Offline Exam Handbook

Table Of Contents

Module C

Autoregressive Models

Understanding Autoregressive Models

Definition: Autoregressive (AR) models predict future values in a time series based on a linear function of previous observations. They are widely used in time-series forecasting, econometrics, and natural language processing.

Equation:

$$X_t = b + \sum_{i=1}^{p} w_i X_{t-i} + \epsilon_t \tag{1}$$

$$X_t = b + w_1 X_{t-1} \quad \text{(AR(1))} \tag{2}$$

$$X_t = b + w_1 X_{t-1} + w_2 X_{t-2} \quad \text{(AR(2))} \tag{3}$$

State Transition Matrix:

$$\begin{matrix} X_t \\ X_{t-1} \\ . \\ X_{t-p+1} \end{matrix} = \begin{matrix} w_1 & w_2 & \dots & w_p \\ 1 & 0 & \dots & 0 \\ . & . & \ddots & . \\ 0 & \dots & 1 & 0 \end{matrix} \begin{matrix} X_{t-1} \\ X_{t-2} \\ . \\ X_{t-p} \end{matrix} + \begin{matrix} b \\ 0 \\ . \\ 0 \end{matrix} \tag{4}$$

Sequence Modelling Auto-Regressive Models

Definition: A time-series model where current values depend on past values.

Equation:

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \epsilon_t \tag{5}$$

Special Case: AR(1):

$$X_t = \phi X_{t-1} + \epsilon_t \tag{6}$$

Markov Chain Connection:

- Transition probability: $P(X_{t+1}|X_t)$
- Can be represented as a transition matrix.

Binary Cross-Entropy Loss (BCELoss)

Definition

Binary Cross-Entropy (BCE) is a widely used loss function for binary classification tasks. It measures the difference between two probability distributions and is commonly used in logistic regression and neural networks.

Mathematical Formulation:

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)] \tag{7}$$

Properties of BCE Loss

- Penalizes incorrect predictions heavily.
- Ensures stable training with a small ϵ to avoid log(0) errors.
- Works well for probabilistic models in classification tasks.

NLP Pipeline

Natural Language Processing (NLP) involves preprocessing textual data before feeding it into machine learning models. The key steps include:

Tokenization

- Splitting text into words or subwords.
- Helps in feature extraction for models.
- Example: Hello, world!" → [Hello", ," , world", "!"]

Part-of-Speech (POS) Tagging

- Assigns grammatical roles to words (noun, verb, adjective, etc.).
- Used in syntactic parsing and named entity recognition.
- Example: The cat sleeps"→ [(The", DET), (cat", NOUN), (sleeps", VERB)]

Text Normalization

- Converts words into a standard format to improve model performance.
- Includes lowercasing, stemming, and lemmatization.

Lowercasing

- Converts text to lowercase to avoid treating Hello" and hello" as different words.

Stemming

- Reduces words to their root form by removing suffixes.
- Example: running" → run"

Lemmatization

- Converts words to their dictionary form.
- Example: better" → good"

Stopword and Punctuation Removal

- Removes commonly occurring words (e.g., the", is", "and") that do not add significant meaning.
- Helps reduce dimensionality and improve model performance.

Understanding N-Grams

Definition: A sequence of N items from a given text.

Types:

- Unigram: $\{word_1, word_2, \ldots\}$
- Bigram: $\{word_1 word_2, word_2 word_3, \ldots\}$
- Trigram: $\{word_1 word_2 word_3, \ldots\}$

Probability Calculation:

$$P(w_n|w_{n-1}) = \frac{count(w_{n-1}, w_n)}{count(w_{n-1})} \tag{8}$$

$$P(w_n|w_{n-2}, w_{n-1}) = \frac{count(w_{n-2}, w_{n-1}, w_n)}{count(w_{n-2}, w_{n-1})} \tag{9}$$

Recurrent Neural Networks (RNNs)

Definition

A Recurrent Neural Network (RNN) is a type of artificial neural network designed to process sequential data. Unlike traditional feedforward networks, RNNs use internal memory (hidden states) to capture dependencies in sequential inputs.

Basic Structure:

- Input Layer: Accepts sequential data.
- Hidden Layer with Loops: Stores past information.
- Output Layer: Produces predictions.

Mathematical Formulation:

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b) \quad (10)$$

$$o_t = W_o h_t \quad (11)$$

Types of RNNs:

- One-to-One: Standard feedforward network.
- One-to-Many: Single input, multiple outputs.
- Many-to-One: Multiple inputs, single output.
- Many-to-Many: Sequence-to-sequence tasks.

Backpropagation Through Time (BPTT):

- Used to train RNNs by unrolling through time and applying backpropagation.

Challenges of RNNs

- **Vanishing Gradient Problem**: Gradients shrink exponentially over time, making it difficult for the network to learn long-term dependencies.
- **Exploding Gradient Problem**: Gradients grow exponentially, leading to unstable training.
- **Limited Memory**: Standard RNNs struggle with capturing long-range dependencies in sequential data.

Long Short-Term Memory (LSTM)

Definition

LSTMs are an advanced type of RNN designed to address the vanishing gradient problem by incorporating memory cells that selectively retain information over long sequences.

LSTM Architecture

LSTM networks consist of three gates:

- **Forget Gate**: Controls what portion of the past information should be discarded.
- **Input Gate**: Regulates what new information should be added to memory.
- **Output Gate**: Determines the final output based on the cell state.

Mathematical Formulation: Forget Gate:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{12}$$

Input Gate:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{13}$$

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C) \tag{14}$$

Cell State Update:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{15}$$

Output Gate:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{16}$$

$$h_t = o_t * \tanh(C_t) \tag{17}$$

Gated Recurrent Units (GRUs)

Definition

GRUs are a variant of LSTMs that use gating mechanisms to control information flow but require fewer parameters than LSTMs.

Mathematical Formulation

Reset Gate:

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r) \tag{18}$$

Update Gate:

$$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z) \tag{19}$$

Candidate Hidden State:

$$\tilde{h}_t = \tanh(W_h[r_t * h_{t-1}, x_t] + b_h) \tag{20}$$

Final Hidden State:

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{21}$$

Bidirectional RNNs

Definition

A Bidirectional RNN (BiRNN) processes input sequences in both forward and backward directions to improve context understanding.

Mathematical Formulation:

$$h_t^{forward} = f(W_x x_t + U h_{t-1}^{forward}) \tag{22}$$

$$h_t^{backward} = f(W_x x_t + U h_{t+1}^{backward}) \tag{23}$$

$$h_t = [h_t^{forward}, h_t^{backward}] \tag{24}$$

Applications of RNNs, LSTMs, and GRUs

- **Natural Language Processing (NLP)**: Machine translation, text generation, and sentiment analysis.
- **Speech Recognition**: Transcribing spoken language into text.
- **Time-Series Forecasting**: Predicting financial market trends and weather patterns.
- **Bioinformatics**: Analyzing DNA sequences and protein structures.

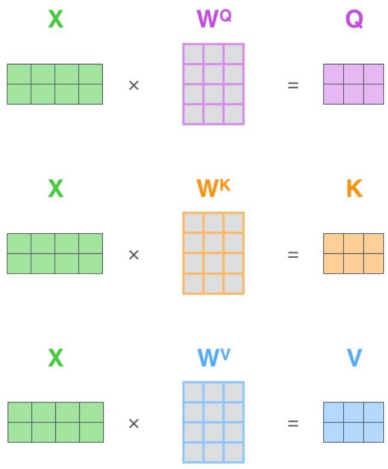


FIGURE 8.8: Input transformation used in a Transformer, $Q = \text{Query}$, $K = \text{Key}$, $V = \text{Value}$. Image Source: (Alammar 2018)

Figure 1: Input transformation used in a Transformer. The input matrix X is multiplied by weight matrices W^Q, W^K, and W^V to produce Query (Q), Key (K), and Value (V) matrices.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW^K, VW^V)$$

The attention mechanism is defined as:

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

Sentiment Classification

Definition:
Sentiment classification is the task of identifying the emotional tone behind a body of text, typically categorized as **positive**, **negative**, or **neutral**.

Goal:

To automatically determine the sentiment expressed in text using computational methods.

Applications:

- Product reviews
- Social media monitoring
- Customer feedback analysis

Approaches:

1. **Rule-based:** Uses manually defined rules and sentiment lexicons (e.g., SentiWordNet).
2. **Machine Learning:** Trains models (e.g., Naive Bayes, SVM) on labeled data.
3. **Deep Learning:** Uses neural networks (e.g., RNNs, LSTMs, Transformers) for better context understanding.

Steps Involved:

- **Text Preprocessing:** Tokenization, stopword removal, stemming/lemmatization
- **Feature Extraction:** Bag of Words, TF-IDF, word embeddings
- **Model Training:** Using labeled data to train classifiers
- **Prediction:** Classify unseen text into sentiment categories

Popular Tools/Libraries:

- NLTK
- TextBlob

- Scikit-learn
- Hugging Face Transformers

Word Embeddings

Definition:

Word embeddings are dense vector representations of words in a continuous vector space where semantically similar words are mapped close together.

Purpose:

To capture syntactic and semantic meaning of words for use in machine learning models.

Characteristics:

- Low-dimensional (e.g., 100–300 dimensions)
- Real-valued vectors
- Context-based representations

Popular Techniques:

- **Word2Vec:** Uses two architectures:
  - **CBOW (Continuous Bag of Words):** Predicts the current word from surrounding context.
  - **Skip-gram:** Predicts surrounding context words given the current (center) word.
- **GloVe (Global Vectors):** Combines global word co-occurrence statistics.
- **FastText:** Improves Word2Vec by using subword information.
- **BERT Embeddings:** Contextual embeddings generated using transformer models.

Advantages:

- Captures semantic similarity (e.g., king - man + woman = queen)
- Improves performance in NLP tasks

Applications:

- Text classification
- Machine translation
- Question answering
- Named entity recognition (NER)

BLEU Score (Bilingual Evaluation Understudy)

Definition:
BLEU is an automatic evaluation metric for comparing machine-generated text (e.g., translations) against one or more reference texts using *n*-gram precision.

Purpose:
To measure the quality of machine translation by checking how many *n*-grams in the candidate sentence appear in the reference sentences.

Formula:

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

Where:

- p_n = modified precision for *n*-grams (usually $n = 1$ to 4)
- w_n = weight for *n*-gram (commonly $w_n = \frac{1}{N}$)
- BP = brevity penalty to penalize short candidates

Brevity Penalty:

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \le r \end{cases}$$

- c = length of candidate translation
- r = effective reference length

Machine Translation (MT)

Definition:

Machine Translation is the task of automatically translating text from one language to another using computational models.

Evolution:

- **Statistical MT:** Early systems based on word/phrase probabilities.
- **Seq2Seq Models:** Encoder-decoder neural networks using RNNs.
- **Attention Mechanism:** Solves long-range dependency issues in Seq2Seq.
- **Transformers:** Fully attention-based models, state-of-the-art in MT.

Preprocessing Pipeline:

- **Tokenization:** Splits text into meaningful units using language-specific tokenizers.
- **Vocabulary Building:** Maps words to indices; includes special tokens like <sos>, <eos>, <pad>, and <unk>.
- **Numericalization:** Converts token sequences into index sequences for model input.

Introduction to Transformers

Transformers revolutionized machine translation by overcoming the limitations of RNNs:

- **RNN Issues:** Struggles with long-range dependencies, poor parallelization, and fading context.
- **Transformer Strengths:** Self-attention allows direct modeling of dependencies, facilitates parallel computation, and maintains context across sentences.

Positional Encoding

Transformers are permutation-invariant. Positional encodings are added to token embeddings to provide order information.

Encoder

The encoder processes input sequences and creates contextualized embeddings. It consists of the following components:

- Input embedding

- Positional encoding
- Multi-head self-attention
- Feed-forward network
- Layer normalization + residual connections

Decoder

The decoder generates the output sequence by attending to both previous tokens and the encoder's output. It consists of the following components:

- Output embedding
- Positional encoding
- Masked multi-head self-attention
- Encoder-decoder attention
- Feed-forward network
- Linear + softmax layer

Complete Transformer Model

The Transformer model consists of:

- **Encoder:** Processes the source sequence in parallel.
- **Decoder:** Generates the target sequence, attending to the encoder's output.
- **Training:** Uses teacher forcing with a shifted version of the target sequence.

Semi-Supervised Learning (SSL)

Introduction

Semi-Supervised Learning (SSL) combines a small amount of labeled data with a large pool of unlabeled data. It strikes a balance between supervised learning (using labeled data) and unsupervised learning (using only unlabeled data), leveraging both to improve model performance.

Motivation

Labeled data is costly to acquire, while unlabeled data is abundant. SSL utilizes unlabeled data to enhance learning, reducing reliance on labeled data.

Mathematical Formulation

Let:

- $D_L = \{(x_i, y_i)\}_{i=1}^{l}$: Labeled data
- $D_U = \{x_i\}_{i=l+1}^{l+u}$: Unlabeled data

The objective is to minimize:

$$\mathrm{L}(f) = \mathrm{L}_{\text{supervised}}(f; D_L) + \mathrm{L}_{\text{unsupervised}}(f; D_U)$$

Importance

SSL reduces the need for large labeled datasets, making it particularly useful in domains with limited labeled data but abundant unlabeled data.

Key Assumptions in SSL

- **Self-Training**: Confident predictions on unlabeled data are added to the training set.
- **Co-Training**: Two classifiers, trained on different views, teach each other.
- **Cluster Assumption**: Points in the same cluster likely share the same label.
- **Manifold Assumption**: Data points lie on a low-dimensional manifold.

Inductive vs Transductive SSL

- **Inductive SSL**: Learns a general classifier applicable to unseen data.
- **Transductive SSL**: Focuses on labeling a fixed set of unlabeled data.

Ladder Networks and -Models (Pi-Models)

Ladder Networks

A Ladder Network combines supervised and unsupervised learning by denoising internal representations:

- **Encoder**: Adds noise to the input and processes it.
- **Decoder**: Recovers clean activations for each layer.
- **Skip Connections**: Form a ”ladder” structure between encoder and decoder.

Loss function:

$$L = L_{supervised} + \sum_{l} \lambda_l L_{reconstruction}(l)$$

Where λ_l is a weight for each layer's reconstruction loss.

Pi-Model

The Pi-Model enforces consistency regularization by ensuring stable predictions under input perturbations:

- Input x is passed through the network twice with different augmentations/noise.
- Two outputs: $f_1(x + \epsilon_1)$ and $f_2(x + \epsilon_2)$.

Loss function:

$$L_{unsupervised} = ||f_1(x) - f_2(x)||^2$$

This encourages predictions to remain consistent under small input changes.

Variational Autoencoders (VAEs)

Introduction

VAEs model latent representations as probability distributions, enabling data generation by sampling from the learned distribution.

Intuition

The encoder compresses an input into a latent variable z, treated as a random variable. This enables diverse outputs from the same input.

VAE Loss

$$L = E_{q(z|x)}[\log p(x|z)] - KL(q(z|x) \,||\, p(z))$$

Consists of:

- **Reconstruction Loss**
- **KL Divergence**

Why Use a Distribution?

Modeling z as a distribution allows infinite variants of data generation.

Diffusion Models

Diffusion models reverse noise addition across steps for sharper outputs, unlike VAEs' single-step approach.

VAEs in SSL

Extended with a classifier, using reconstruction, KL divergence, and classification losses for labeled data, and pseudo-labels for unlabeled data.

Applications

- **Medical Imaging**: Small labeled datasets, synthetic augmentation.
- **Autonomous Driving**: Unlabeled video frames.

Conditional VAEs

CVAEs guide generation by conditioning on a label y, e.g., generating a digit "4" by conditioning on $y = 4$.

Introduction to Reinforcement Learning

Definition and Basic Concepts

Reinforcement Learning (RL) is a machine learning paradigm where an agent learns to make decisions through interaction with an environment to maximize rewards.

Comparison with Other Learning Paradigms

Key Components of Reinforcement Learning

Agent

Learns and makes decisions to maximize cumulative reward.

Environment

Responds to agent's actions, provides feedback, and can be deterministic or stochastic.

State/Observation

Represents the environment's current situation. States can be complete or partial.

Action

Decisions made by the agent that affect the environment.

Reward

Scalar feedback indicating the quality of actions taken by the agent.

Policy

Strategy that defines how actions are selected.

Exploration vs. Exploitation Dilemma

Balancing between exploring new actions and exploiting known good actions.

Common Approaches

- ϵ-greedy
- Decaying ϵ-greedy
- Softmax exploration
- Upper Confidence Bound (UCB)
- Thompson Sampling
- Intrinsic Motivation/Curiosity

Types of Reinforcement Learning Algorithms

- Model-based vs. Model-free
- Value-based vs. Policy-based
- On-policy vs. Off-policy
- Deterministic vs. Stochastic Policies

Concepts in Reinforcement Learning

Episodes

An episode is a sequence from an initial state to a terminal state:

- Finite in length, with a defined start and endpoint.

State Spaces

The state space is the set of all possible states:

- **Discrete**: Finite number of states
- **Continuous**: Infinite number of states.
- **Fully Observable**: Complete state visible to the agent.
- **Partially Observable**: Only partial state visible to the agent.

Markov Property

The Markov property states that the future depends only on the current state:

- Forms the foundation of Markov Decision Processes (MDPs).
- Simplifies learning by discarding past history.

Challenges in Reinforcement Learning

Delayed Rewards

Challenges with delayed rewards:

- **Credit Assignment Problem**: Determining which actions led to the final reward.
- Approaches: Temporal Difference Learning, Eligibility Traces, Reward Shaping, Hierarchical RL.

Continuous vs. Discrete Action Spaces

- **Discrete Actions**: Finite set (e.g., Q-learning).
- **Continuous Actions**: Infinite set (e.g., DDPG, SAC, PPO).

Multi-Armed Bandit Problem (MAB)

Problem Overview

The goal in the multi-armed bandit (MAB) problem is to maximize the expected total reward by selecting actions (arms) over time, while facing uncertainty in reward distributions. We aim to learn which actions yield better rewards.

Reward Estimation

Each arm corresponds to an unknown reward distribution. We estimate the expected reward using the sample mean:

$$Q_t(a) = \frac{1}{N_t(a)} \sum_{i=1}^{N_t(a)} R_i$$

where $Q_t(a)$ is the estimated value of arm a at time t, and $N_t(a)$ is the number of times arm a has been selected up to time t.

Incremental Update Rule

To avoid storing all past rewards, we update the estimated value incrementally:

$$Q_n = Q_{n-1} + \frac{1}{n}(R_n - Q_{n-1})$$

where Q_n is the updated estimate after observing reward R_n at step n.

Greedy Algorithm

The greedy algorithm selects the arm with the highest estimated reward:

$$\text{Action} = \arg\max_a Q_t(a)$$

However, it may converge prematurely to suboptimal arms and avoid exploring.

Epsilon-Greedy Algorithm

The Epsilon-Greedy algorithm balances exploration and exploitation:

- With probability ϵ, select an arm randomly (exploration).
- With probability $1-\epsilon$, select the arm with the highest $Q_t(a)$ (exploitation).

The exploration rate ϵ controls the trade-off.

Upper Confidence Bound (UCB)

The UCB algorithm explores uncertain arms more frequently. The UCB1 formula is:

$$\text{UCB}^t(a) = Q^t(a) + c \cdot \sqrt{\frac{\ln t}{N_t(a)}}$$

where c is the exploration coefficient. Arms with higher uncertainty are explored earlier.

Markov Decision Processes (MDPs)

State-value function:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | s_t = s]$$

Expected return starting from state s under policy π.

Action-value function:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a]$$

Expected return from state s, taking action a, then following π.

Solving MDPs with Dynamic Programming

Bellman Expectation Equation (Policy Evaluation):

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s'} P(s'|s, a)[R + \gamma V^{\pi}(s')]$$

Iteratively update value estimates using current policy.

Value Iteration:

$$V(s) = \max_{a} \sum_{s'} P(s'|s, a)[R + \gamma V(s')]$$

Optimal value found by repeatedly applying Bellman optimality.

Monte Carlo Methods

MC Value Estimation:

$$V(s) = \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_i$$

Average return over episodes starting from s.

Learn from complete episodes without model of environment.

Temporal Difference (TD) and Deep Reinforcement Learning

TD(0) Update:

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$$

Update value using immediate reward and next state's estimate.

SARSA (On-policy):

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

Learn action values while following the same policy.

Q-Learning (Off-policy):

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

Learn optimal policy regardless of the behavior policy.

DQN Loss Function:

$$L(\vartheta) = \left[r + \gamma \max_{a'} Q(s', a'; \vartheta^{-}) - Q(s, a; \vartheta) \right]^2$$

Neural network predicts Q-values; loss guides learning.

Module D

Elective 1: Applications of Generative AI

---

1. AI in Search and Recommendation Systems

Filtering Techniques:

- Content-based filtering identifies item similarity based on features.
- Collaborative filtering measures similarity between users or items.
- Hybrid filtering combines content-based and collaborative methods.

Similarity Metrics:

- Cosine similarity is calculated as: $cos(\theta) = A \cdot B / (||A|| \times ||B||)$
- Pearson correlation measures linear correlation between two variables.
- Jaccard similarity is computed as: $(A \cap B)/(A \cup B)$

Netflix & Amazon Pipelines:

- These platforms use deep learning, reinforcement learning, and hybrid recommendation systems.

Code (Python):

```
from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity([user_vector], [item_vector])
```

---

2. NLP: Sentiment Analysis & Chatbots

Key Concepts:

- Text preprocessing involves lowercasing, stopword removal, and tokenization.

- Tokenization can be performed using:
  from nltk.tokenize import word_tokenize
- Transformer models include BERT, GPT, and T5.
- Sentiment analysis classifies text as positive, negative, or neutral.
- Intent recognition is performed using pretrained models.

Chatbot Types:

- Rule-based chatbots use if-else logic.
- AI-based chatbots use transformer models like ChatGPT.

Code (HuggingFace):

```
from transformers import pipeline
sentiment_pipeline = pipeline("sentiment-analysis")
sentiment_pipeline("I love AI!")
```

---

3. AI in Computer Vision (CV)

Applications:

- In healthcare, AI detects tumors (MRI/X-ray) and diagnoses retinopathy.
- In surveillance, AI is used for face recognition and anomaly detection.

Key Models:

- CNNs like ResNet and VGG are commonly used.
- Transfer learning utilizes pretrained models.

Teachable Machine Workflow:

- *The steps are: Collect → Train → Export → Predict*

---

4. AI in Finance

Credit Risk Scoring:

- Logistic Regression, Random Forest, and XGBoost are used.
- Important features include income, age, credit score, and debt.

Fraud Detection:

- Anomaly detection uses Isolation Forest and Autoencoders.
- Supervised models include Random Forest and Logistic Regression.

Code (Python):

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier().fit(X_train, y_train)
```

---

5. AI in Agriculture

NDVI Formula:

- NDVI is calculated as: *(NIR - RED) / (NIR + RED)*

Yield Prediction Models:

- Models include Linear Regression, Random Forest, XGBoost, and LSTM.

Irrigation Planning:

- Inputs include soil moisture, weather, and crop type.
- AI models used are Decision Trees, LSTM, and Reinforcement Learning.

Tools:

- Common tools include Python, TensorFlow, and Google Earth Engine.

---

6. AI in Smart Cities

Traffic Signal Optimization:

- Optimization is done using RL algorithms (DQN, PPO) and rule-based methods.
- Sensors used include LiDAR, cameras, and GPS.

Energy Optimization:

- Load forecasting uses LSTM, ARIMA, and Prophet.

Waste Management:

- Fill-level prediction uses LSTM and ARIMA.
- Waste routing is optimized using Genetic Algorithms.

---

7. Robotics & Automation

Path Planning Algorithms:

- Classical algorithms include A* and Dijkstra.
- AI-based algorithms include DQN, PPO, and A3C.

SLAM (Simultaneous Localization and Mapping):

- SLAM involves mapping and localization together.
- Tools used are ORB-SLAM, RTAB-Map, and CNNs for feature extraction.

Real-time Control:

- Real-time control uses LSTM, DDPG, and PPO.
- CNNs are used for visual servoing.

---

8. AI in Learning & Creativity

Tools:

- Tools include GPT, DALL·E, and MusicGen.

Key Uses:

- AI powers adaptive quizzes, image generation, text creation, and music composition.

---

9. AI for Climate & Sustainability

Key Terms:

- NDVI is used to assess vegetation health.
- AI helps predict deforestation and drought.
- Google Earth Engine is a common tool for environmental analysis.

---

10. AI Bias, Fairness, and Regulation

Bias Mitigation:

- Techniques include fair ML algorithms and using balanced datasets.

Key Acts:

- Regulatory acts include the EU AI Act and India's DPDP Act.

Evaluation Metrics:

- Metrics include Disparate Impact, Equal Opportunity, and Fairness through Unawareness.

Code (Python):

```
from sklearn.metrics import classification_report
print(classification_report(y_true, y_pred))
```

Elective 2: TinyML

I. Introduction to TinyML

1. What is TinyML?

TinyML is a rapidly growing field that brings the power of machine learning to extremely resource-constrained devices such as microcontrollers. These tiny devices can run sophisticated ML models with limited computational power, memory, and energy. By processing data locally on the device, TinyML eliminates the need to send data to the cloud, thereby reducing latency and preserving privacy.

Key Characteristics

TinyML is designed to run in environments where efficiency is paramount:

- **On-device Learning**: Models are trained elsewhere but run directly on the device.
- **Low Power Consumption**: Operates using just milliwatts of energy, suitable for battery-powered systems.
- **Small Memory Footprint**: Often requires less than 256KB of RAM.
- **Cost-effective Deployment**: Enables scalable ML applications at minimal expense.

2. Why Edge Intelligence?

Edge AI is the practice of placing intelligence near the source of data. In the context of TinyML, this means enabling microcontrollers to interpret sensor data without needing to connect to the internet or a server. This offers several key advantages:

- **Low Latency**: Decisions can be made in milliseconds.
- **Improved Privacy**: Sensitive data stays on the device.
- **High Reliability**: Continues working even if offline.
- **Reduced Cost and Bandwidth**: Minimizes the need for data transmission.
- **Real-time Action**: Supports immediate response to data inputs, crucial in fields like healthcare or automation.

3. Challenges in TinyML

Despite its promise, TinyML comes with inherent limitations due to its small hardware footprint:

- **Limited Compute Power**: Constrains model complexity and size.
- **Memory Constraints**: Requires careful management of data and model architecture.
- **Energy Efficiency**: Every cycle matters in battery-powered devices.
- **Latency Requirements**: Real-time systems must respond swiftly under all conditions.

4. Core Techniques for TinyML

To make ML feasible on such constrained hardware, developers use several strategies:

- **Model Compression**: Techniques like pruning, quantization, and distillation reduce model size while preserving accuracy.

- **Efficient Inference**: Using integer arithmetic and optimized kernels increases inference speed.

- **Hardware Acceleration**: Custom microcontrollers and NPUs (Neural Processing Units) boost ML performance.

- **Software Frameworks**: Libraries like TensorFlow Lite Micro and Edge Impulse help in creating and deploying TinyML applications.

5. Applications of TinyML

TinyML is reshaping industries by enabling smart devices that act on data instantly:

- **Healthcare**: Wearable devices detect irregularities in real-time.
- **Industrial IoT**: Machines predict failures before they occur.
- **Agriculture**: Soil moisture and crop condition monitoring improve yield.
- **Smart Homes**: Devices respond to voice commands and automate routine tasks.
- **Voice Recognition**: Efficient on-device keyword spotting enables hands-free interfaces.

6. Tools and Utilities

A range of hardware and software tools support the development of TinyML applications:

- **Google Coral Dev Board Micro**: Edge AI prototyping platform.
- **TensorFlow Lite Micro**: Open-source library optimized for microcontrollers.
- **Edge Impulse Studio**: A complete pipeline for model training and deployment.
- **ARM Ethos-U55**: Dedicated hardware accelerator for inference on embedded devices.

II. Hardware for TinyML

1. Types of Processors

TinyML systems rely heavily on specialized processors that balance performance and power consumption:

- **Microcontroller Units (MCUs)**: These are simple, integrated circuits that combine a processor with memory and I/O peripherals. They are used for dedicated tasks in embedded systems.
  - *Examples*: Arduino, ESP32
- **Central Processing Units (CPUs)**: General-purpose processors found in laptops and servers. They offer high speed but consume more energy.
  - *Examples*: Intel i9, AMD Ryzen
- **Neural Processing Units (NPUs)**: Specialized chips for running AI and deep learning tasks efficiently, particularly optimized for operations like matrix multiplication.
  - *Examples*: Google TPU, Huawei Ascend

Comparison Table

Parameter	MCU	CPU	NPU
Power Consumption	Low	High	Medium
Performance	Basic	General	Specialized
Use Case	Embedded Systems	General Computing	AI Workloads

2. Memory and Speed Considerations

Efficient memory and timing are crucial:

- **RAM**: Temporarily stores data for quick access.
- **Flash Memory**: Non-volatile memory for program storage.
- **Clock Speed**: Influences how fast a processor can execute instructions. Higher speeds yield better performance but increase power usage.

3. Understanding Sensors

Sensors are the input devices for TinyML systems. They convert real-world stimuli into measurable signals.

- **Active Sensors**: Emit energy to detect the environment (e.g., LIDAR, Ultrasonic).
- **Passive Sensors**: Detect without emitting energy (e.g., LDR, Accelerometers).
- **Analog vs. Digital**: Analog gives continuous values, digital provides binary outputs.

Real-World Applications

- **Automotive**: 100+ sensors in modern cars assist in parking, collision avoidance, etc.
- **Healthcare**: Wearables monitor vital signs.
- **Smart Homes**: Automation using light, motion, and voice sensors.
- **Industrial Automation**: Monitor machinery and reduce downtime.

4. Wokwi Simulator

Wokwi is an online simulation platform for microcontroller projects, especially useful when physical hardware is unavailable.

Features

- Real-time code simulation
- Component drag-and-drop
- Circuit design and virtual wiring
- Integrated code editor and serial monitor

Getting Started with Wokwi

- Create an account at www.wokwi.com
- Start a new project or load examples.
- Drag components like LEDs or sensors.
- Write your Arduino code.
- Run the simulation and observe outputs.

III. ML Model Design for Embedded Systems

1. Lightweight Neural Networks

Convolutional Neural Networks (CNNs) are commonly used in TinyML due to their effectiveness in processing image and time-series data. A typical CNN includes:

- Convolution Layers
- Pooling Layers
- Flattening
- Fully Connected Layers (FC)

2. Transfer Learning for TinyML

Rather than training from scratch, developers often use pre-trained models and fine-tune them:

1. Choose a pre-trained model like MobileNet.
2. Remove the final classification layers.
3. Add a new output layer suited to your problem.
4. Freeze initial layers to retain learned features.
5. Fine-tune the model on your dataset.

Example Use Cases

- Image classification using ImageNet models
- NLP with BERT or GPT
- Audio processing with YAMNet or VGGish
- Edge deployment using MobileNet or SqueezeNet

3. Model Compression Techniques

Making models smaller without losing too much performance:

- **Quantization**: Converts 32-bit floats to 8-bit integers.
- **Pruning**: Removes weights or neurons with low importance.

Model Variant Comparison

Model	Type	Size	Accuracy (%)
MobileNet	Base	10.81 MB	89.64
MobileNet	Quantized	4.58 MB	81.02
MobileNet	Pruned	9.54 MB	80.96

SqueezeNet	Base	0.82 MB	88.94
SqueezeNet	Quantized	0.13 MB	91.59
SqueezeNet	Pruned	0.30 MB	91.58

4. Case Study: Human Activity Recognition

A model is trained to recognize human activities (e.g., walking, sitting) using sensor data from accelerometers and gyroscopes. The trained model is then deployed on the Arduino Nano 33 BLE Sense, showcasing real-world TinyML deployment.

IV. Model Compression — Quantization and Pruning

Model compression techniques are crucial for making ML models feasible on TinyML platforms. Two of the most effective methods are **pruning** and **quantization**.

1. Why Compress Models?

Large models are computationally expensive and require significant memory and energy, which makes them unsuitable for edge deployment. Compression helps by:

- Reducing model size
- Lowering energy consumption
- Improving response time
- Enabling deployment on devices with limited resources

2. Pruning

Pruning is the process of eliminating unnecessary parameters from the model — much like trimming branches off a tree.

Types of Pruning

- **Structured Pruning**: Removes entire filters, neurons, or layers. It simplifies the model architecture and speeds up inference.
- **Unstructured Pruning**: Removes individual low-impact weights without changing the model structure. Easier to implement but harder to optimize for real-time performance.

Example

- **Model**: LeNet-5
- **Dataset**: MNIST
- **Result**: Reduced size by 64.5% with only a 0.24% drop in accuracy

3. Quantization

Quantization reduces the precision of numbers used in a model, typically from 32-bit floating-point to 8-bit integers. This greatly improves efficiency on edge devices.

Types of Quantization

- **Post-Training Quantization (PTQ)**:
  - Quick and simple
  - Performed after model training
  - May result in accuracy loss
- **Quantization-Aware Training (QAT)**:
  - Model is trained with quantization in mind
  - Higher accuracy retention
  - Requires more resources and retraining

Example

- **Model**: LeNet-5
- **PTQ Accuracy**: 98.56%
- **QAT Accuracy**: 98.99%
- **Model Size after Quantization**: 0.07 MB (from 0.79 MB)

Tradeoffs

Technique	Model Size	Accuracy	Complexity
PTQ	Smaller	Medium	Low
QAT	Smaller	High	High

4. Key Metrics

- **Accuracy**: Prediction correctness
- **Model Size**: Memory used (in MB/KB)
- **Latency**: Time per inference
- **Memory Footprint**: RAM usage during execution

5. Other Techniques

- **Knowledge Distillation**: Transfer of learning from a large "teacher" model to a smaller "student" model.
- **Low-Rank Factorization**: Matrix decomposition to eliminate redundancy

6. Applications

- **Wearables**: Fitness tracking, voice commands
- **Smart Cameras**: Object recognition at the edge
- **Medical Devices**: Real-time diagnostics
- **Drones**: Navigation and obstacle avoidance

V. TinyML Model Deployment with Edge Impulse

1. Introduction to Edge Impulse

Edge Impulse is a cloud-based platform that simplifies the development and deployment of ML models for embedded systems. It supports data collection, model training, optimization, and deployment—all in a single workflow.

Key Features

- No-code and low-code ML environment
- Supports Python, C++, JavaScript code export
- Real-time data acquisition from edge devices
- Seamless deployment to devices like Arduino, Raspberry Pi, ESP32

2. Workflow in Edge Impulse

The development process typically follows these stages:

1) Step-by-Step Pipeline

a) **Create an Edge Impulse account** and log in

b) **Start a new project** with proper naming and sensor settings

c) **Collect Data**:

   i. Use phone IMU sensors for experiments like fall detection

   ii. Scan a QR code to connect phone sensors with Edge Impulse

d) **Label Data**:

   i. Assign labels like "Safe" or "Fall" to collected samples

   ii. Ensure diversity in the dataset (e.g., 30–40 samples per class)

e) **Split Data** into training and test sets

f) **Create Impulse**:

   i. Add signal processing block (e.g., Spectral Features)

   ii. Add learning block (e.g., Decision Tree or Neural Net Classifier)

g) **Generate Features** and visualize them

h) **Train the Model**

   i. Use default or tuned hyperparameters

i) **Model Evaluation**:

   i. Test on unseen data or live samples

j) **Deploy the Model**:

   i. Export firmware library

   ii. Deploy to physical devices or simulate

3. Benefits of Edge Impulse

- **Real-time Visualization**: View live predictions
- **Optimization**: Up to 80% reduction in model size
- **Cross-Platform**: Deploy on multiple device families
- **Hands-On Simulation**: Test without hardware using online tools

4. Example: Fall Detection

An experiment using phone IMU sensors to distinguish between "fall" and "safe" activities.

- **Goal**: Enable real-time fall detection using Edge Impulse
- **Device**: Mobile Phone + Arduino Nano BLE Sense
- **Output**: Offline ML model with high classification accuracy

---

VI. TinyML Software Frameworks and Tools

Designing and deploying TinyML models requires more than just knowledge of machine learning—it demands lightweight, efficient software that aligns with constrained hardware. The following frameworks and tools have emerged to support this effort.

TensorFlow Lite for Microcontrollers (TFLM)

TensorFlow Lite Micro is Google's answer to bringing ML inference to microcontrollers with as little as **16KB RAM**. Unlike full TensorFlow, TFLM avoids dynamic memory allocation entirely, ensuring reliability on tiny devices.

Why it's useful:

- C++ based and optimized for embedded platforms.
- Supports a wide range of operations needed for basic ML tasks.
- Widely adopted with tutorials and community support.

CMSIS-NN

Developed by ARM, CMSIS-NN is a set of efficient kernels tailored for Cortex-M processors.

Key advantages:

- Improves inference speed via hardware-level optimizations.
- Ideal for time-critical applications like gesture or motion detection.

Edge Impulse

Edge Impulse offers an end-to-end platform with **no-code and low-code interfaces**, allowing users to collect data, train models, and deploy them to devices.

What makes it powerful:

- Seamless sensor integration.
- AutoML features for beginners.
- Code export in multiple formats: C++, Arduino, and TFLite Micro.

Other Tools

- **uTensor**: Early C++ framework for TinyML, now largely replaced by TFLM.
- **Arduino IDE with TensorFlow Libraries**: Simplifies code integration for TinyML deployment.

VII. Model Deployment Pipeline on Microcontrollers

Deploying a TinyML model involves translating theory into practice on hardware with strict resource limits. The process involves several sequential steps:

1. Data Acquisition

The first step is collecting clean, relevant data using the **same sensors** that will be used during inference. This ensures the deployed model sees similar inputs as during training.

2. Feature Extraction

Depending on the problem, raw data may not be ideal. Signal processing or statistical features are often extracted either offline or live on the device.

Example: FFT features from accelerometer data for motion classification.

3. Model Training and Evaluation

Training occurs off-device, typically on a laptop or cloud. Once trained, models are validated to ensure acceptable accuracy and generalization.

4. Model Conversion and Compression

This step is crucial. The full-precision model is converted to a lightweight version using:

- `TFLiteConverter` or ONNX
- Quantization (INT8 or lower)
- Pruning (optional)

These reduce memory and power usage dramatically.

5. Deployment and Integration

The compressed model is embedded into microcontroller firmware using tools like:

- **Arduino IDE**
- **STM32CubeIDE**

Static memory allocation is planned, and the model is compiled with firmware for live testing.

6. Evaluation On Device

Finally, evaluate the model **on-device**:

- Accuracy using test data.
- Latency (in milliseconds).
- Power consumption under live conditions.

VIII. Advanced Topics in TinyML

As TinyML matures, new methods are emerging to enhance performance, personalization, and efficiency on edge devices.

Federated Learning on Edge

Federated learning trains models **locally on devices** and shares only model updates with a server. This approach:

- Enhances **privacy**
- Reduces **bandwidth**
- Ideal for healthcare and personalized assistants

On-Device Learning

Unlike traditional TinyML, where training happens in the cloud, on-device learning adapts the model in real time.

Example: A fitness tracker adapting to an individual's gait over time.

Tiny AutoML

AutoML tools automatically:

- ❖ Choose the best model architecture.
- ❖ Tune hyperparameters.
- ❖ Compress and export for deployment.

Tools: Edge Impulse AutoML, Google EdgeTPU Compiler.

Multi-Modal TinyML

Combining data from multiple sensors—such as audio + IMU + temperature—enables richer, more robust models.

Use Case: Smartwatches detecting falls + heart anomalies simultaneously.

Hardware-Aware NAS (Neural Architecture Search)

Here, machine learning is used to **design ML models** optimized for:

- ■ Speed
- ■ Size
- ■ Energy efficiency

NAS helps generate compact models tailor-made for microcontrollers.

IX. Responsible TinyML – Ethics, Privacy, and Sustainability

With great power comes great responsibility. TinyML touches the real world—often in intimate ways like healthcare or home environments. Developers must design with care.

Privacy at the Edge

Since TinyML runs on-device, **data never leaves** the hardware, boosting privacy. However, firmware updates and logging should be handled cautiously to prevent leakage.

Example: Baby cry detection systems must avoid recording sensitive audio.

Ethical Concerns

Bias, fairness, and consent must be considered:

- Is the model trained on representative data?
- Does the user **consent** to continuous monitoring?
- Can users **opt-out** or disable monitoring?

Ethics is not just about the model—it's about the **context** of its use.

Environmental Sustainability

TinyML is generally more power-efficient than cloud-based systems, making it ideal for sustainable tech.

However, challenges include:

- **E-waste** from discarded hardware.
- **Battery lifecycle management**
- Need for **green certifications** for ML systems

Best Practices

- **Design for Privacy**: Keep inference and data on-device.
- **Audit for Bias**: Use diverse datasets and review predictions.
- **Track Power**: Measure and report energy consumption.
- **Ensure Transparency**: Document limitations and performance openly.

Elective 3: Internet of Things

1. IoT Basics, Sensors & Actuators

- **Sensors** measure physical properties in the environment and convert them into electrical signals that other systems can understand. For example, a temperature sensor might detect changes in heat, while a light sensor measures brightness, and a motion sensor picks up movement.

- **Analogue Sensors:** These produce a continuous output signal that varies smoothly in proportion to the measured parameter. Example: A temperature sensor might output a voltage signal ranging continuously from 0.5V (representing 0°C) to 4.5V (representing 100°C).

- **Digital Sensors:** These output discrete, binary signals (like ON/OFF or HIGH/LOW) or digital codes (like binary numbers). Example: A Passive Infrared (PIR) motion sensor typically outputs a simple HIGH (e.g., 5V) signal when motion is detected and LOW (e.g., 0V) when no motion is present.

- **Active Sensors:** These require an external power source to operate. They typically emit some form of energy (like light or sound) into the environment and measure the reflected or modified signal. Example: A LiDAR sensor actively emits laser pulses and measures the time it takes for them to reflect back to calculate distance.

- **Passive Sensors:** These generate their output signal by utilizing ambient energy already present in the environment, without needing to emit their own energy source. Example: A thermocouple passively generates a small voltage proportional to the temperature difference between its two junctions, using the heat energy from the object being measured.

- **Actuators** perform the opposite function of sensors. They receive an electrical control signal and convert it into physical action or movement. This action allows systems to interact with the physical world.

- **Electrical Actuators**: These convert electrical energy directly into mechanical motion. Example: Electric motors in robots rotate wheels or joints based on the electrical signals they receive.

- **Hydraulic Actuators**: These use pressurized fluid (oil) to generate powerful linear or rotational force and motion. Example: Hydraulic cylinders in excavators lift heavy buckets or extend booms using the force transmitted by pressurized hydraulic fluid.

- **Pneumatic Actuators**: These use compressed air or gas to generate motion, often offering fast response times and being suitable for lighter loads. Example: Pneumatic cylinders in factory automation lines rapidly open and close valves, clamp parts, or push items along a conveyor belt using compressed air.

2. HTTP/TCP & ThingSpeak

- **HTTP Methods**
  - HTTP **(Hypertext Transfer Protocol)** defines a set of request methods to interact with web servers:
    - **GET**: Retrieves data from a server (e.g., fetching a webpage or sensor readings).
    - **POST**: Sends data to a server (e.g., submitting a form or uploading sensor data).
    - **PUT**: Updates existing data on a server (e.g., modifying stored records).
    - **DELETE**: Removes data from a server (e.g., deleting an entry in a database).

- **TCP** Features
  - TCP **(Transmission Control Protocol)** ensures reliable data transfer with the following features:
    - **Reliable**: Guarantees data delivery by retransmitting lost packets.
    - **Connection-oriented**: Establishes a stable connection before data exchange.
    - **Error-checked:** Uses checksums to detect and correct corrupted data.

- **HTTP-TCP Relationship**
  - HTTP defines the structure and format of data (e.g., requests and responses).
  - TCP handles the actual delivery of data packets reliably over the network.

- Example: ThingSpeak (an IoT platform) uses HTTP for API calls and TCP to ensure data reaches the server without errors.

- **ThingSpeak Setup**
  - To log and visualise sensor data in ThingSpeak:
  - **Create a Channel:**
    - Define fields (e.g., "Temperature," "Humidity").
  - **Use API Keys:**
    - WRITE_API_KEY allows uploading data via HTTP POST requests.
    - READ_API_KEY fetches data for analysis.
  - **Visualise Data:**
    - Use built-in charts, gauges, or graphs to display real-time sensor readings.

3. MQTT Protocol

MQTT (Message Queuing Telemetry Transport) is a lightweight publish-subscribe protocol designed for low-bandwidth IoT devices. It is 8 times more efficient than HTTP, making it ideal for constrained networks.

The key components of MQTT are:

- **Publisher**: Devices like sensors that send data to topics
- **Subscriber**: Devices like actuators that receive data from topics
- **Broker**: The central server that manages message routing
- **Topic**: Named data channels that organize messages (e.g., "home/temperature")

MQTT offers three Quality of Service (QoS) levels:

1. QoS 0 (At most once): The fastest option with no delivery guarantees
2. QoS 1 (At least once): Reliable delivery but may have duplicate messages
3. QoS 2 (Exactly once): The most reliable option with no duplicates or lost messages

HiveMQ Cloud is a popular managed MQTT broker that provides secure

communication through **TLS/SS**L encryption on **ports 8883 and 8884.** It offers scalable cloud-based messaging for IoT applications.

4. IoT Analytics

- **Data Types in IoT Systems:**
  - Structured Data: Organized and easily searchable data formats (e.g., numerical sensor readings in databases)
  - Unstructured Data: Complex formats without predefined organization (e.g., camera images, video feeds)
  - Data in Motion: Real-time streaming data from active sensors (e.g., live temperature monitoring)
  - Data at Rest: Historical data stored for analysis (e.g., archived weather patterns)
- **IoT Analytics Categories:**
  - **Descriptive Analytics:**

    Answers "What happened?"

    Example: Monthly energy consumption reports showing usage patterns
  - **Predictive Analytics:**

    Answers "What will happen?"

    Example: Machine learning models forecasting equipment failures before they occur
  - **Prescriptive Analytics:**

    Answers "What should we do?"

    Example: Smart irrigation systems recommending watering schedules based on soil conditions

- **Machine Learning Deployment Options:**
  - **Cloud ML:**
    - Handles complex processing tasks
    - Example: Using Convolutional Neural Networks (CNNs) to detect crop diseases from satellite images
  - **Edge ML:**
    - Processes data locally for faster response
    - Example: Real-time face detection on security cameras
  - **Tiny ML:**
    - Runs on low-power microcontrollers
    - Example: Wildlife monitoring devices classifying animal sounds in remote forests

5. Network Graphs in IoT

- Basic Components:
  - **Nodes**: Represent IoT devices (sensors, actuators, gateways)
  - **Edges**: Represent communication links between devices
- **Minimum Spanning Tree (MST)** in IoT Networks:
  - **Definition**: A subset of edges that connects all nodes with the minimal total connection cost
  - **Key Algorithm:** Kruskal's algorithm (time complexity O(E log E))
  - **Essential Properties:**
    - Contains no cycles (acyclic)

- Connects all nodes in the network
- Always contains exactly V-1 edges (where V = number of nodes)

- **Application:** Optimising network infrastructure costs while maintaining connectivity

- **Node Communication Conditions:**
  - Two devices (i and j) can communicate if:
  - Distance between them (d(i,j)) ≤ minimum of:
  - Transmission range of i (Tx_i) AND
  - Reception range of j (Rx_j)
  - AND simultaneously:
  - Distance between them (d(i,j)) ≤ minimum of:
  - Transmission range of j (Tx_j) AND
  - Reception range of i (Rx_i)

- **IoT Network Architecture Layers:**
  - **Edge Layer:**
    - Contains physical sensors and actuators
    - Responsible for data collection and basic actuation
  - **Fog Layer:**
    - Local processing nodes (gateways, edge servers)
    - Handles time-sensitive processing and preprocessing
  - **Cloud Layer:**

- Centralized computing infrastructure
- Performs global analytics and long-term data storage

- **Data Flow:** Raw data originates at Edge → gets processed in Fog → gets aggregated/stored in Cloud

6. Key Protocols Comparison

- **HTTP (Hypertext Transfer Protocol)**
  - Best suited for: Web services and RESTful APIs
  - Key characteristics:
    - Stateless protocol (each request independent)
    - Flexible request methods (GET, POST, PUT, DELETE)
    - Human-readable format
  - Strengths:
    - Excellent for client-server interactions
    - Wide compatibility with existing web infrastructure
    - Supports various data formats (JSON, XML, HTML)
- **TCP (Transmission Control Protocol)**
  - Best suited for: Reliable data transmission
  - Key features:
    - Guaranteed delivery with error correction
    - Flow control mechanisms
    - Connection-oriented communication

- Strengths:
  - Ensures complete, uncorrupted data transfer
  - Handles network congestion effectively
  - Maintains packet order

- **MQTT (Message Queuing Telemetry Transport)**
  - Best suited for: Low-bandwidth IoT applications
  - Key characteristics:
    - Publish-subscribe messaging model
    - Quality of Service levels (0-2)
    - Lightweight protocol overhead
  - Strengths:
    - Efficient for constrained networks
    - Supports last will and testament feature
    - Ideal for intermittent connections

- **Protocol Use Cases:**
  - **MQTT is optimal for:**
    - Large-scale sensor networks
    - Remote monitoring systems
    - Battery-powered devices
    - Example: Weather station network reporting temperature data
  - **HTTP is ideal for:**

- Web dashboard APIs
- Configuration interfaces
- System administration
- Example: Mobile app fetching device status

- **TCP is essential for:**
  - File transfers and updates
  - Firmware upgrades
  - Mission-critical communications
  - Example: Sending security patches to edge devices

- **Selection Considerations:**
- **Bandwidth:** MQTT for limited, HTTP/TCP for sufficient
- **Reliability needs:** TCP for must-have, MQTT/HTTP for nice-to-have
- **Latency requirements:** MQTT for real-time, HTTP for periodic
- **Device capabilities:** MQTT for constrained, HTTP/TCP for powerful

Elective 4: Robotics

Introduction to Robotics & Ethics

Definition and Domains

Robotics is the interdisciplinary study of programmable machines that sense, plan, and act.

Applications
• Industrial: Welding, assembly lines
• Healthcare: Surgical robots, rehabilitation
• Defense: UAVs, UGVs, surveillance
• Service: Cleaning robots, delivery bots

System Components
• Sensors: Perceive environment (e.g., IR, IMU)
• Actuators: Execute motion (e.g., DC motors)
• Controllers: Compute control inputs to guide behavior

Ethics in Robotics
• Autonomy: Maintain human oversight and fail-safes
• Privacy: Minimize surveillance/data misuse
• Transparency: Explainability in decision-making

Sensors and Actuators

Sensor Types
• Ultrasonic Sensor: Measures distance using sound echo
• IMU (Inertial Measurement Unit): Measures orientation using gy-roscopes and accelerometers
• Encoder: Measures shaft rotation/counts position
• IR Sensor: Detects proximity using infrared light

Actuator Types
• DC Motor: Provides continuous rotation
• Servo Motor: Provides angle control (PWM)
• Pneumatic Actuator: Motion via compressed air

Feedback and Calibration
• Sensor readings are compared to reference to adjust actuator behavior
• Calibration aligns raw sensor values with real-world units

Degrees of Freedom, Joints, and Coordinate Frames

Degrees of Freedom (DOF)

DOF is the number of independent variables required to uniquely define the configuration of a mechanical system.

DOF = Number of independent motion variables (translational + rotational)
For a 3D rigid body:

Max DOF = 6 = 3 (translation) + 3 (rotation)

Gruebler-Kutzbach Criterion (for planar mechanisms):

$DOF = 3(n - 1) - 2j_1 - j_2$
Where:
• n: Number of links (including ground)
• j_1: Number of lower pairs (1 DOF joints: revolute/prismatic)
• j_2: Number of higher pairs (2 DOF joints)

Gruebler-Kutzbach (spatial version):

$$\mathrm{DOF} = 6(n-1) - \sum_{i=1}^{j} f_i$$

Where:
• n: Number of links (including base)
• j: Number of joints
• f_i: DOF permitted by the ith joint

Workspace

The set of all positions a robot end-effector can reach.

Joint Types
• Revolute Joint (R): Allows rotation around an axis.
• Prismatic Joint (P): Allows linear displacement.

Coordinate Frames
• World Frame: Global fixed reference.
• Local Frame: Attached to each link/joint

Kinematics: Forward and Inverse

Denavit–Hartenberg (DH) Parameters
Each link is described using 4 parameters:

$$\theta_i,\ d_i,\ a_i,\ \alpha_i$$

$$^{i-1}T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Forward Kinematics (FK)

Maps joint space to end-effector pose using homogeneous transformations:

$$^{0}T_n = {}^{0}T_1 \cdot {}^{1}T_2 \cdots {}^{n-1}T_n$$

Basic Transformation Matrices

Translation Matrix (in 3D): Translates a point by (x, y, z)

$$T(x,y,z) = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about X-axis:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about Y-axis:

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about Z-axis:

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Homogeneous Transformation:

$$^{A}T_B = R \cdot T = \begin{bmatrix} R_{3\times3} & t_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}$$

Inverse Kinematics (IK)

Solves for joint variables given desired end-effector pose:
• Geometric: Uses trigonometry (closed-form).
• Numerical: Uses iterative methods (Jacobian-based).

Control Systems and PID Feedback

Open vs Closed Loop Control
• Open-loop: No feedback, control input is fixed.
• Closed-loop: Uses output feedback to adjust input

PID Control

PID computes the control signal as:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

- K_p: Proportional gain
- K_i: Integral gain
- K_d: Derivative gain

Tuning Methods

- Ziegler–Nichols, manual tuning, software-based autotuning

Introduction to ROS (Robot Operating System)

ROS Architecture
• Nodes: Individual processes
• Topics: Pub/Sub communication
• Services: Synchronous client-server calls

ROS Workspace
• catkin ws: Default ROS workspace
• src/: Stores all packages
• launch/: XML files for node orchestration

Package Structure
Each package contains:
• CMakeLists.txt, package.xml, src/, launch/, msg/

Mobile Robot Locomotion

Wheeled Robot Models
• Differential drive: Two independently driven wheels.

• Kinematic model:

$$\dot{x} = v \cos\theta$$
$$\dot{y} = v \sin\theta$$
$$\dot{\theta} = \omega$$

Constraints
• Holonomic: Constraints that can be expressed as functions of position.
• Non-holonomic: Velocity-based constraints, e.g., no sideways slip.

Pathfinding vs Sampling Approaches
• Pathfinding: Grid-based; optimal paths (e.g., A*)
• Sampling: Random exploration in C-space (e.g., RRT)

A* Algorithm

$$f(n) = g(n) + h(n)$$

• g(n): Cost from start to current node
• h(n): Heuristic cost to goal

Rapidly-exploring Random Trees (RRT)
• Builds a tree by sampling random configurations and connecting them to the nearest node.

Configuration Space (C-space)
Robot's valid motion space represented in terms of its DOFs; obstacles inflate in this space.

Simultaneous Localization and Mapping (SLAM)

SLAM Problem
Estimate both robot pose and map of the environment simultaneously.

Kalman Filter (EKF-SLAM)

$$\text{Prediction: } \hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k)$$

$$\text{Update: } \hat{x}_{k|k} = \hat{x}_{k|k-1} + K(z_k - h(\hat{x}_{k|k-1}))$$

Particle Filter SLAM
• Uses weighted particles to represent pose probability distribution.
• Incorporates resampling to focus on likely states.

Sensor Fusion
Combines data from LiDAR, odometry, IMU for robust localization.

Learning in Robotics + Human-Robot Interaction (HRI)

Reinforcement Learning

Q-Learning Update Rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

- α: Learning rate
- γ: Discount factor

Policy Learning

Learn a mapping $\pi : S \rightarrow A$ that maximizes long-term reward.

Human-Robot Interaction (HRI)

- **Interfaces:** Gesture control (e.g., OpenCV), voice commands
- **Ethical HRI:** Safety, trust calibration, social compliance

Elective 5: Mechanics

Role of Mechanics in Robotics

• Motion: Determined by force $\vec{F}$ and torque $\vec{\tau}$.
• Control: Uses models from mechanics to compute actuation.
• Stability: Linked to force balance and dynamic response.

Physics-Based Modeling

• Free body diagrams show all external forces.
• Use differential equations from Newton's laws.
• Kinetic + potential energy ⇒ Lagrangian methods.

Physical Quantities

• Scalar: No direction (e.g., m = 2 kg).
• Vector: Represented as $\vec{A} = A_x \hat{i} + A_y \hat{j} + A_z \hat{k}$.
• Derived quantities: Velocity

Newton's Laws of Motion

- **First Law (Law of Inertia):**
  An object remains at rest or moves with constant velocity unless acted upon by a net external force.

- **Second Law:**
  The net force on an object is equal to the product of its mass and acceleration: F=ma.

- **Third Law:**
  For every action, there is an equal and opposite reaction

Vectors, Coordinate Frames & Kinematics

Position and Displacement

• Position: $\vec{r} = x\hat{i} + y\hat{j} + z\hat{k}$.
• Displacement: $\Delta\vec{r} = \vec{r}_f - \vec{r}_i$.
• Distance is scalar, displacement is vector.
Vector Algebra • $\vec{A} + \vec{B} = (A_x + B_x)\hat{i} + (A_y + B_y)\hat{j} + (A_z + B_z)\hat{k}$.
• Dot: $\vec{A} \cdot \vec{B} = A_xB_x + A_yB_y + A_zB_z$.

Cross: $\vec{A} \times \vec{B} = \det \begin{bmatrix} \hat{i} & \hat{j} & \hat{k} \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{bmatrix}$.

Rotation Matrices

- $R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$.
- $R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$.
- $R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$.
- Properties: $R^{-1} = R^\top$, $\det(R) = +1$.

Homogeneous Coordinates

- 2D point: $\vec{p} = \begin{bmatrix} x \\ y \end{bmatrix} \Rightarrow \vec{p}_{\text{hom}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$.
- 3D point: $\vec{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow \vec{p}_{\text{hom}} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$.

Linear and Angular Motion

Linear Motion

Position: The location of a particle in space at time t, typically relative to a reference frame. ⃗r(t) = x(t)ˆi + y(t)ˆj + z(t) ˆk

Velocity: The rate of change of position with respect to time; describes how fast and in what direction a particle moves.

$$\vec{v}(t) = \frac{d\vec{r}(t)}{dt}$$

Acceleration: The rate of change of velocity with respect to time; indicates how velocity changes over time.

$$\vec{a}(t) = \frac{d\vec{v}(t)}{dt} = \frac{d^2\vec{r}(t)}{dt^2}$$

Equations of motion (constant acceleration): Describe linear kinematics under uniform acceleration.

$$\vec{v}(t) = \vec{v}_0 + \vec{a}t$$

$$\vec{r}(t) = \vec{r}_0 + \vec{v}_0 t + \frac{1}{2}\vec{a}t^2$$

$$\vec{v}^2 = \vec{v}_0^2 + 2\vec{a} \cdot (\vec{r} - \vec{r}_0)$$

Angular Motion

Angular Position: The angle θ(t) that an object has rotated through, relative to a fixed axis. θ(t) ∈ R, measured in radians

Angular Velocity: The rate of change of angular position with respect to time. ω(t) = dθ(t)/dt , ω = ωn

Angular Acceleration: The rate of change of angular velocity with respect to time.

$$\alpha(t) = \frac{d\omega(t)}{dt}, \quad \vec{\alpha} = \frac{d\vec{\omega}}{dt}$$

Equations of angular motion (constant angular acceleration):

$$\omega(t) = \omega_0 + \alpha t$$

$$\theta(t) = \theta_0 + \omega_0 t + \frac{1}{2}\alpha t^2$$

$$\omega^2 = \omega_0^2 + 2\alpha(\theta - \theta_0)$$

Relation Between Linear and Angular Quantities

Linear displacement, velocity, and acceleration can be related to angular motion for rotational systems.

$$s = r\theta$$
$$v = r\omega$$
$$a = r\alpha$$

Torque and Rotational Motion

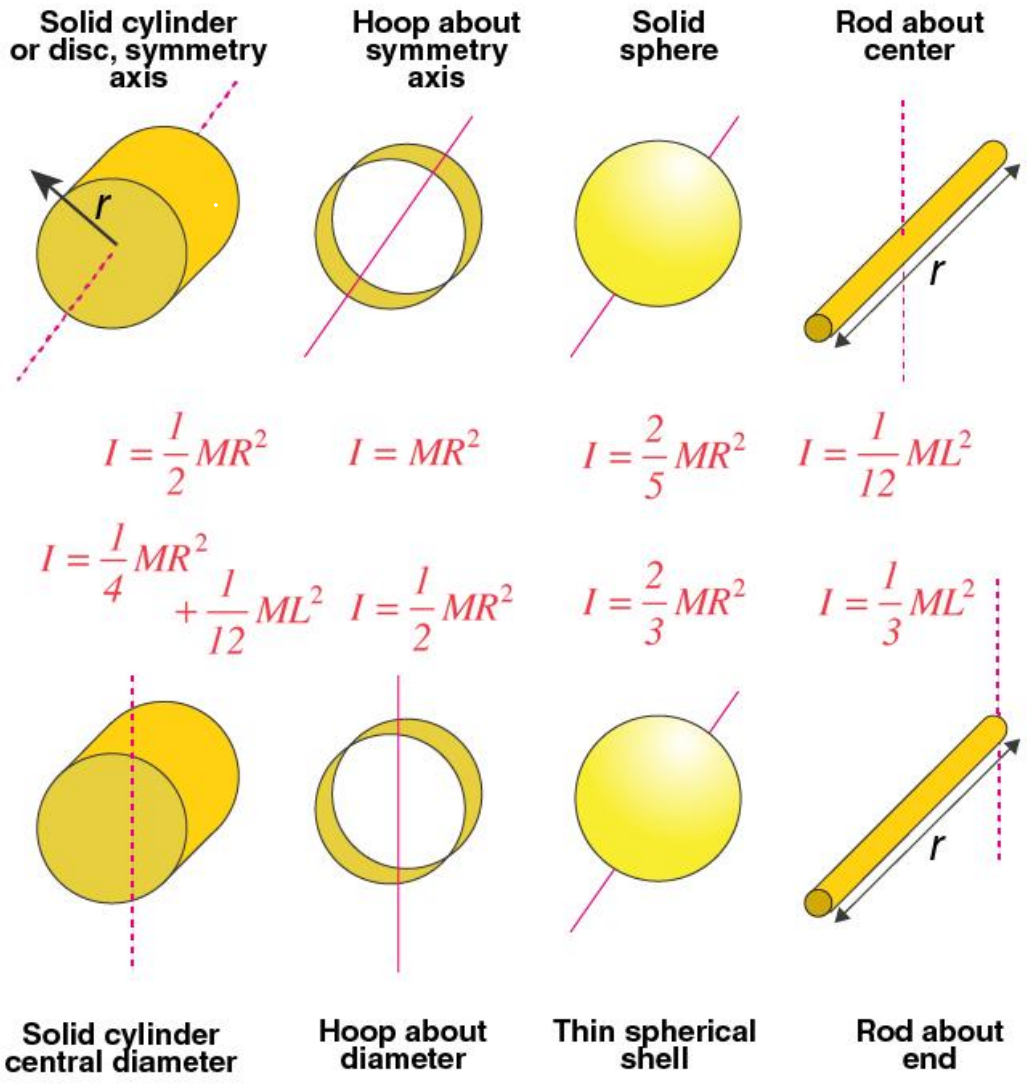
Torque: A measure of the tendency of a force to rotate an object about an axis.

$$\vec{\tau} = \vec{r} \times \vec{F}$$

Rotational Newton's 2nd Law: Angular version of F = ma for rotating bodies.

$$\vec{\tau} = I\,\vec{\alpha}$$

Moment of Inertia of Standard Shapes



Statics and Center of Mass

Static Equilibrium and Stability

Static Equilibrium: A system is in static equilibrium when net force and net torque on it are zero.

$$\sum \vec{F} = 0, \sum \tau = 0$$

Conditions for Stability: An object is stable if its center of mass lies within its base of support under disturbance.

Free-Body Diagrams (FBD)

Free-body Diagram: A diagram showing all external forces and torques acting on a body.

Force Balance: Newton's First Law applied to static bodies.

$$\sum Fx = 0, \sum Fy = 0, \sum Fz = 0$$

Moment Balance: Sum of torques about any point must be zero in equilibrium. $\sum \tau = \sum (ri \times Fi) = 0$

Energy, Work, and Power

Kinetic and Potential Energy

Kinetic Energy (Translational): Energy due to motion of mass m at velocity

$$KE = \frac{1}{2}mv^2$$

v.

Kinetic Energy (Rotational): Energy due to rotation about an axis with angular velocity ω

$$KE_{\text{rot}} = \frac{1}{2}I\omega^2$$

Potential Energy (Gravitational): Energy due to position in a gravitational field.

P E = mgh

Work-Energy Theorem: Net work done equals the change in kinetic energy.

W_net = ΔKE

Power

Instantaneous Power: Rate at which work is done at any instant.

$$P = \frac{dW}{dt} = \vec{F} \cdot \vec{v}$$

Rotational Power: Power delivered by torque and angular velocity.

$$P = \tau \cdot \omega$$

Average Power: Total work done divided by time interval.

$$\bar{P} = \frac{W}{\Delta t}$$

Friction, Contact, and Compliance

Types of Friction

Static Friction: Resists motion up to a maximum value.

$$f_s \leq \mu_s N$$

Kinetic (Sliding) Friction: Constant resistive force once motion starts.

$$f_k = \mu_k N$$

Rolling Friction: Resistance due to rolling contact; generally much smaller.

$$f_r = \mu_r N$$

Contact Forces and Reaction Modeling

Contact Force: Modeled using normal and tangential components during contact.

$$\vec{F}_{\text{contact}} = F_n \hat{n} + F_t \hat{t}$$

Hooke's Law and Spring Compliance

Hooke's Law: Force exerted by a linear spring is proportional to displacement.

$$F = -kx$$

Rotational Spring: Torque is proportional to angular displacement.

$$\tau = -k_\theta \theta$$

Damping Models

Viscous Damping: Force proportional to velocity; smooth, continuous damping.

$$F_d = -bv$$

Rotational Damping: Torque opposing angular motion.

$$\tau_d = -b_\theta \omega$$

Coulomb Damping: Constant frictional force opposing motion direction.

$$F_{\text{coulomb}} = -\mu N \cdot \text{sgn}(v)$$

Compliance in Robotics

Compliance: The inverse of stiffness; describes how easily a system deforms under force.

$$C = \frac{1}{k}, \quad \text{for linear springs}$$

Application: Used in robotic joints, grippers, and end-effectors to absorb impacts and adapt to surfaces.