}(y_p - y)^2$$

Gradients:

$$\frac{\partial J}{\partial w} = \frac{2}{n}(y_p - y)x, \quad \frac{\partial J}{\partial b} = \frac{2}{n}(y_p - y)$$

Update Rule:

$$w = w - \gamma \nabla J(w, b), \quad b = b - \gamma \nabla J(w, b)$$

Learning Rate (α)

learning Rate	Effect
Too Small	Slow learning
Too Large	Overshoots → diverges
Balanced	Fast & stable convergence

Vanishing & Exploding Gradients

Problems during training deep networks

- Vanishing Gradient: Gradients become very small → early layers stop learning
- Exploding Gradient: Gradients become too large → unstable training

Solutions

- Use ReLU activation
- Gradient clipping
- Batch Normalization
- Good weight initialization

Advantages	Disadvantage
• Simple & effective	Sensitive to learning rate
• Works for large models	Can get stuck in local minima
• Many fast variants	Slow for very large datasets
• Scalable & flexible	Depends on good initialization

Date: __/__/20 Day: _____

Types of Gradient Descent

- Batch GD

Uses entire dataset for each update → accurate but slow

- Stochastic GD

Uses one sample per update → fast but noisy

- Mini-Batch GD

Uses Small batches → balance of speed & stability

- Momentum GD

Adds pervious update's momentum → avoids local minima

- Nesterov Accelerated GD (NAG)

looks ahead before updating → faster convergence

- Adagrad

~~Uses~~ Adapts learning rate per parameter → good for sparse data

- RMSProp

Uses moving average of squared gradients → smooth updates

- Adam

Combines Momentum + RMSprop → best for deep learning

Stochastic Gradient Descent

What is SGD?

Stochastic Gradient Descent (SGD) is an optimization algorithm used in machine and deep learning to minimize loss functions efficiently → especially when working with large datasets

Date: __/__/20__ Day: _____

It is faster, scalable version of traditional Gradient Descent

How SGD Works

- In Batch Gradient Descent, gradients are calculated using the entire dataset, which is slow and memory-heavy
- In SGD, gradients are calculated for one sample (or small batch) at a time.
- This makes the training faster but noisier

Update Rule:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x_i, y_i)$$

where:

θ: model parameters (weights & biases)

η: learning rate

$\nabla_\theta J(\theta; x_i, y_i)$: gradient of loss for one data point

(x_i, y_i): i-th training example

∴ Key difference → parameters updated after each sample, not after the whole dataset

Steps to Implement SGD (Linear Regression Example)

- Generate Data
- Model equation = $y = \theta_0 + \theta_1 X$

θ_0 = Intercept (bias) θ_1 = Slope (weight)

- Define SGD Function
- Train the model
- Visualize Cost
- Plot Regression Line
- Interpretation of Final model
  - When $X = 0$, → $y = 4.35$ (Intercept)
  - For each 1 unit increase in X → y increases by 3.45

Date: __/__/20 Day: _____

Applications

- Deep learning (Default optimizing to train Neural N)
- NLP (Used in Word2Vec, Transformer etc)
- Computer Vision (CNNs for Image classification, detection)
- Reinforcement learning (Optimizes policy network & Q-Networks)

Advantages

- Fast and Efficient
- Memory Friendly
- Escapes Local Minima
- Supports Online learning

Challenges

- Noisy Convergence
- Sensitive to learning Rate
- longer Training

∴ Tip: Use adaptive optimizers (Adam, RMSProp) or learning rate scheduling to stabilize training

Batch Normalization

Batch Normalization (BN) is a technique to stabilize and speed up training in deep neural networks
It reduces internal covariate shift - when layer input distributions change during training as earlier layers update
BN makes training faster, more stable, and prevents overfitting

Date: __/__/20 Day: _____

Why Use Batch Normalization?

- Fix Internal Covariate Shift
  - Activations of layers change as parameters update → slow training
  - BN normalizes activations → stable training and faster convergence

- Improves Gradient Flow
  - Keeps gradients stable → helps deeper networks train effectively
  - Reduces vanishing/exploding gradients

- Acts as Regularization
  - Adds small noise to activations → reduces overfitting (like dropout)

How batch Normalization Works?

BN normalizes layer activations across each mini-batch during training

- Normalization Within Mini Batch
  - for each mini-batch

$$\mu_B = \frac{1}{m} \sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

- μ_B = Mean of Batch
- σ_B^2 = Variance of Batch
- ε = Small constant (for stability)

Date: __/__/20 Day: _____

- Scaling and Shifting

After normalization

$$y_i = \gamma \hat{x}_i + \beta$$

- γ : Scale (learned)
- β : Shift (learned)

This lets the model learn the best range for activations

- Learnable Parameters

γ and β are trainable — adjusted by backpropagation

- Where to Apply BN
  - Typically after convolution + before activation in CNNs
  - Can also be used before or after activation depending on ~~arc~~ architecture

- During Training Vs Testing
  - Training: Mean and variance from the current mini-batch
  - Testing: Uses averaged (running) mean and variance from training for consistent normalization

Benefits of Batch Normalization

Faster training	Normalizes activation → faster convergence
Stable Gradients	Prevents vanishing/exploding gradients
Regularization	Adds noise → reduces overfitting
Allow Higher learning Rate	Training become more robust
Works with Deep Networks	Easier to train many layers

Key points to Remember : Normalizes activations per batch.

- Has two learned parameter: γ (scale), β (shift)
- Applied after Conv & before Activation

Date: __/__/20 Day: _____

Mini-Batch Gradient Descent

Gradient Descent is an optimization algorithm used in ML to find parameters (weights & bias) that minimize the loss function

It updates parameters by moving in the direction of the negative gradient (steepest descent)

Types of Gradient Descent

- Batch Gradient Descent
  - Uses the entire dataset to compute gradient each time
- Stochastic Gradient Descent
  - Uses only one sample per update
- Mini-Batch Gradient Descent
  - Uses a small subset (mini-batch) of data per update

Why Mini-Batch Gradient Descent?

MBGD combines the accuracy of BGD and the speed of SGD

It uses mini-batches for updates → faster, more stable, and memory efficient

Key Advantages

- Efficient
- Faster Convergence
- Less Noisy
- Better Generalization
- Memory Friendly

Date: __/__/20__ Day: _____

Algorithm Steps

Let

- θ = model parameters
- η = learning rate
- max_iters = number of epochs
- b = batch size

For each iteration

- Shuffle training data
- Split into mini-batches
- For each batch(X_mini, y_mini):
  - Forward Pass:
    - Predict → y_hat = f(x_mini, θ)
    - Compute loss → $J(\theta)$ = L(y_hat, y_mini)
  - Backward Pass:
    - Compute gradient → $\nabla_\theta J(\theta)$
  - Update Parameters:

$$\theta = \theta - \eta * \nabla_\theta J(\theta)$$

Comparision of Gradient Types

Batch GD	update full dataset Strategy	Slow – Smooth/Stable
SGD	One sample per update	Fast – Noisy
Mini-Batch GD	Small Subset	Fast/efficient → Moderate

Adam

Adam Optimizer (Adaptive Moment Estimation)

Adam is an optimization algorithm that combines the best ideas from Momentum and RMSprop.

It adapts the learning rate for each parameter automatically and works great for large datasets

Date: __/__/20__ Day: _____

and complex deep learning models

How Adam Work?

Adam has 2 Key ideas:

- Momentum
  - Momentum helps speed up learning by using past gradients
  - It smooths out noisy updates and avoids zig-zag ~~momen~~ movement

Formula

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t}$$

$$w_{t+1} = w_t - \alpha m_t$$

where

- m_t : moving average of Past gradients
- α : learning rate
- β_1 : momentum term (usually 0.9)

- RMSprop
  - RMSprop adjusts the learning rate based on how large the gradients are
  - Prevents the learning rate from becoming too small or two large

Formula

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\partial L}{\partial w_t}\right)^2$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t} + \varepsilon}\frac{\partial L}{\partial w_t}$$

Where

- v_t : moving average of squared gradients
- β_2 : decay rate (usually 0.999)
- ϵ : Small value (like 10^{-8}) to avoid division by zero

Date: __/__/20 Day: _____

Adam = Momentum + RMSprop

Adam combines both to make updates more stable and efficient

Equations:

Step 1: First Moment (Mean):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t}$$

Step 2: Second Moment (Variance):

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\partial L}{\partial w_t}\right)^2$$

Step 3: Bias Correction

(to fix the bias at the start)

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Step 4: Weight Update:

$$w_{t+1} = w_t - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Parameters of Adam

Symbol	Meaning	Default Value
α	Learning Rate	0.001
β_1	Momentum decay rate	0.9
β_2	RMSprop decay rate	0.999
ϵ	Small constant to avoid division zero	10^{-8}

Why Adam Works So Well

- Adaptive Learning Rate:
  - Each parameter has its own learning rate → faster convergence.

- Bias Correction:
  - Removes the early-stage bias toward zero → stable updates

Date: __/__/20 Day: _____

- Less Hyperparameter Tuning:
  - Usually works well with default values

- Efficient & Fast:
  - Handles large datasets & parameters efficiently

Comparison With Other Optimizers

Optimizers	Learning Rate	Memory Use	Speed - Stability
SGD	Fixed	Low	Slow + Unstable
Momentum	Fixed	Medium	Fast + Stable
RMSprop	Adaptive	Medium	Fast + good
Adam	Adaptive + momentum	Medium	Fastest + More Stable

Performance:

- Adam usually converges faster and needs less tuning
- It performs better than SGD & momentum in most all tasks
- Excellent for large datasets & non-stationary problems

Momentum-Based Gradient Optimizer

Momentum-Based optimizers improve model training by adding a "momentum" term that remembers past gradients
This helps the optimizer move faster, reduce oscillations and reach the minimum more efficiently - especially in deep neural networks or large datasets.

What is Momentum?
Idea from Physics
Momentum means an object keeps moving based on its previous velocity.
In ML, it helps smooth the optimization path

Date: __/__/20 Day: _____

by remembering past gradient directions

Mathematical Formulas

$$v_{t+1} = \beta v_t + (1-\beta)\nabla L(w_t)$$

$$w_{t+1} = w_t - \eta w_{t+1}$$

Where

- v_t → Velocity (average of past gradient)
- β (beta) → Momentum factor (0-1, usually 0.9)
- $\nabla L(w_t)$ → Gradient of loss
- η (eta) → Learning rate

Hyperparameters

- Learning Rate (η):
  - Controls how big each update step is.

- Momentum Factor (β):
  - Controls how much past gradients affect the current step
  - β close to 1 → More inertia (smoother path)
  - β close to 0 → Less memory of past gradients

How It Works

- Velocity Update
  - Combines previous velocity & current gradient
  - β controls how much of old velocity we keep
- Weight Update
  - Updates weights using the velocity (average gradient)

Types of Momentum - Based Optimizers

- Nesterov Accelerated Gradient (NAG)
  - Calculates gradient at the future position

Date: __/__/20 Day: _____

- Formula

$$v_{t+1} = \beta v_t + \nabla L(w_t - \eta \beta v_t)$$

$$w_{t+1} = w_t - \eta v_{t+1}$$

- Benefit: Faster & more accurate than classical momentum

- Adamomentum
  - Combines adaptive learning rate + momentum
  - Adjusts momentum based on recent gradients
  - More sensitive to shape of loss function

- RMSprop
  - Adapts the learning rate for each parameter.
  - Acts like momentum by averaging squared gradients
  - Works great with non-stationary data (e.g RNNs)

Advantages
- Faster Convergence
- Less Oscillation
- Better generalization
- Escapes Local minima

Challenges
- Hyperparameter Tuning:

  Learning rate (η) and momentum (β) need careful selection. Common: $\beta = 0.9$
- Over-accumulation:

  Too much momentum may overshoot the minimum
- Initialization

  Poor initial momentum can slow convergence

Date: __/__/20 Day: _____

Adagrad Optimizer

Adagrad adjusts the learning rate for each parameters during training using the past squared gradients.

This helps when features vary in frequency or scale - great for sparse data like text or recommender systems

Main IDea

Instead of one fixed learning rate (like standard gradient descent)

Adagrad gives each parameter its own rate - smaller for frequent features, larger for rare ones

How Adagrad Works

- Initialization
  - Randomly initialize parameters (θ)
  - Initialize $G_t = 0$, a running sum of squared gradients
- Compute gradient
  - Calculate gradient $\nabla_\theta J(\theta)$ for each parameter (same as normal GD)
- Adaptive Learning Rate
  - The adaptive rate:

$$lr_t = \frac{\eta}{\sqrt{G_t + \varepsilon}}$$

Where:

- η = global learning rate (small constant, e.g 0.01)
- G_t = sum of squared gradients up to step t
- ε = small value (1e-8) to avoid division by zero
- Parameter Update

$$\theta_{t+1} = \theta_t - lr_t \cdot \nabla_\theta J(\theta)$$

Date: __/__/20__ Day: _____

When to Use Adagrad

Use For:

- Sparse Data (NLP, recommender systems)
- Different feature importance or frequencies
- Stable training (not very fast but reliable)

Avoid if:

- You need a constant learning rate → use RMSprop or Adam instead

Variants of Adagard

- RMSProp (Root Mean Square Propagation)

Fixes adagard's issue of learning rate shrinking too much

Uses an exponential moving average of squared gradients

$$G_t = \gamma G_{t-1} + (1-\gamma)(\nabla\theta J(\theta))^2$$

parameter update:

$$\theta_{t+1} = \theta - \frac{\eta}{\sqrt{G_t+\varepsilon}} \nabla\theta J(\theta)$$

Where

- γ = decay factor (≈ 0.9)

keeps learning rate steady → better for deep networks

- Ada Delta

Also fixes decreasing learning rate

Uses moving average of both gradients and parameters updates

$$\Delta\theta_{t+1} = -\frac{\sqrt{E[\Delta\theta^2]_t + \epsilon}}{\sqrt{E[\nabla\theta J(\theta)^2]_t + \epsilon}} \nabla\theta J(\theta)$$

More stable and bounded updates

Date: __/__/20 Day: _____

- Adam (Adaptive Moment Estimation)

Combines Adagard + Momentum

Uses averages of gradients (m_t) and squared gradients (v_t)

Formulas:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla\theta J(\theta)$$

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

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\varepsilon}\hat{m}_t$$

Robust, widely used, and fast to converge

Advantages

learns different rates per parameter → better for sparse data

Handles noisy gradients & rare features well

Auto-tunes learning rate → less manual tuning

Efficient when gradient magnitudes vary a lot

Limitations

learning rates keep shrinking, slowing training

Sensitive to initial η (learning rate) choice

No momentum → harder to escape local minima

Can stop learning early due to too-small rates

RMSprop Optimizer

RMSProp (Root Mean Square Propagation) is an adaptive learning rate optimization algorithm used to train deep learning models faster and more efficiently

It is a modified form of Gradient Descent, where

Date: __/__/20__ Day: _____

the learning rate is adjusted automatically for each parameter based on recent gradient values. This helps when working with non-stationary objectives or sparse gradients

Why RMSprop is Needed?

- SGD uses one fixed learning rate → can be slow or unstable
- AdaGrad reduces learning rate too fast → stops learning early
- RMSprop fixes this by keeping a moving average of squard gradients, balancing speed & stability during training

∴ Hence, RMSprop is widely used in modern DL.

How RMSProp Works?

It uses a moving average of squard gradients to normalize parameter updates, preventing very small learning rates (a problem in AdaGrad)

Steps:

- Compute gradient
  - $g_t = \nabla \theta_t$
- Update moving average of squard gradients
  - $E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$
- Update parameters:
  - $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} g_t$

where

- η (eta) = learning rate
- ε (epsilon) = small value to prevent division by zero

Date: __/__/20 Day: _____

RMSProp Parameters

Parameter	Meaning	Typical Value
η (learning Rate)	Step size for updates	0.001
γ (Decay Rate)	Controls gradient memory	0.9
ε (Epsilon)	Prevents division by zero	$1e^{-8}$

These values can be adjusted depending on the dataset & model

Advantages of RMSProp

- Adaptive Learning Rate - Each parameter gets its own rate
- Handles Non-Stationary Objectives - Good for changing data
- Avoids Fast Decay - Unlike AdaGrad
- Faster Convergence - learns efficiently

Disadvantages

- Sensitive to Hyperparameters
- Not Great for Sparse Data

Types of Deep learning Models

- Convolutional Neural Network (CNNs)
- Recurrent Neural Network (RNNs).
- GEnerative Models in Deep learning
- Deep Reinforcement learning (DRL)

Date: __/__/20__ Day: _____

Convolutional Neural Networks

• Deep learning Algorithms

Top 10 Deep learning Algorithms

What is Deep Learning?

- A subfield of ML under Artifical Intelligence (AI)
- Uses multi-layer neural networks to learn from data automatically
- Inspired by the human brain
- Best for images, speech, and text tasks
- Improves with more data and computing power

Quick History

- 1940s – Concept Introduced
- 1950s – 'Perceptron' invented (early neural model)
- Progress accelerated with big data, GPUs & better algo.

What are Deep Learning Algorithms?

- Special ML models based on artifical neural networks
- Learn complex patterns automatically – no manual feature extraction
- Used for image Classification, speech recognition, NLP.

Top 10 DL Algorithms (2025)

- Convolutional Neural Networks (CNNs)

Used for: Images, vedio, object dectection, facial recognition

Key Mechanisms:

- Convolution Layer: Extracts basic patterns (edges, shapes)
- Pooling Layer: Reduces data size, keeps key features
- Fully Connected layer: Makes final prediction
- Activation Function: Adds non-linearity to learn complex data

Date: __/__/20__ Day: _____

- Recurrent Neural Network (RNN)

Used for: Sequential / time data - speech, text, stock prediction

Key Mechanisms:

- Sequential Processing: Handles data step-by-step
- Hidden States: Remembers previous information
- Weight Sharing: Same weights across all time steps
- BPTT: Learns from future steps to improve predictions

- Long Short-Term Memory (LSTM)

Used for: long sequences, text generation, video analysis

Key Mechanisms:

- Cell State: Keeps log-term memory
- Forget gate: Removes unneeded info
- Input gates: Adds new useful info
- Output gate: Sends filtered info forward

- Autoencoders

Used for: Data Compression, denoising, anomally detection

Key Mechanisms:

- Encoder: Compresses input data
- Bottleneck: Small representation capturing key info
- Decoder: Reconstructs data back
- Loss Function: Measures difference (e.g MSE)

- Deep Belief Network (DBN)

Used for: Feature Extraction, pretraining, Image recognition

Key Mechanisms:

- Stacked RBMs: Multiple layer of Restricted Boltzmann Machines
- Unsupervised Pretraining: learns features without labels
- Fine-tuning: Adjusts with labeled data
- Stochastic Units: Learn nonlinear relationships using probabilities

Date: __/__/20__ Day: _____

- Generative Adversarial Network (GAN)

Used for: Image/video, deepfakes, data augmentation

Key Mechanisms

- Generator: Creates fake data
- Discriminator: Detects fake vs real data
- Adversarial Training: Both compute to improve
- Loss Function: Measures difference to train both models

- Self-Organizing MAP (SOM)

Used for: Clustering, visualization, dimensionally reduction

Key Mechanisms

- Neuron Grid: Represents clusters of data
- Competitive learning: Neurons "Compute" to match data
- Neighborhood Function: Neighbors also learn similar info
- Topology Preservation: Keeps similar data close together

- Variational Autoencoders (VAEs)

Used for: Generating new images or data samples

Key Mechanisms:

- Encoder: Learns mean and variance of data distribution
- Latent Space: Samples points from learned distribution
- Decoder: Reconstructs data from sampled points
- KL Divergence: Keeps learned distribution close to normal

- Graph Neural Network (GNN)

Used for: Social networks, molecules, recommendation system

Key Mechanisms

- Node Aggregation: Combines neighbor information
- Message Passing: Sends info b/w nodes
- Graph Pooling: Creates overall graph representation
- Backpropagation: Optimizes node features for predictions

Date: __/__/20 Day: _____

- Transformers:

Used for: NLP, translation, chatbots, summarization

Key Mechanisms:

- Self-Attention: Each word attends to other for context
- Multi-head Attention: Multiple relationships learned at once
- Positional Encoding: Adds order info since model isn't sequential
- Feedforward Layers: Process attention outputs for predictions

Introduction to Convolutional Neural Network

CNN (Convolutional Neural Network) is an advanced type of Artificial Neural Network (ANN) mainly used to extract patterns and features from grid-like data such as images & videos

It is widely used in computer vision tasks

Best Structure of CNN

A CNN has several layers:

- Input layer - takes image input (e.g, 32×32×3 for RGB images)
- Convolutional layer - extracts features using filters/Kernals (small matrics like 3×3 or 5×5)
- Activation layer - adds non-linearity using functions like ReLU, Tanh, or leaky ReLU.
- Pooling Layer - reduces size and computation using Max Pooling or Average Pooling
- Flattening Layer - Converts feature maps into a 1D vector
- Fully Connected layer - performs classification or regression
- Output layer - gives final predictions using softmax or sigmoid.

Date: __/__/20__ Day: _____

How Convolution Works

- Images can be seen as 3D matrices (width × height × channels like RGB)
- A filter/Kernal slides (called stride) across the image and performs dot products with parts of the image
- The result forms a feature map showing important image features (edges, shapes, etc)
- Using small filters reduces weights and speeds up learning
- If the patch covers the whole image → it acts like a normal neural network.

Mathematics

For image size $34 \times 34 \times 3$, filters may be $3 \times 3 \times 3$ or $5 \times 5 \times 3$

Each step (stride) moves the filter and forms new output depth equal to the number of filters.

Example: Applying CNN to an Image

Steps

- Import libraries (Numpy, Tensorflow, Matplotlib)
- Load image & resize.
- Define Kernel (eig, edge detection filter)
- Apply:
  - Convolution layer → detect edges/patterns
  - Activation (ReLU) → Keeps positive values
  - Pooling (Max Pool) → Reduce dimensions

Result:

- Original → Convolution → Activation → Pooling
- Shows feature extraction process visually

Date: __/__/20 Day: _____

- Advantages
- Excellent for detecting patterns in images/videos.
- Works well even if image is shifted, rotated or scaled.
- Learns features automatically (no manual extraction)
- High accuracy with large datasets.

Disadvantages
- Requires high computational power & memory
- Can overfit if not enough data or regularization
- Needs large labeled datasets
- Difficult to interpret (hard to know what each neuron learns)

Digital Image Processing (DIP)

DIP means using computer algorithms to process and improves digital images
- The goal is to enhance image quality, extract useful information, and automate image-related tasks

Basic Steps In DIP
- Image Acquistion: Capture or import image using camera or scanner.
- Image Enhancement: Improve visibility - Increase contrast, reduce noise
- Image Restoration: Remove blurring, noise, & distortion
- Image Segmentation: Divide image into regions/objects
- Image Representation & Description: Represent and describe image features for analysis
- Image Analysis: Extract information - detect patterns, recognize objects.

Date: __/__/20 Day: _____

- Image Synthesis & Compression: Generate new images or reduce file size

Also Include:

Import Image → Analyze/manipulate → output

What is an Image?

- Image = 2D Function $F(x, y)$ → 'x' and 'y' and coordinates; $F(x, y)$ = intensity
- When values of x, y and F are finite → Digital Image
- Digit Image = grid of small elements called pixels

Type of Images

Type	Description
Binary	Only 0 (Blacks) & 1 (white) called Monochrome
Black & white	Only two colors → Black & white
8-Bit (Grayscale)	256 shades (0 = Black, 255 = white, 127 = Gray)
16-Bit (High color)	65,536 colors → divided into Red, Green Blue (RGB) Channels

Image As a Matrix

- Image = 2D array (rows × columns)
- Each element (pixel) stores Intensity / color value.
- Example representation

$$f(x, y) = \text{Pixel value at location } (x, y)$$

In MATLAB:

- Index starts at 1 (not 0)
- Image stored in a variable (e.g; x, input_image)
- Example: $f(1, 1)$ in MATLAB = $f(0, 0)$ in other system

Date: __/__/20 Day: _____

Phases of Image Processing

Phase	Description
Acquisition	Capture Image, scaling, color
Enhancement	Improve image visibility
Restoration	Fix degradation
Color Processing	Handle color models
Wavelets & Multi-Resolution	Represent Image at diff level
Compression	Reduce size for storage
Morphological Processing	Extract shapes & structures
Segmentation	Partition image into objects/part
Representation & Description	Transform raw data into features
Object Detection & Recognition	label objects based on features

Overlapping Fields

Input → Output	Field
Image → Image	Digital Image Processing
Image → Info	Computer Vision
Code → Image	Computer Graphics
Code → Code	Artifical Intelligence

Advantages of DIP

- Enhances image clarity and detail
- Automates tasks like object dectection & pattern recognition
- Fast and Efficient → handles large data quickly
- More accurate than manual analysis

Disadvantages of DIP

- High Computational cost
- Some algorithms are hard to interpret
- Output quality depends on input image quality
- Struggle with poor lightening or clutter scenes.

Date: __/__/20 Day: _____

Importance of CNNs

CNNs are special neural networks that use convolution operations to work with visual data like images and videos for classification & prediction

Weight Sharing & Local Spatial Coherence

- CNNs use shared weights (kernels) that move across the image
- Same detector works for all positions → fewer parameters
- Benefits:
  - Less Computation
  - Faster training
  - Works well even low-power devices

Example

Instead of learning separate features for every pixels, CNNs reuse the same filters everywhere

Memory Efficiency

- Because of shared weights, CNNs use much fewer parameters than fully connected networks

Example

- CNN on MNIST → few hundred parameters
- Fully Connected NN → ~19,000 parameters

- Benefits
  - Needs less memory
  - Reduces overfitting risk
  - Works well on small devices

Robustness to Local Variations

- CNNs extract features from different image parts
- They can still detect objects even if position changes slightly

Date: __/__/20 Day: _____

- Example:
  - CNN can recognize faces in full-body photos
  - Fully connected NN may fail if image view changes

Benefits

- Handles small shifts and variations well

Equivariance to Transformations

- Equivariance: when the input moves, the feature map moves too
- In CNNs

$$f(g(x)) = g(f(x))$$

where f = convolution, g = translation

Benefits

- CNN can detect the same object even if moves in image
- Reliable & Consistent predictions

- Invariance to Image Etas Transformations
- CNNs are partly invariant to translation, rotation & scaling
- They can recognize objects in different positions, angeles or sizes

Example

- Translation: Detects an object even if it's shifted
- Rotation: Still identifies it if rotated

Benefits

- Works well for real-world images where position or angle changes.

Date: __/__/20 Day: _____

Padding

Why padding is needed?

When we apply a Kernel (filter) on an image during convolution, the output size becomes smaller than the input.

This causes loss of information at the borders

To solve this, we use Padding - adding extra pixels (usually zeros) around the border of the image

What is padding?

Padding = Adding extra pixels around the border of the input image before convolution.

This helps to preserve image size and border information

Two Types:

- Valid Padding - No padding is added
  - Output size < Input Size
  - Used when we want to reduce the feature map size
- Same Padding - Padding is added so output size = input size
  - Used when we want to perserve the spatial size

Common padding type: Zero-padding (adds zeros around the image)

Benefits:

- Keeps edge information
- Helps build deeper CNNs
- Slightly increase computation

Date: __/__/20 Day: _____

Problem Without Padding

If Image size = $(n \times n)$ and filter = $(f \times f)$

$$\text{Output Size} = (n - f + 1) \times (n - f + 1)$$

Example:

8×8 Image with 3×3 filter → output = 6×6

- Each convolution reduces the size - image shrinks after many layers

→ Borders used less than center pixels → border info lost

Example:

- Pixel A (corner): used once
- Pixel B (edge): used 3 times
- Pixel C (center): used 9 times
  - Center pixels are used more → border info fades

Effects of Padding

Padding adds layers of zeros → images becomes bigger before convolution

If p = padding size, then

$$(n \times n) \rightarrow (n + 2p) \times (n + 2p)$$

Output Size =

$$(n + 2p - f + 1) \times (n + 2p - f + 1)$$

Example:

8×8 image + 1 layer of padding + 3×3 filter → output = 8×8 (same as input)

- Border pixels are now used more often → no info loss

Types of Padding

Type	Formula	Notes
Valid	$(n \times n) * (f \times f) \rightarrow (n - f + 1) \times (n - f + 1)$	No padding added
Same	$(n + 2p) \times (n + 2p) * (f \times f) \rightarrow (n \times n)$	Keeps same output size

To keep same size:

$$p = \frac{(f - 1)}{2}$$

Date: __/__/20 Day: _____

Convolution Layer in CNN

What are Convolution layers?

Convolution layers are the main part of CNNs used for image and video processing

They use filters (Kernals) that slide over the input image, multiply values, and sum them to create a feature map.

- This helps detect edges, textures, shapes, & patterns in data

Key Components

- Filters (Kernals)
  - Small matrices that extract features
  - Example: one filter detects horizontal edges, another detects vertical edges
  - Values are learned during training

- Stride
  - Step size of how far the filter moves each time
  - Larger Stride → Smaller output & faster computation.

- Padding
  - Adds zeros around the image to control output size.
  - Types
    - Valid: no padding → Smaller output
    - Same: adds padding → Same output size as input

- Activation Function
  - Adds non-linearity after Convolution
  - Common: ReLU, Tanh, Leaky, ReLU.

Date: __/__/20 Day: _____

Types of Convolution layers

Type	Description
2D Convolution (conv2D)	For image data (filters move in height & width)
Depthwise Separable	Breaks convolution into smaller parts
Dilated (Atrous)	Adds spaces b/w kernel elements

Steps In a Convolution Layer

- Initialize Filters with random weights
- Slide Filters over Input → Compute dot product
- Apply Activation Function to add non-linearity
- Pooling to reduce size & keep key features

Example

Input: $32 \times 32 \times 3$ image (RGB)

Layer: 10 Filters (5×5), Stride = 1, padding = 'Same'

Output : $32 \times 32 \times 10$ feature map

Each filter learns a different pattern or feature

Applications

- Medical Imaging
- Self-driving Cars
- NLP/ Speech
- Industry

Convolutional vs Fully Connected layers

Aspect	Convolutional	Fully Connected
Connectively	Local	Global
Parameters	Fewer (shared weights)	Many
Spatial info	Preserved	Lost
Use	Feature extraction	Classification / Regression

Date: __/__/20__ Day: _____

Benefits

- Parameter Sharing
- Local Connectively
- Hierarchical learning
- Efficiency

Limitations

- High Resource Needs
- Large Data Requirement
- Limited Global Context
- Overfitting Risk

Pooling Layer

A pooling layer in CNN is used to reduce the size (height & width) of the feature maps while keeping the key information

It slides a filter (window) over the feature map and summarizes the value (like max or avgerage)

If the feature map size = $(n_h \times n_w \times n_c)$

After pooling → Output size =

$$\left(\frac{n_h - f}{s} + 1\right) \times \left(\frac{n_w - f}{s} + 1\right) \times n_c$$

where

- n_h, n_w: height and width of feature map
- n_c: No. of channels
- f: filter size
- s: stride

Why Pooling layers are Important?

- Dimensionally Reduction
  - Makes model smaller & faster by reducing feature map size

Date: __/__/20 Day: _____

- Translation Invariance
  - Small shifts in input don't affect output (object can move slightly, still detected)
- Prevents Overfitting
  - Fewer parameters = less chance of overfitting
- Feature Hierarchy:
  - Helps build a multi-level understanding (edge → shapes → objects)

Types of Pooling layers

- Max Pooling
  - Takes the maximum value in each region
  - Keeps the most importance features (like edges)
  - Most Commonly used

Example Output

Input 4×4 → 2×2 output using 2×2 filter
→ Picks max value from each patch.

- Average Pooling
  - Takes the average value from each region
  - Keeps overall structure instead of just strong feature
  - Useful when context is more important than fine details

- Global Pooling
  - Converts each channel into a single value (1×1×nc)
  - Used near the end of CNNs before fully connected layers

Types

- Global Max Pooling: Max of entire feature map
- Global Average Pooling: Average of entire feature map

Date: __/__/20__ Day: _____

How Pooling Works?

- Choose filter/window size (e.g 2x2) & stride (e.g 2)
- Slide the filter over the feature map
- Apply pooling operation (max or average)
- Output smaller feature map → less computation

Key Factors to Optimize Pooling

Factor	Description
Window Size	Larger window = more downsampling
Stride	Bigger stride = smaller output & fast training
Padding	Ensures full coverage of input, sometimes used

Advantages

- Reduces Dimensions
- Adds Translation Invariance
- Feature Selection

Disadvantages

- Information Loss
- Over-Smoothing
- Tuning Needed

Fully Connected Layer

- It is called a Dense layer
- Every neuron is connected to all neurons in the previous and next layers
- Found in Deep learning networks, especially after CNN layers, to interpret extracted features and make final predictions
- In feedforward networks, FC layers are the main layers that map inputs → Outputs

Date: __/__/20 Day: _____

Structure of FC Layer

Each neuron in one layer connects to every neuron in the next layer.

Key Elements

- Neurons: Basic units that receive & send information
- Weights: Define the strength of connections b/w neurons
- Bias: Added to help adjust output
- Activation Function: Adds non-linearity (ReLU, Sigmoid, Tanh)

Working of Fully Connected layer

- Input processing

$$z_j = \sum_i (w_{ij} \cdot x_i) + b_j$$

- Each neuron gets inputs from all previous neurons
- Multiplies them by weights & adds bias

- Activation:

$$a_j = f(z_j)$$

- Passes the result through an activation function to learn non-linear patterns.

Example:

If a layer has 4 neurons and the next FC Layer has 3 neurons → total = 12 weights + 3 biases

Key neuron is the new layer connects to all neurons in the previous one

Key Role of FC Layers

- Feature Integration
  - Combine all extracted feature (from CNN/RNN)

Date: __/__/20 Day: _____

- Decision Making
  - Used as final ~~year~~ layer for classification or regression
- Non-linearity:
  - Activations helps learn complex patterns
- Universal Approximation:
  - With enough neurons, FC layer can learn any continuous function
- Flexibility:
  - Works for all domains— Vision, Speech, NLP
- Regularization
  - Dropout/L2 reduce overfitting & improve generalization

Advantages
- Integrates features
- Flexible
- Simple

Disadvantages
- High-Computation
- Overfitting
- Loses Spatial Info

Backpropagation in CNNs
- Backpropagation: Backward Propagation of Errors
- It calculates how much each weight in the network contributed to the error (Loss)
- Helps update weights to reduce loss and improves accuracy
- In CNNs, it- fine-tunes filters, biases, & weights across layers

Date: __/__/20 Day: _____

- Why Backpropagation is important in CNNs?
- Adjusts filters and weights in convolutional, pooling, and fully connected layers
- Enables CNNs to learn useful features & differentiate objects

Basic Ideas

It uses calculus (chain rule) to find how the loss function (L) changes with each weight (w)
Three main steps:

- Forward Pass
- Loss Calculation
- Backward Pass

Foward Pass

Data moves forward through the network to produce an output.

- Convolutional Layer:

$$O = (I * F) + b$$

  - I: Input
  - F: Filter
  - b: Bias
  - O: Output feature map
- Activation Function:

$$O = \sigma((I * F) + b)$$

  - Applies ReLU or another activation for non-linearity
- Pooling layer:
  - Reduces dimensions & highlights important features

Date: __/__/20 Day: _____

Loss Calculation

Measures how far the prediction is from the actual output

For Classification

$$L = -\sum y \log(\hat{y})$$

- y : true label
- $\hat{y}$: predicted label

The goal: Minimize L (Loss)

Backward ~~Bros~~ Pass (Backpropagation)

Backpropagation moves errors backward through the network to update weights

- Gradient of Loss w.r.t Output

$$\frac{\partial L}{\partial O}$$

- Finds how loss changes with output

- Gradient through Activation

$$\frac{\partial L}{\partial I} = \frac{\partial L}{\partial O} \cdot \frac{\partial O}{\partial I}$$

- For ReLU : gradient = 1 (if input > 0), else 0

- Gradient w.r.t Filters

$$\frac{\partial L}{\partial F} = \frac{\partial L}{\partial O} * \text{rot}180(I)$$

- Input rotated 180° for convolution alignment

Weight Update Rule

Weights are adjusted using gradient descent

$$F_{new} = F_{old} - \eta \frac{\partial L}{\partial F}$$

- η = learning rate (step size)

Date: __/__/20 Day: _____

Challenges In Backpropagation

- Vanishing Gradients
  - Gradients become very small → training slow/stop
  - Solution: ReLU, Batch Normalization
- Exploding Gradients
  - Gradients become too large → unstable training
  - Solution: Gradient Clipping

CNN Based Architectures

- Convolutional Neural Network (CNN) Architectures
- LeNet - 5
- AlexNet
- VGGnet
- VGG-16 Network
- GoogLeNet/Inception
- ResNet (Residental Network)
- MobileNet

CNN Architectures

CNN is a deep learning model used to recognize patterns in structured data like images

- LeNet - 5 (1998)
- Alex Net (2012)
- GoogleNet (2014)
- ResNet (2015)
- DenseNet (2017)

We will discuss them more deeply one by one.

Date: __/__/20__ Day: _____

LeNet-5 Architecture

Created by: Yann LeCun, Leon Bottou, Yoshua Bengio
Patrick Haffner (1998)

Purpose: Recognize handwritten & printed characters

Importance: First successful CNN - introduced convolution
pooling, hierarchical features

Applications

- Handwriting recognition
- Object/ face recognition
- Traffic sign detection

Architecture Summary

- Total layers: 7 (excluding input)
- Input Size: 32 x 32 pixels
- Activation: tanh (original), sometimes ReLU in modern versions
- Pooling: Average pooling
- Output: 10 classes (digits 0-9)

layer-By-layer Breakdown

- Input layer
  - Input: 32 x 32 grayscale image
  - Centered in 28 x 28 field (digits upto 20x20)
  - Normalization: Background → 0, Foreground → 1
    → Speeds up training (mean ≈ 0, Variance ≈ 1)

- C1 - Convolution layer
  - Feature Maps: 6
  - Kernel: 5 x 5
  - Output: 28 x 28
  - Parameters: 156
  - Connections: 117,600
  - Each filter extracts basic features (edges, curves)

Date: __/__/20__ Day: _____

- S2 – Subsampling (Pooling) layer
  - Feature Maps: 6
  - Pooling Size: 2 × 2
  - Output: 14 × 14
  - Parameters: 12
  - Connections: 5,880
  - Performs average pooling → reduces size & noise
  - Uses trainable coefficient + bias → adds flexibility

- C3 – Convolution layer
  - Feature Maps: 16
  - Kernel: 5 × 5
  - Output: 10 × 10
  - Parameters: 1,516
  - Connections: 151,600
  - Partially connected to S2 → learns diff. feature combinations

- S4 → Subsampling layer
  - Feature Maps: 16
  - Pooling Size: 2 × 2
  - Output: 5 × 5
  - Parameters: 32
  - Connections: 2,744
  - Further reduces feature map size → controls overfitting

- C5 → Convolution layer (Fully Connected Type)
  - Feature Maps: 120
  - Kernel: 5 × 5
  - Output: 1 × 1
  - Parameters: 48,000
  - Each neuron connected to all 16 maps of S4 → acts as a fully connected conv layer.

Date: __/__/20 Day: _____

- F6 → Fully Connected layer
  - Units: 84
  - Connection: 10,164
  - Activation: Scaled tanh
    - $f(a) = 1.7159 \times \tanh(2a/3)$
  - Combines learned features for final classification

- Output layer
  - Units: 10 (for 10 digits)
  - Activation: Euclidean RBF (Radial Basis Function)
  - Computation:
    - $y_i = \sum_j x_j \times w_{ij}$
  - Measures Euclidean distance b/w input & class model
  - larger distance → higher penalty (less likely class)

Key Layer Roles

Layer Type	Function
Convolution (Cx)	Extract features using filters
Subsampling (Sx)	Reduce size (average pooling)
Fully Connected (Fx)	Combine learned features for classification

AlexNet

Created: By Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton

Famous For: Winning ImageNet 2012 (Top-5 error 15.3% vs 26.2% before)

Use: Image Classification, object detection, & feature extraction

Main Features

Date: __/__/20 Day: _____

Main Features

- Overfitting Prevention
  - Used Dropout (0.5) on first 2 fully connected layers
  - Used Data Augmentation (flip, Crop, etc) to increase dataset
- Faster training
  - Used ReLU activation (faster than sigmoid / tanh, avoids vanishing gradient)

- Architecture Overview
  - Input: 224 x 224 RGB image (for imageNet), modified to 32 x 32 for CIFAR-10
  - Layers
    - 5 Convolution layers → Feature extraction
    - Max-Pooling after 1st, 2nd, & 5th layers
    - 2 Fully Connected (FC) Layers → Each uses Dropout
    - 1 Softmax layer → Final classification

Overlapping Max Pooling

- Filter: 3x3, Stride: 2 → Better accuracy (reduce errors slightly)

Advantages	Disadvantages
ReLU Activation	large Model (≈ 60 M params)
Dropout	High Computation Cost
GPU Parallelism	Manually Designed
Overlapping Pooling	Overfits on Small Datasets
High Accuracy.	Old design.

Applications

- Image & Object Classification
- Feature Extraction
- Object Detection

Date: __/__/20 Day: _____

VGG-Net Architecture

VGG models (VGG-16 & VGG-19) were created by Karen Simonyan and Andrew Zisserman from the University of Oxford's Visual Geometry Group (VGG)
They became famous in the 2014 ImageNet Challenge (ILSVRC) for their deep, simple, and Uniform CNN structure
VGG-19 is the deeper version with 19 layers (16 conv + 3 fully connected)
It's known for simplicity, consistency, and great performance

Evolution of VGG Models

Befor VGG:

- LeNet-5 (1990s): Recognized handwritten digits
- AlexNet (2012): Introduced deep CNNs & GPU training

Then came VGG (2014)

- Introduced in paper: "Very Deep Convolutional Network for large-Scale Image Recognition".
- Goal: Study how increasing depth affects performance
- VGG-16 & VGG-19 had 16 & 19 weight layers
- Used small 3 x 3 filters everywhere → Simpler, uniform

VGG-19 Architecture

- Total layer: 19 (16 convolution + 3 fully connected)
- Paltern: Simple and repetitive blocks
- Input Image Size: 224 x 224 RGB

Key Components

- Convolutional layers
  - 3x3 filters, Stride = 1, padding = 1
  - Perserve Image Size

Date: __/__/20__ Day: _____

- Activation function
  - ReLU after each conv layer for non-linearity
- Pooling layers
  - Max Pooling (2×2 filter, stride = 2) to reduce spatial size
- Fully Connected Layers
  - 3 layers at the end for classification
- Softmax Layer
  - Gives final class probabilities

Architecture Design Principles

- Uniform 3×3 Filters:
  - Same size throughout the network → Clean, simple Design
- Depth:
  - More layers → better ability to learn complex features
- ReLU Activation
  - Adds non-linearity and speeds up training
- Max Pooling
  - Reduces dimensions, keeps main features
- Fully Connected layers
  - Combine all learned features for final classification

Impact and legacy of VGG-19

Influence on New Models

- Inspired later models like ResNet and Inception
- Proved that depth + simplicity improves accuracy

Transfer Learning

- Widely used for feature extraction and fine-tuning on other datasets
- Used in object detection, image segmentation, and style transfer

Date: __/__/20 Day: _____

Research & Industry

- Used ~~as~~ as a baseline model in research papers
- In industry applied in medical imaging, autonomous ~~vech~~ vehicles, etc

Summary

- Simplicity + Power
- High Computational cost
- Excellent Feature Extractor
- Data Augmentation
- Influence

VGG-16 Networks

VGG-16 is a Convolutional Neural Network (CNN) develop by the Visual Geometry Group (VGG) at Oxford University

It has 16 layers — 13 Convolutional + 3 fully connected layers

Famous for its simplicity, depth, & high accuracy

Used for image classification, object detection, and Image segmentation

Achievement 92.7% test accuracy on the ImageNet dataset (14M images, 1000 classes)

Still widely used due to its versatility & strong performance.

Model Objective

- Input image size: 224 × 224 × 3 (RGB)
- The model outputs a vector of 1000 probabilities one for each class:

$\hat{y} = [\hat{y}_0, \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{999}]$

- Each value shows how likely the image belongs to that class.

Date: __/__/20 Day: _____

Softmax Function

Used to convert raw outputs (logits) into the probabilities

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

- Ensures all probabilities add up to 1
- The top-5 predictions are taken as the most likely classes

Example:

If predicted classes = [780, 0, 1, 2, 999] and ground truth = [780, 2, ~~88~~ 999], then loss = 0, because all ground truth classes are in top-5

VGG-16 Architecture

- Deep CNN with simple and uniform design
- 16 total layers → 13 Conv + 3 Fully Connected
- Layers grouped into blocks (each with conv + MaxPooling)
- Uses ReLU activation after every Conv & FC Layer

Limitations

- Slow to train
- Very large file size
- High Memory Use
- Exploding gradient risk

GoogLeNet / Inception

GoogLNet (also called Inception V1) is a deep CNN for Image classification

Designed to be fast, accurate, and computationally efficient

Introduced the Inception module; which runs 1x1,

Date: __/__/20 Day: _____

3×3, 5×5, convolutions, and max pooling in parallel
Uses 1×1 convolutions, global average pooling, and auxiliary classifiers to improve speed & depth
Suitable for large-scale visual recognition tasks
won the ILSVRC 2014 competition in classification & detection

Key Features of GoogleNet
GoogleNet improved over older folder CNNs (like AlexNet, ZFNet) using 5 main innovations

- 1×1 Convolutions (Dimensionality Reduction)
  - Purpose: Reduce the number of parameters while keeping important information
  - Works as a bottleneck layer to make the network deeper & faster.

Example
- Without 1×1 Conv:
  → $(14 \times 14 \times 48) \times (5 \times 5 \times 480) = 112.9M$ operations
- With 1×1 Conv:
  → $(14 \times 14 \times 16) \times (1 \times 1 \times 480) + (14 \times 14 \times 48) \times (5 \times 5 \times 16) = 5.3$ operation
  - Result: Same performance, but much less computation

Global Average Pooling (GAP)
- Replaces fully connected layers used in old models like AlexNet
- GAP takes the average value of each feature map (7×7 → 1×1)
- Reduces parameters and prevents overfitting

Date: __/__/20__ Day: _____

Benefits of GAP:

- No extra trainable parameters
- Less overfitting
- Improves top-1 accuracy by ~0.6%

- Inception Module (Main Innovation)
  - The heart of GoogLeNet
  - Runs multiple operations in parallel
    - 1×1 convolution
    - 3×3 convolution
    - 5×5 convolution
    - 3×3 max pooling
  - Then concatenates their outputs along the depth

Purpose: Capture features at multiple scale (small + large)
Advantage: Strong feature extraction without heavy computation.

- Auxillary Classifiers (Help During Training)
  - Solve the vanishing gradient problem in deep networks
  - Two extra classifiers added in middle layers to stabilize training
  - They are used only during training, not during inference

Each Auxiliary Classifier has:

- Average Pooling (5×5, Stride 3)
- 1×1 conv (128 filters, ReLU)
- Fully connected (1024 units, ReLU)
- Dropout (rate = 0.7)
- Fully Connected Softmax (1000 classes)

Date: __/__/20__ Day: _____

- Model Architecture Overview
  - 22 layer deep (excluding pooling layers)
  - Focuses on speed + efficiency so it can run on limited hardware

ResNet (Residual Networks)

What is ResNet?

ResNet was proposed in 2015 by Microsoft research
It solved the vanishing/exploding gradient problem in deep CNNs
Earlier networks (like AlexNet, VGG) improved by adding more layers, but after a limit, accuracy decreased due to gradient problems
ResNet allowed training very deep networks (100–1000 layers) efficiently

The Problem Before ResNet

- As CNNs get deeper, gradients either vanish (become 0) or explode (too large)
- This increases training & testing errors instead of reducing them

Example:

- A 56-layer CNN had more error than a 20-layer CNN → caused by gradient issues

ResNet Solution: Residual Blocks

- Introduced the concept of Residual learning using Skip (shortcut) Connections
- Instead of learning the output directly ($H(x)$), the network learns only the residual (difference) b/w

Date: __/__/20 Day: _____

Input and output

$$F(x) = H(x) - x$$

Then,

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

- The skip connection adds the input x directly to the output $F(x)$

Result:

If some layers don't help, the shortcut lets information skip them - preventing gradient vanishing

Skip (Shortcut) Connection

- A skip connection bypasses one or more layers and adds the input x directly to the output $F(x)$
- Helps the model skip bad layers if they hurt performance
- Solves the vanishing gradient problem -> allows 100-1000 layers to train effectively
- Learning small adjustments ($F(x)$) is simpler & faster

Comparison

Concept	Explanation
Normal Network	Learns $H(x)$ directly
ResNet	learns residual $F(x)$ and adds input x
Benefit	Faster learning, avoids vanishing gradient

Similar Architectures

- Highway Networks -> also use skip connections with gates (like LLMs)
- But ResNet is simpler and gives better accuracy than highway networks

Date: __/__/20 Day: _____

ResNet Architecture

- Based on VGG-19 style, with 34-layer plain network, then skip connections added → forms ResNet-34
- Stacking multiple residual blocks builts deeper networks (ResNet-50, ResNet-101, ResNet-152)

MobileNet

MobileNet V2 is a lightweight and fast CNN architecture made for mobile and embedded vision applications

It improves MobileNet V1 with better accuracy & less computation

Background

- Deep learning models are usually large and slow not suitable for phones or IoT devices
- MobileNet V2 fixes this by using an efficient design that keeps good performance while using fewer parameters and memory

Key Features

1. Inverted Residuals
   - The core idea of MobileNet V2
   - Instead of compressing, it expands channels first and then reduces them back
   - Each block has 3 layers:
     - 1×1 conv (Expansion): Increases channels
     - Depthwise Conv: Applies convolution on each channal seperately
     - 1×1 Conv (Projection): Reduces channels to output size

This reduces computation and keeps accuracy

Date: __/__/20__ Day: _____

2- Depthwise Separable Convolutions

- A normal convolution is split into:
  - Depthwise convolution (per channel)
  - Pointwise convolution (1×1 to mix channels)
- This makes the network faster and smaller (used in MobileNet V1 too)

3- linear Bottlenecks

- Keeps input data information safe by using linear layers (no activation) at the output.
- Pattern

  1×1 Conv (expand) → Depthwise Conv → 1×1 Conv (project)
- Prevents information loss and improves accuracy

4- ReLU6 Activation

- Similar to ReLU but limited b/w 0 and 6
- Helps quantization (for mobile processors)
- keeps training stable and efficient

MobileNet Vs Architecture Overview

Structure

- Initial Conv layer: 32 filters, stride 2
- Inverted Residual Blocks: Repeated in several stages
  - Each has its own expression factor, stride, & output channel
- Final Conv layer: 1×1 conv with 1280 filters
- Global Average Pooling
- Fully Connected (Softmax) for classification

Advantages

- Efficient
- Flexible

Date: __/__/20__ Day: _____

- Compact
- Fast

Applications

- Image Classification
- Object Detection
- Semantic Segmentation
- Embedded Vision

Recurrent Neural Network

Introduction to RNN

RNNs are special neuron networks that can remember past information

Unlike normal networks (which go one-with-way from input → output), RNNs feed outputs from previous steps back into the network.

Example

When predicting the next word in a senstence, you use the previous works to guess correctly - just like RNNs remember past data.

Key Components

1- Recurrent Neurons

- Have a hidden state that stores info from previous inputs
- This helps them remember patterns across time steps

2- RNNs Unfolding (Unrolling)

- Shows how RNN Processes data over multiple time steps
- Each step depends on the previous one
- Used in Backpropagation Through time (BPTT) to update weights

Date: __/__/20 Day: _____

Architecture & Formulas

RNN uses shared weights across time steps

For each input x_i, it updates hidden state H_i:

- Hidden State

$$h = \sigma(Ux + Wh_{t-1} + B)$$

- U, W, B: weights & bias
- h_{t-1}: previous hidden state

- Output

$$Y = O(Vh + C)$$

- V, C: output weights & bias

- Overall Function:

$$Y = f(x, h, W, U, V, B, C)$$

How RNN Works

At every time step:

- Take input x_t and previous hidden state h_{t-1}
- Updates hidden states

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

- Outputs results

$$y_t = W_{hy} h_t$$

Weights are updated using Backpropagation Through Time (BPTT)

Backpropagation Through Time (BPTT)

- Extends normal backpropagation to handle sequences
- Gradients pass through all time steps
- Formula: Simplified

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial h_3} \cdot \sum_{k=1}^{3} \frac{\partial h_3}{\partial h_k} \cdot \frac{\partial h_k}{\partial W}$$

This helps RNN learn long or short dependencies

Date: __/__/20__ Day: _____

Types of RNN

Type	Description	Example
One-to-One	One Input → one output	Image classification
Many-to-One	One input → many outputs	Image captioning
Many-to-One	Many inputs → one outputs	Sentiment analysis
Many-to-Many	Many inputs → many outputs	Machine translation

Variants of RNN

- Vanilla RNN
  - Basic RNN with one hidden layer
  - Works for short-term data
  - Suffers from vanishing gradient problem

- Bidirectional RNN
  - Processes input forward and backward
  - Learns past + future context
  - Used in tasks like question answering

- LSTM (Long Short-Term Memory)
  - Has memory cells with gates:
    - Input Gate → add new info
    - Forget Gate → remove old info
    - Output Gate → decide what to output
  - Solves vanishing gradient issue
  - Handles long-term dependencies

- GRU (Gated Recurrent Unit)
  - Simplified LSTM
  - Combine input + forget gate → update gate
  - Trains faster, less complex

Date: __/__/20 Day: _____

RNN vs Feedforward Neural Network (FNN)

Feature	FNN	RNN
Flow	One-direction	Has feedback loops
Memory	no memory	Remembers previous input
Best for	Independence data	Sequential data (text, time series)

Advantages

- Memory: Keeps info from past inputs
- Good for sequences: Perfect for time-series & text analysis
- Combines with CNNs: Useful for video & Image sequence

Limitations

- Vanishing Gradient
- Exploding Gradient

Applications

- Time Series Prediction
- NLP
- Speech Recognition
- Image/video Processing

Difference b/w RNN & FNN

Neural Networks help solve complex ML tasks

Two main types

- Feed-Forward Neural Networks (FNNs)
- Recurrent Neural Networks (RNNs)

Both learn patterns from data but differ in structure and function

Feed-Forward Neural Networks (FNN)

- Data flows one way (input → output)

Date: __/__/20__ Day: _____

- No loops, no memory of Past inputs
- Best for static data (images, tables)
- Simple and fast to train
- Cannot handle sequences or time-based data
- Example: Classifying handwritten digits (MNIST)

Key Points:

- No memory
- works on independent inputs
- Easy and quick to train

Reaurrent Neural Network (RNN)

- Adds memory to the network
- Uses hidden states to remember past information
- Ideal for sequential data (text, speech, time series)
- Can face vanishing gradient problem
- Slower and more complex to train
- Example: Predicting the next word in sentence

Key points:

- Has memory
- works on sequence
- Harder and slower to train

When to Use

Use FNN for:

- Independent inputs (e.g image classification)
- Medical diagnosis (no relation b/w patients)
- Credit scoring (each case independent)

Use RNNs For:

- Language translation (word order matters)
- Stock price prediction (depends on past values)
- Wheather forcasting
- Speech Recognition

Date: __/__/20__ Day: _____

Computational Comparison

Feed-Forward Networks

- Simple structure:
- Parallel training
- Stable training
- Low resource use

Recurrent Neural Networks

- Sequential processing
- Training method
- Captures temperoal patterns
- High resources use

Backpropagation Through Time (BPTT)

It is the training method used for RNNs

Unlike normal backpropagation, it updates weights using errors from all time steps, $(t, t-1, t-2, \ldots)$

It "unfolds" the RNN across time and sums gradients for every step

Allows RNNs to learn long-term relationships in data

RNN Architecture

At each timestep t:

- Hidden State: $g_1(w_x X_t + w_s S_{t-1})$
- Output: $Y_t = g_2(w_y S_t)$

Where

- X_t : Input at time t
- S_t : Hidden state (memory)
- Y_t : Output
- w_x, w_s, w_y : Weights for input, hidden, & output
- g_1, g_2 : Activation functions

Date: __/__/20 Day: _____

Error Function

To measure prediction error at time t:

$$E_t = (d_t - Y_t)^2$$

Example at $t=3$

$$E_3 = (d_3 - Y_3)^2$$

This shows how far the prediction Y_3 is from the true value d_3

Updating Weights Using BPTT

BPTT updates W_y, W_s, W_x to reduce total error

Updating Output Weight (W_y)

- Directly affects the output Y_3
- Use chain rule:

$$\frac{\partial E_3}{\partial W_y} = \frac{\partial E_3}{\partial Y_3} \times \frac{\partial Y_3}{\partial W_y}$$

- Simple update — only affects current output

Updating Hidden State Weight (W_s)

- Affects all previous hidden states (since each state depends on the previous one)
- Gradient across all states:

$$\frac{\partial E_3}{\partial W_s} = \sum_{i=1}^{3} \frac{\partial E_3}{\partial Y_3} \times \frac{\partial Y_3}{\partial S_i} \times \frac{\partial S_i}{\partial W_s}$$

- Must propogate error through all past hidden state (S_3, S_2, S_1)

Updating Input Weight (W_x)

- Similar to W_s but focuses on inputs
- Gradient

$$\frac{\partial E_3}{\partial W_x} = \sum_{i=1}^{3} \frac{\partial E_3}{\partial Y_3} \times \frac{\partial Y_3}{\partial S_i} \times \frac{\partial S_i}{\partial W_x}$$

- Each input influences states & outputs

multiples hidden

Date: __/__/20 Day: _____

Vanishing Gradient & Exploding Gradient

Gradient: Slopes or derivatives used in backpropagation to update weights

If gradients are too small → vanishing, too large → exploding

Both cause problems in training deep networks

Vanishing Gradient Problem

What Happens

In deep networks, gradients become smaller as they move backward through layers

Early layers get almost no update, making training slow or stop completely

Why Its Happens

- In sigmoid or tanh activations, output range is small → derivatives also small (0-0.25)
- When multiplied through many layers, these small gradients become almost zero
- Deeper the network → worse the effect
- Especially happens when activations go into saturated regions (too high or too low)

How to Identify

- loss stops decreasing (stuck around same value)
- Accuracy doesn't improve
- Gradients close to 0 (check using visualization tools or histograms)
- Weights don't change much over epochs

How to Fix It

- Use ReLU Activation

Date: __/__/20__ Day: _____

- Batch Normalization
- Skip Connections (Rel ResNets)
- LSTM/GRU for RNNs
- Gradient Clipping

Types of RNN

- One-to-One
- One-to-Many
- Many-to-One
- Many-to-Many

One-to-One RNN

- Acts like a normal feedforward network
- One input → One output
- No sequence use, just one-time data

Use case: Binary Classification

One-to-Many RNN

- One input → Multiple Outputs (sequence)
- The model generates a sequence from one input

Use Case: Image captioning

Many-to-One RNN

- Multiple inputs (sequence) → One Output
- Model reads a full sequence, then predicts one result

Use Case: Sentiment Analysis

Many-to-Many RNN

- Sequence input → Sequence output
- Used when both input & output are sequences

Types: Synchronized & Asynchronous

- Use case: Machine translation.

Date: __/__/20__ Day: _____

Bidirectional RNNs

RNNs handle sequential data (speech, text, time series) using hidden states to remember past info

Problem: Normal RNNs suffer from vanishing gradients hard to learn long-term patterns

BRNNs fix this by reading data forward and backward, capturing both past and future context

How BRNN works

- Two hidden layers
  - Forward layer → reads sequence left → right
  - Backward layer → reads sequence right → left
- Both outputs are combined for the final prediction
- Result: The model understands the full context of a sequence

Steps In Working

- Input Sequence: Feed in a series of word or feature vectors
- Dual Processing
  - Forward: uses current input + previous hidden state
  - Backward: uses current input + next hidden state
- Hidden State Update: Non-linear activation maintains memory
- Output Generation: Each time step's output = function of combined forward + backward states

Advantages	Disadvantages
Better Context	Higher Computation
Higher Accuracy	longer Training
Handles Variable lengths	Not Real-Time
Noise Residence	Less Interpretability

Date: __/__/20__ Day: _____

LSTM, BiLSTM GRU

What is LSTM?

long Short-term Memory

It is an advanced version of RNN made by Hochreiter and Schmidhuber

It solves the problem of learning long-term dependencies — remembering important info for a long-time

Used for

- Speech Recognition
- Language Translation
- Time Series Forcasting

LSTM Architecture

LSTM adds a Memory Cell and 3 Gates

- Forget Gate — Decides what info to throw away

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

- Input Gate — Decides what new info to store

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\hat{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t$$

- Output Gate — Decides what to send as output

$$O_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = O_t \odot \tanh(C_t)$$

Where

- σ = Sigmoid function (0–1 range)
- $\odot$ = Element-wise multiplication

Date: __/__/20 Day: _____

How It Works

- LSTM has cells to store long-term memory
- Gates control what to keep, forget, and output
- Helps model remember old info & use new info wisely

Bidirectional LSTM (BiLSTM)

Normal LSTM: Reads sequence forward only

BiLSTM: Reads in both directions (forward + backward)

→ captures more context

Working of BiLSTM

- Has 2 LSTM layers
  - Forward LSTM → reads from start → end
  - Backward LSTM → reads from end → start
- Output: $P_t = P_t^f + P_t^b$

This helps the model understand past and future context together

Gated Recurrent Unit (GRU)

GRU = Simplified version of LSTM (by Cho et al., 2014)

It keeps performance similar but is faster & simpler

GRU Architecture

Uses 2 Gates only:

- Update Gate (z_t):

  How much old info to keep

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

- Reset Gate (r_t):

  How much past info to forget

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

Date: __/__/20 Day: _____

Then,

- Candidate Hidden State:

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t])$$

- Final Hidden State:

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

No separate cell state (uses only hidden state)

Generative Models In DL

Introduced by: Ian Goodfellow (2014)

Goal: Make machines generate new, realistic data by learning from real examples

GANs changed how AI creates, Arts, games, and more

Architecture

GAN = 2 neural networks competing & learning together:

- Generator (G) → Creates fake data
- Discriminator (D) → Judges if data is real or fake

Generators

- Takes random noise (z) as input
- Generates Fake but realistic data (eg image, text)
- Learns patterns from real data using backpropagation
- Goal: Fool the discriminator so fake looks real.

Generator Loss (Simplified)

It wants $D(G(z)) \to 1$, meaning "fake looks real"

Discriminator

- Binary classifier → decide "real" or "fake"
- Learns to improve accuracy using CNN
- Goal: Detect fake correctly

Date: __/__/20 Day: _____

Discriminator Loss (Simplified)

It wants $D(x) \to 1$ for real data & $D(G(z)) \to 0$ for fake

Training = MinMax Game

$$\min_G \max_D \left[E_{x \sim p_{data}}(\log D(x)) + E_{z \sim p_z}(\log(1 - D(G(x)))) \right]$$

- Generator (G) → minimize loss (fool D)
- Discriminator (D) → maximize loss (catch fakes)

How GAN Works

- Generator's Move
  - Starts with random noise → creates fake data
- Discriminator Turn
  - Gets both real & fake data → predicts real/fake
- Adversarial learning
  - If D catches fakes → D improves
  - If G fools D → G improves
- Training Loop
  - Both keep improving
- End Goal
  - D can't tell real/fake → G generates realistic data

Main Types of GANs

- Vanilla GAN
- Conditional GAN
- Deep Convolutional GAN
- Laplacian Pyramid GAN
- Super Resolution GAN
- Style GAN

Date: __/__/20 Day: _____

GANs Vs Transformer Models

Generative models help machines create things like images, text, and audio

Two popular types are:

- GANs (Generative Adversarial Networks)
- Transformers

Both can generate data, but they work differently

- GANs: use competetion b/w two networks
- Transformers use self-attention to find relationships in data
- Choosing the right one depends on your task type

Understanding GANs

Idea: Two neural networks play a game - one creates data, one judges it

Architectures

- Generator (G): Makes Fake data (tries to look real)
- Discriminator (D): Detects if data is real or fake

They train together - G improves to fool D, and D improves to catch fakes

Training continues until D can't tell fake from real

Training Goal

- G minimizes D's accuracy
- D maximizes its accuracy
  - This minimax game makes generated data look real

Understanding Transformer

Idea: Model that uses self-attention to understand & generate sequences (like text)

Date: __/__/20 Day: _____

How It Works

- Looks at all parts of input at once (parallel)
- Captures relationships b/w every word or token
- Used in models like GPT, BERT, ChatGPT

Main Parts

- Self-Attention : Every token looks at others → finds importance
- Encoder-Decoder : Encodes input → decodes to output
- Positional Encoding: Adds order info (since attention ignores position)

Results

Fast training, handles long sequences, great for text tasks

Real World Application

GANs	Transformers
Image generation	NLP
Style transfer	Chatbots
Data augmentation	Code generation
Synthetic datasets	Document understanding
Deepfakes	Search

Types of GANs

- Deep Convolutional GAN (DC
- Conditional GAN (cGAN)
- Cycle-Consistent GAN (CycleGAN)
- Super-Resolution GAN (SRGAN)
- StyleGAN

Date: __/__/20 Day: _____

Deep Convolutional GAN

DCGAN was proposed by researchers from MIT and facebook AI Research

It's a stable and convolution-Based GAN used for image generation (like clothes from Fashion MNIST)

Goal → make GAN training more stable & reduce mode collapse

Why DCGANs? (Need)

Problem:

- Normal GANs face mode collapse → generator produces only a few types of outputs

Example:

In MNIST (digits 0-9), generator may create only a few digits repeatly

Then discriminator learns only those - training gets stuck

Solution:

- DCGAN uses Convolutional layers + BatchNorm + ReLU / LeakyReLU → Better stability, less collapse

Architecture

- Generator
  - Input: 100 random numbers (noise)
  - Converts to shape 4×4×1024, then upsamples 4 times (Stride = 1/2)
  - Each layer doubles image size, reduces channels
  - Output: 64 × 64 × 3 image
  - Uses:
    - No fully connected layers
    - Batch Normalization (for stable training)
    - ReLU activations, except tanh in output layer
- Generates fake images similar to real ones.

Date: __/__/20__ Day: _____

- Discriminator
  - Input: Image (real or generated)
  - Works like a CNN classifier
  - Uses:
    - Strided Convolutions (instead of dense layers)
    - LeakyReLU activation
    - No pooling layers
  - Output: Single Score (real = 1, fake = 0)

Conditional Generative Adversarial Networks

CGANs are upgraded GANs that generate data based on conditions like labels or text.

In normal GANs → output is random

In CGANs → output depends on given info (condition)

Example:

If dataset = many car brands → we can make CGAN generate only "Mercedes" images by giving "Mercedes" as condition

So, CGAN = GAN + extra control → more targeted and precise data generation

Architecture and Working

CGAN adds conditioning information (y) to both Generator & Discriminator

- Generator
  - Inputs:
    - Random noise (z) → adds variation
    - Condition (Y) → guides what to make
  - Combines both to create data that matches conditions
    - Example: → if y = "cat", it generates cat images

Date: __/__/20 Day: _____

- Discriminator
  - Inputs
    - Real or fake Image (x)
    - Same condition (y)
  - Checks two things
    - Is image real or fake?
    - Does it match the label "y"?
  - Example → input image + label "cat": checks if it looks like a real cat

- Training (Adversarial Process)
  - Generator: tries to fool discriminator using ($z+y$)
  - Discriminator: tries to detect fake vs real using ($x+y$)

Goal:

Network	Goal
Generator	Make data that looks real for given label
Discriminator	Catch fake data and verify label match

Loss Function

CGAN objective

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

Meaning

- First term → discriminator should classify real data correctly
- Second term → generator should fool the discriminator
- E = Expected value
- P_{data} = real data distribution
- P_z = random noise distribution

During training → both improves → generator makes realistic images conditioned on y

Date: __/__/20 Day: _____

Cycle Generative Adversarial Network

What is GAN?

A Generative Adversarial Network (GAN) has 2 neural networks

- Generator (G): Creates fake images
- Discriminator (D): Checks if an image is real or fake

What is CycleGAN?

CycleGAN removes the need for paired data
It can translate images from one domain to another (like Photo ↔ painting, summer ↔ winter, horse ↔ zebra) without paired samples

example

- G: Turns photo → painting
- F: Turns painting → photo

Cycle GAN checks if converting and then converting back gives the same image. This is called Cycle Consistency.

Core Idea:

- $G(x)$ converts an image x from domain $X \to Y$
- $F(G(x))$ converts it back $Y \to X$
- The model measures how close $F(G(x))$ is to X.
- The smaller the difference, the better the mapping

Architecture of CycleGAN

- Generators (G and F)
  - $G: X \to Y$ (e.g: photo → artwork)
  - $F: Y \to X$ (e.g: artwork → photo)

Both generators learn style transfer b/w two images domains

Date: __/__/20 Day: _____

- Discriminators (D_X and D_Y)
  - D_X: Checks real X images vs. fake (F_y)
  - D_Y: Checks real Y images vs. fake $G(x)$

Each discriminator decides if an image looks real or generated.

- Cycle Consistency Loss Type:
  - Forward cycle loss:
    - G then F should return original image
    - $x \to G(x) \to F(G(x)) \approx x$
  - Backward Cycle Loss:
    - F then G should return original image
    - $y \to F(y) \to G(F(y)) \approx y$

Generator Architecture

Structure:

c7s1-64 → d128 → d256 → R256 (x6 or 9) → u128 → u64 → c7s1-3

Loss Function in CycleGAN

- Adversarial Loss

encourages generators to produce realistic images.

$$L_{advers}(G, D_y, X, Y) = \frac{1}{m}\sum(1 - D_y(G(x)))^2$$

$$L_{advers}(F, D_x, Y, X) = \frac{1}{m}\sum(1 - D_x(F(y)))^2$$

- Cycle Consistency Loss

Ensures mappings are reversible & meaningful

$$L_{cyc}(G, F, X, Y) = \frac{1}{m}[F(G(x)) - x) + (G(F(y)) - y)]$$

- Total Loss

$$L(G, F, D_x, D_y) = L_{advers}(G, D_y) + L_{advers}(F, D_x) + \lambda L_{cyc}(G, F)$$

Date: __/__/20 Day: _____

The goal is:

$$\min_{G,F} \max_{D_x, D_y} L(D, F, D_x, D_y)$$

Applications of Cycle GAN

- Collection Style Transfer
- Object Transformation
- Seasonal Transfer
- Photo ↔ Painting Conversion
- Photo Enhancement

Performance Evaluation

- AMT Perceptual Studies:
  - Human testers judge realism of generated images
- FCN Scores:
  - Measures pixel accuracy & IoU (object overlap accuracy)

Limitations

- Works best for texture changes, not shape changes
- Sometimes produces distorted or unnatural results
- Cannot significantly alter object structures

Super-Resolution (GAN)

SRGAN is a deep learning model used to increase image resolution - turning low-resolution (LR) images into high-resolution (HR) ones.

It solves the problem of blurry and smooth images produced by traditional upsalling methods (like bilinear interpolation)

Understanding the Problem

- Traditional upscalling = smooth but lacks fine details.

Date: __/__/20 Day: _____

- SRGAN learns textures, patterns, & edges → produces sharps, realistic HR images.
- Goal: Not Just reduce pixel difference but make images look real to humans

Architecture Overview

SRGAN = GAN Structure →

- Generator (G): Makes high-resolution images
- Discriminator (D): Checks if image is real or fake
  - Both compete → improves each other's performance

Loss Functions

- Content loss
  - Traditional methods use MSE → smooth results.
  - SRGAN uses VGG Perceptual Loss:
    - Compares feature maps from a pre-trained VGG-19 network (instead of pixels)
    - Helps model focus on textures, edges, & important features
  - Can use layers:
    - VGG2, 2: low-level details
    - VGG 5, 4: high-level textures

Formula:

$$L_{VGG} = \Sigma((\phi(i,j)(IHR) - \phi(i,j)(G(ILR)))^2)$$

Adversarial Loss:

- Encourages generator to make images that discriminator cannot detect as Fake
- Helps in producing sharp and ~~relas~~ realistic textures
  - Formula:

$$L_{Gen} = -\log(D(G(ILR)))$$

Date: __/__/20 Day: _____

Total (Perceptual) Loss:

- Combines both:
  - $L_{SR} = L_{Content} + 10^{-3} \times L_{Adversarial}$

This balance maintains both realism & accuracy

Training Process

- Downsample HR images → create LR versions
- Generator: creates HR from LR input
- Discriminator: checks if generated HR looks real
- Both networks improves through adversarial training

Results:

SRGAN produces images with better detail & texture (higher MOS score)

Limitations

- Training Instability
- High Computation
- Dataset Dependency
- Trade-off

Applications

- Medical Imaging
- Satelite imagery enhancement
- Mobile Photography

Variants

- ESRGAN
- Real-ESRGAN

Date: __/__/20__ Day: _____

StyleGAN

StyleGAN (by NVIDIA) is a generative model that creates highly realistic images, especially faces. Its main power: separates style from context, giving fine control over pose, texture, lightening color, and micro-details.

Architecture

StyleGAN uses a normal GAN framework, but changes the generator to give more control and quality

Discriminator stays similar to normal GANs

- Progresive Growing
  - Starts from tiny 4×4 images, then grows to 1024×1024 by adding layers.
  - Makes training stable by learning coarse → fine features
  - Improves overall image quality

- Bi-linear Sampling
  - Replaces nearest neighbor with bi-linear filtering when resizing
  - Smooths upsampling/downsampling → fewer artifacts & less pixelation

- Mapping Network & Style Network

Instead of sending latent vector z directly into generator

- z → 8-layer MLP → w
  - w controls high-level features (pose, color, texture)

Date: __/__/20 Day: _____

- Affine transform (A) converts $w \to y = (y_s, y_b)$ for AdaIN

AdaIN operation

$AdaIN(x_i, y) = y_{s,i} * ((x_i - \mu_i) / \sigma_i) + y_{b,i}$

- Normalizes each feature map
- Then applies scale & bias from style vector
- Controls how each layer looks

- The synthesis network has 18 conv layer (from 4x4 to 1024 x 1024)

- Constant Input + Noise Injection
- Generator does NOT use noise as input → starts with a learned constant 4 x 4 x 512 tensor
- Adds Gaussian noise per layer to create:
  - pores
  - wrinkles
  - hair texture
  - Small random details
  - This noise adds realism without changing structure

- Mixing Regularization
- Uses two latent vectors z1 & z2 applied at different layers
- Forces the network to handle mixed styles → more diversity and fewer artifacts

- Style Control at Different Resolution

StyleGAN controls image features at 3 levels, each with its own noise

Date: __/__/20 Day: _____

- Coarse (4×4 – 8×8)
  - Pose
  - Head shape
  - Broad structure
- Middle (16 × 16 – 32 × 32)
  - Face layout
  - Eyes, nose, mouth
  - Hair shape
- Fine (64×64 – 1024 × 1024)
  - Skin tone
  - Texture, pores
  - small color variations

- Feature Disentanglement

Two metrics measure how well StyleGEN separates feature

- Perceptual Path length
  - Measures smoothness when interpolating b/w latent vectors
  - Shorter = smoother transitions = better disentanglement

- Linear Separately
  - Tests if features (age, gender, smile, etc) can be separated with a simple classifier
  - Shows w-space is more disentangled than z-space

Results

- State-of-the-art performance on CelebA-HQ.
- NVIDIA released FFHQ dataset → more diversity
- StyleGAN produces extremely realistic faces
- Evaluated with FID score using 50,000 samples

Date: __/__/20 Day: _____

Autoencoders

Autoencoders are special neural networks that learn to compress data and then reconstruct it back.

They have two main parts:

- Encoder: Compresses the data (reduces dimensions)
- Decoder: Rebuilts the data from compressed form

They learn by reducing reconstruction error using MSE or Binary Cross-Entropy

Used For:

- Noise removal
- Feature extraction
- Error detection

Architecture of Autoencoder:

- Encoder

The encoder reduces the input into a smaller representation.

It has:

- Input layer: Raw data (image, text features, etc)
- Hidden layer: Transform data using weights + activations
- latent Output: Compressed vector (important features only)

- Bottleneck (Latent Space)
  - The smallest layer
  - Holds the most compressed & important information
  - Forces network to learn only key patterns
  - Improves generalization & efficient encoding

- Decoder
  - Rebuilts original data from latent vector
  - Hidden layers: Expand the vector. Output layer: Produces final reconstructed data

Date: __/__/20 Day: _____

Loss Function

Autoencoders train by minimizing reconstruction loss

- MSE: For continuous data
- Binary Cross-Entropy: For Binary data

Weights are updated using backpropagation so the model learns important features

How Autoencoders learn Efficient Representations

- Small Hidden layers: Forces focus on important features
- Regularization (L1/L2): Prevents overfitting, removes unnecessary weights
- Denoising: Add noise → model learns clean features.
- Activation Tuning: Creates sparsity → only a few neurons active

Types of Autoencoders

- Denoising Autoencoder
  - Input = násy
  - Output = clean
  - Helps model learn main structure without memorizing input.

- Sparse Autoencoder
  - Many hidden units
  - Only a few active at a time
  - Uses sparsity penalities (forces useful features)
  -

- Variational Autoencoder (VAE)
  - learns probability distribution of data
  - Can generate new data (images, text)
  - Encoder learns $q_\phi(z/x)$ • Decoder learns $q_\theta(x/z)$

Date: __/__/20 Day: _____

- Convolutional Autoencoder
  - Uses CNNs
  - Best for images
  - Encoder uses convolutions
  - Decoder uses deconvolutions /upsampling

Limitations of Autoencoders

- May memorize data instead of learning patterns
- Reconstructions may be blurr
- Need large datasets
- Require hyperparameter tuning (latent size, learning rate, etc)

Types of Autoencoders

Autoencoders are special neural networks used to compress data and then rebuild at again

They have 2 parts:

- Encoder : Compresses the input
- Decoder : Reconstructs the original input

They are useful for feature extraction, noise removal, anomaly detection, and dimensionality reduction.

- Vanilla Autoencoder

This is the basic and simplest autoencoder

It compresses the input into a smaller size and then tries to rebuild the same input

How it works

- Encoder compresses the data
- Decoder reconstructs the data
- Model is trained to reduce reconstruction error
- Uses backpropagation for learning

Date: __/__/20 Day: _____

Applications

- Data compression
- Feature learning
- Anomaly detection

- Sparse Autoencoder

This autoencoder forces only a few neurons to activate, making the model focus on the most important feature

How it works

- Adds L1 regularization
- Uses sparsity constraints
- Learns sharp and meaningful features
- KL divergence is also used to enforce sparsity

Applications

- Feature selection
- Dimensionality reduction
- Noise reduction

- Denoising Autoencoders

A denoising autoencoder learns to remove noise from the input

How it works

- Add noise to input data
- Train model to reconstruct clean data
- Learns strong & noise-resistant features

Applications

- Image denoising
- Audio/signal cleaning
- Preprocessing noisy data

Date: __/__/20 Day: _____

- Undercomplete Autoencoder

This autoencoder has a smaller hidden layer klhyp

The smallest layer forces the model to only keep the most important information

Applications

- Anomaly detection
- Feature Extraction
- Efficient data compression

- Contractive Autoencoder

It makes features stable even if the input changes slightly

How it works

- Adds a penalty to stop large changes in latent space
- Learns robust and smooth features

Applications

- Stable feature extraction
- Transfer learning
- Robust data representation

- Convolutional Autoencoder

A type of autoencoder that uses convolutional layers, best for image data

How it does

- Captures spatial features
- Learns patterns like edges, textures, shapes
- Uses Conv2D + MaxPooling + Upsampling layers

Applications

- Image reconstruction
- Image denoising

Date: __/__/20 Day: _____

Deep learning (Reinforcement) - DRL

What is Deep Reinforcement learning?

Deep learning (DRL) = Reinforcement Learning + Deep Neural Networks

It allows an agent to learn by:

- interacting with an environment
- taking actions
- receiving rewards
- improving its policy over time

DRL helps agents learn complex decisions directly from raw inputs like images or sensor data.

DRL is powered by:

- Q-Learning
- Policy Gradient methods
- Actor-Critic methods

Used in robotics, gaming, finance, healthcare, & many more areas

Main challenges:

- unstable training
- exploration vs. exploitation
- sample inefficiency
- safety concerns

But contineous research is improving DRL for real-world use

Core Concepts of DRL

- Agent - learner / decision-maker
- Environment - World where the agent operates
- State - Current situation of the environment
- Action - What the agent chooses to do

Date: __/__/20 Day: _____

- Reward - Feedback from environment
- Policy - Rule mapping states - actions
- Value Function - Expected total reward from a state
- Model - Predicts environment dynamics (optional)
- Exploration-Exploitation - Try new actions vs. use known good actions
- Learning Algorithm - Updates policy or value function
- Deep Neural Networks - Function approximators handling large state spaces.
- Experience Replay - Stores past experiences to improve training stability.

∴ All these work together so the agent can learn and make smart decisions.

How DRL Works

- Initialize agent and environment
- Interact: agent takes action → environment returns state + reward.
- Store Experience
- Learn from past experiences
- Update policy using DRL algorithms
- Balance exploration & exploitation
- Maximize rewards
- Converge to a stable policy
- Generalize to new situations
- Evaluate agent performance
- Deploy in real tasks

Applications of DRL

- Gaming • Robotics • Finance • Healthcare • Energy • NLP

Date: __/__/20 Day: _____

Deep Reinforcement Learning Advancements

- Early Breakthrough
  - DRL became famous after DeepMind's DQN beat Atari games.
  - Showed DRL can learn directly from pixels.

- Modern Improvements
  - PPO and TRPO → more stable training
  - Actor-Critic models → better convergence
  - Distributional DRL → Better reward prediction

- Using Prior Knowledge
  - Hierarchical RL → break tasks into small parts.
  - Transfer Learning → reuse learned skills
  - Combine simulation & real world

- Hybrid Approaches
  - Model-Based + model-free methods combined
  - Curiosity-Based exploration → agents explore more smartly
  - Intrinsic motivation → agent learns even without rewards

Reinforcement Learning

Reinforcement learning is a part of machine learning where an agent learns by doing actions, getting rewards or penalties, and improving over time. It learns through trail & error to maximize total reward.

Basic Idea of RL

RL works on interaction

- Agent → makes decisions

Date: __/__/20 Day: _____

- Environment → where the agent works
- State → current situation
- Action → move/decision
- Reward → Feedback (good/bad)

The agents keeps learning which actions give the best long-term reward.

Core Components

- Policy
  - Rules the agent follows to choose actions
  - Can be simple or complex
  - Example: A self-driving car stops when its detects a person.

- Reward Signal
  - Tells the agent what's good or bad.
  - Positive reward = good, negative = bad.
  - Example: Fewer crashes = higher reward

- Value Function
  - Measures future benefits, not just immediate reward
  - Focuses on long-term success
  - Example: Avoiding risky moves for long-term safety

- Model
  - Predicts what will happen next
  - Helps the agent plan ahead
  - Example: Predicting car movements to avoid accidents

Date: __/__/20 Day: _____

How RL works?

- Agent sees current state
- Takes an action
- Environment gives reward + new state
- Agent updates its learning
- Repeats again & again

This process is called Mardov Decision Process

Q-Learning

Q-Learning updates values like this:

$$Q(s, a) = Q(s, a) + \alpha [r + \gamma \max Q(s', a') - Q(s, a)]$$

Meaning:

New Q-Value = old + learning rate x (reward + future reward - old value)

Types of Reinforcement

- Positive Reinforcement
  - Add something good → behavior increases
    - Advantages: High motivation
    - Disadvantages: Oversuse = less effect
- Negative Reinforcement
  - Remove something bad → behavior increases
    - Advantages: Ensures minimum performance
    - Disadvantages: Only enough to avoid penalty

Applications of RL

- Robotics
- Games
- Industrial Control

Advantages

- Works well for complex decision problems
- No labeled data needed
- Learns from experience

Date: __/__/20 Day: _____

Disadvantage

- Needs a lot of data + computation
- Hard to design reward function
- Not good for simple problems
- Hard to debug

Markov Decision Process

A Mardov Decision Process is a way to describe how an agent (robot, game player, machine) makes decisions step-by-step in an environment where results are uncertain

MDP Tells:

- What action should the agent take?
- What will happen after that action?
- Is the result good or bad?

MDPs are used in AI to model situations where decisions happen one after another and outcomes are not guaranteed

Five Key Components of MDP

- States (S)
  - A state = the condition or position of the agent
  - Example: being at grid cell (1, 1)
- Actions (A)
  - Actions = choices the agent can make
  - Example: move UP, DOWN, LEFT, RIGHT
- Transition Model (T)
  - Tells what happens after taking an action
  - Movement is uncertain (stochastic)
  - E.g: 80% → move in intended direction
    - 10% → slip left, 10% → slip right

randomness = stochastic

Date: __/__/20 Day: _____

- Reward (R)
  - A number given after an action
  - Positive = good result
  - Negative = bad result
  - Example:
    - +1 for goal
    - -1 for falling into fire
    - -0.04 for each step
    - 0 for hitting a wall (no movement)
- Policy (π)
  - A policy = the agent's plan
  - It tells: "In this state → take this action"
  - The goal: find the best policy to get the highest total reward over time

Key Algorithms in Deep Reinforcement learning

Deep Q-Networks (DQN)

Deep Q-Learning is a reinforcement learning method that uses a neural network to help an agent make decisions in complex environments (like games, robotics, images)

It is an improved version of Q-learning

Why not only Q-learning?

Q-learning stores Q-values in a table

This works only when states are small

Deep-Q-learning fixes this by using a deep neural network to estimate Q-Values - so it can handle millions of states.

Date: / /20 Day:

Key Problems Solved by Deep Q-Learning

- High number of states
  - Q-table becomes too big → neural network replaces the table
- Continuous data (like images)
  - Neural networks can process pixels, signals, & large inputs easily
- Scalability
  - Work for large tasks: video games, robots, self-driving cars

Deep Q-Network (DQN) Architecture

A DQN has 3 main parts

- Neural Network
  - Takes a state (eg images) as input
  - Outputs Q-values for every action
  - Approximates: $Q(s, a)$
- Experience Replay
  - Stores past experience: (state, action, reward, next state)
  - Random mini-batch sampling helps break correlation
  - Makes training stable and efficient
- Target Network
  - A copy of the main network
  - Used to calculate target Q-values
  - Updated after a few steps for stability

Loss Function

DQN tries to reduce the difference b/w

- Predicted Q-Value
- Target Q-value = reward + γ × max(next state Q-values)

This difference is squared (MSE Loss)

Date: __/__/20 Day: _____

Training steps

- Initialize
- Exploration vs. Exploitation (ε-greedy)
- Collect Experience
- Training Update
- Update Target Network
- Reduce ε

REINFORCE

It is a reinforcement learning (RL) algorithm that learns a policy (a way to choose actions). It improves decisions by trying actions, looking at the total reward, and increasing the chance of actions that led to higher rewards

It does NOT estimate Q-values like Q-Learning. Instead, it directly improves the policy using policy gradients

It works well when actions are continuous or there are many actions

How REINFORCE Works?

- Collect Episodes
  - The agent runs in the environment using current policy
  - It collects a trajectory: states, actions, rewards
- Compute Returns

For each time step, t, compute the return

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} R_k$$

- G_t : total future reward from step t
- γ : discount factor
- T : last time step

Date: __/__/20 Day: _____

- Policy Gradient Update

Policy parameters θ are updated as:

$$\theta_{new} = \theta + \alpha \nabla_\theta \log \pi_\theta (a_t | s_t) G_t$$

Meaning:

- Increase probability of actions that produced high returns
- α = learning rate
- π = policy (probability of choosing an action)

- Repeat

Run many episodes → improve policy over time

Variants of REINFORCE

- REINFORCE with Baseline

Substract a baseline value (usually $V(s)$) to reduce variance.

$$\theta_{new} = \theta + \alpha \nabla_\theta \log \pi_\theta (a_t | s_t)(G_t - b_t)$$

- Actor- Critic
  - Actor chooses actions
  - Critic evaluates actions
  - More stable and faster learning
  - Lower variance

Advantages

- Simple & easy to implement
- Directly improves policy
- Good when tasks have clear episodes endings
- Works well with many or continuous actions

Challenges

- High variance
- Sample inefficient
- Slow convergence

Date: __/__/20 Day: _____

Actor-Critic Algorithm

What is Actor-Critic?

It is a reinforcement learning algorithm that has 2 parts:

- Actor: chooses actions (decision maker)
- Critic: Evaluates how good those actions are (feedback giver)

Actor learns how to act, Critic learns how to judge

This balance makes learning faster & more stable

Key Terms

- Policy (Actor)
  - Written as $\pi(a|s)$
  - It gives the probability of taking a in state s
  - Actors improves this policy to maximize total reward
  - Actor network has parameters θ
- Value Function (critic)
  - Written as $V(s)$
  - Predicts the expected future reward from state s
  - Critic's parameters = w

How Actor-Critic Works

Actor-Critic uses two objectives

- Policy Gradient (Actor Objective)

Formula:

$$\nabla\theta J(\theta) \approx (1/N) \Sigma \nabla\theta \log \pi\theta(ai|si) \times A(si, ai)$$

Meaning

- $J(\theta)$ = expected return
- $\pi\theta(a|s)$ = actor's policy
- $A(s, a)$ = advantage
- N = number of samples

The actor improves actions with positive advantage

Date: __/__/20 Day: _____

and avoids actions with negative advantage

- Value Function Update (Critic Objective)

Formula:

$$\nabla_w J(w) \approx (1/N) \sum \nabla_w (V_w(s_i) - Q_w(s_i, a_i))^2$$

Meaning:

- Critic reduces the error b/w predicted value $V(s)$ and actual action value $Q(s, a)$
- Updates parameters w using gradient descent.

Updates Rules

- Actor Update

$$\theta(t+1) = \theta(t) + \alpha \nabla_\theta J(\theta(t))$$

- α = actor learning rate

- Critic Update

$$w(t+1) = w(t) - \beta \nabla_w J(w(t))$$

- β = critic learning rate

Advantage Function

$$A(s, a) = Q(s, a) - V(s)$$

Meaning:

- Tells how much better an action is than the average expected value of that state

∴ Actor uses this to decide which actions to increase or decrease

Variants of Actor - Critic

- A2C (Advantage Actor-Critic)
  - Uses advantage function $A(s, a)$
  - Reduces variance → more stable learning
    - $A(s, a) = Q(s, a) - V(s)$

Date: __/__/20__ Day: _____

- A3C (Asynchronous Advantage Actor-Critic)
  - Many agents run in parallel
  - Each agent updates the global network
  - Faster and more stable because updates are less correlated

Advantages of Actor-Critic
- More sample efficient
- Faster convergence
- Handles discrete + continuous actions
- Some versions allow off-policy learning
- More Stable than plain policy gradient methods.

Proximal Policy Optimization

PPO is a reinforcement learning (RL) algorithm
It improves the policy like other policy gradient methods
But it uses clipping to stop very big updates
This makes training stable, safe, and reliable
PPO is popular in robotics, games, and gen AI

PPO vs Older Methods
- REINFORCE
  - Simple but unstable
  - High variance
  - PPO fixes this by limiting policy changes
- Actor-Critic
  - Actor chooses actions
  - Critic evaluates actions
  - PPO acts similarly but uses:
    - Value function
    - Advantage estimation
    - Clipping
      - Makes training more stable

Date: __/__/20 Day: _____

Role of PPO in Generative AI

Reasons PPO is used in GenAI

- RLHF (Human Feedback) → core algorithm
- Stable training for huge models
- Safe exploration so models don't generate harmful content
- Efficient optimization for large datasets
- Better human-like-interaction (aligned & coherent outputs)

PPO Parameters (Hyperparameters)

- Clip Range (ε)
  - limits how much new policy can change from old policy
- Learning Rate
  - Step size for updating weights
- Discount Factor (γ)
  - Weight given to future rewards
- GAE Lambda (λ)
  - Controls bias-variance tradeoff for advantage
- Number of Epochs
  - How many times each batch is reused
- Batch Size
  - Number of samples per update
- Value Loss Coefficient (c_1)
  - Weight of critic loss
- Entropy Coefficient (c_2)
  - Encourages exploration by penalizing low entropy

PPO Mathematics

- Policy Update Rule
  - PPO updates the policy using gradients
  - Ensures updates are safe and controlled

Date: __/__/20 Day: _____

- Surrogate Objective

Formula:

$$L(\theta) = E_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t\right]$$

Meaning:

- Ratio compares new vs old policy
- Advantage tells if action was good or bad
- Helps update policy without huge jumps

- Clipping Mechanism

$$clip(r, 1-\epsilon, 1+\epsilon)$$

- r = ratio (new policy / old policy)
- Clipping prevents extremely large updates
- Makes training stable

- Advantage Estimation
- Calculates how much better or worse an action was
- Used to update policy correctly

Integrating PPO with Generative AI

Ways PPO helps GenAI:

- Multimodel Alignment (text + image + audio)
- Personalization for users
- Online learning with continuous feedback
- Safety-aware generation using reward penalties
- Task-specific fine tuning (legal, tutoring, medical)

How PPO Works

- Collect Experiences
  - Get states, actions, rewards

Date: __/__/20 Day: _____

- Compare/Compute Advantage
  - Check if action was good or bad
- Update Policy
  - Use surrogate + clipping
- Update Value Function
  - Improve critic's prediction
- Repeat
  - Continue until performance becomes stable

Applications of PPO

- Robotics
- Game AI
- Self-Driving
- Finance

Advantages of PPO

- Very stable training
- Good sample efficiency
- Simple
- Works for both continuous + discrete actions
- Reliable performance

Disadvantages of PPO

- High computation cost
- Needs hyperparameter tuning
- Can still require many samples
- No theorectical guarantee like TRPO
- Risk of overfitting batches