Semester 2 Handbook

Minor in AI

Contents

Module C Handbook

1 Introduction to Natural Language Processing

Natural Language Processing (NLP) is a branch of artificial intelligence that focuses on the interaction between computers and human language. It involves developing algorithms and models that can understand, interpret, and generate human language in a valuable way.

Key Terms & Definitions:

- **Natural Language Processing (NLP)**: Interdisciplinary field combining computational linguistics, machine learning, and artificial intelligence to enable computers to understand human language
- **Computational Linguistics**: The scientific study of language from a computational perspective
- **Language Understanding**: The process of extracting meaning from text or speech
- **Language Generation**: The process of producing human-like text or speech from structured data

Key Points:

- NLP bridges the gap between human communication and computer understanding
- It combines rule-based approaches with statistical and machine learning methods
- Applications span from simple text processing to complex dialogue systems
- Success depends on understanding both linguistic theory and computational methods

1.1 Core Components of NLP

- **Syntax**: The arrangement of words and phrases to create well-formed sentences
- **Semantics**: The meaning conveyed by text or speech
- **Pragmatics**: The context-dependent interpretation of language
- **Discourse**: The structure of connected text beyond individual sentences

Key Terms & Definitions:

- **Syntax**: Rules governing word order and sentence structure (e.g., Subject-Verb-Object in English)
- **Semantics**: Study of meaning in language, including word meanings and compositional semantics
- **Pragmatics**: Context-dependent aspects of meaning, including speaker intent

and situational context

- **Discourse**: Organization of language units larger than sentences, including coherence and cohesion
- **Morphology**: Study of word internal structure and formation
- **Phonology**: Sound system of language (relevant for speech processing)

2 Tokenization

Tokenization is the process of breaking down text into smaller units called tokens, which can be words, sentences, or subword units.

Key Terms & Definitions:

- **Token**: A meaningful unit of text (word, punctuation, or subword)
- **Vocabulary**: The set of all unique tokens in a corpus
- **Out-of-Vocabulary (OOV)**: Words not present in the training vocabulary
- **Delimiter**: Character(s) used to separate tokens (spaces, punctuation)

2.1 Types of Tokenization

2.1.1 Word Tokenization

Splits text into individual words based on whitespace and punctuation.

Input: ”Hello world, how are you?” (1)

Output: [”Hello”,”world”,”,”,”how”,”are”,”you”,”?”] (2)

Practical Pointers:

- Use regular expressions for simple tokenization: `re.split(r’\s+’, text)`
- Consider language-specific rules (German compounds, Chinese word segmentation)
- Handle contractions carefully: ”don’t” → [”do”, ”n’t”] or [”don’t”]
- Preserve important punctuation while removing noise

2.1.2 Sentence Tokenization

Divides text into sentences using punctuation marks as delimiters.

Input: ”Hello world. How are you? I’m fine.” (3)

Output: [”Hello world.”,”How are you?”,”I’m fine.”] (4)

2.1.3 Subword Tokenization

Breaks words into smaller meaningful units to handle out-of-vocabulary words.

$$\text{Input: "unhappiness"} \tag{5}$$

$$\text{Output: ["un", "happy", "ness"]} \tag{6}$$

Key Terms & Definitions:

- **Byte Pair Encoding (BPE)**: Algorithm that merges most frequent character pairs iteratively
- **WordPiece**: Google's tokenization method used in BERT
- **SentencePiece**: Language-independent subword tokenizer
- **Unigram Language Model**: Probabilistic approach to subword segmentation

2.2 Challenges in Tokenization

- **Punctuation handling**: Deciding whether punctuation is part of a word
- **Abbreviations**: Handling contractions like "don't", "I'm"
- **Multilingual text**: Different languages have different tokenization rules
- **Special characters**: URLs, emails, hashtags require special handling

Practical Pointers:

- Use NLTK's word_tokenize() for English text
- For multilingual text, consider spaCy or Moses tokenizer
- Normalize text before tokenization (Unicode, case, accents)
- Preserve tokenization consistency between training and inference

3 N-Grams

An **N-gram** is a contiguous sequence of n items from a given sample of text or speech.

Key Terms & Definitions:

- **N-gram**: Contiguous sequence of n items (characters, words, or phonemes)
- **Context Window**: The span of preceding words considered for prediction
- **Markov Assumption**: Future states depend only on current state, not full history
- **Language Model**: Statistical model that assigns probabilities to sequences of

words

3.1 Mathematical Definition

For a sequence of words $w_1, w_2, \ldots, w_m$, an n-gram is defined as:

$$\text{n-gram} = (w_i, w_{i+1}, \ldots, w_{i+n-1}) \tag{7}$$

3.2 N-gram Probability

The probability of a word given previous n-1 words:

$$P(w_i|w_{i-n+1}, \ldots, w_{i-1}) = \frac{C(w_{i-n+1}, \ldots, w_i)}{C(w_{i-n+1}, \ldots, w_{i-1})} \tag{8}$$

where $C(\cdot)$ represents the count function.

3.3 Types of N-Grams

- **Unigram (1-gram)**: Single words
- **Bigram (2-gram)**: Pairs of consecutive words
- **Trigram (3-gram)**: Triplets of consecutive words
- **4-gram and higher**: Longer sequences for more context

3.4 Example

For the sentence ”The cat sat on the mat”:

$$\text{Unigrams: } [\text{"The"}, \text{"cat"}, \text{"sat"}, \text{"on"}, \text{"the"}, \text{"mat"}] \tag{9}$$

$$\text{Bigrams: } [\text{"The cat"}, \text{"cat sat"}, \text{"sat on"}, \text{"on the"}, \text{"the mat"}] \tag{10}$$

$$\text{Trigrams: } [\text{"The cat sat"}, \text{"cat sat on"}, \text{"sat on the"}, \text{"on the mat"}] \tag{11}$$

Key Formulae:

- **Chain Rule**: $P(w_1^n) = \prod_{i=1}^{n} P(w_i|w_1^{i-1})$
- **Bigram Approximation**: $P(w_i|w_1^{i-1}) \approx P(w_i|w_{i-1})$
- **Trigram Approximation**: $P(w_i|w_1^{i-1}) \approx P(w_i|w_{i-2}, w_{i-1})$

3.5 Smoothing Techniques

3.5.1 Laplace Smoothing (Add-1)

$$P_{\text{Laplace}}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + |V|} \tag{12}$$

3.5.2 Add-k Smoothing

$$P_{\text{Add-k}}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + k}{C(w_{i-1}) + k|V|} \tag{13}$$

3.5.3 Good-Turing Smoothing

$$C^* = (C+1)\frac{N_{C+1}}{N_C} \tag{14}$$

where N_C is the number of n-grams occurring exactly C times.

4 Markov Models and Hidden Markov Models

Markov Models assume that future states depend only on the current state, not on the entire history.

Key Terms & Definitions:

- **Markov Property**: Future state depends only on current state
- **State Space**: Set of all possible states
- **Transition Probability**: Probability of moving from one state to another
- **Stationary Distribution**: Long-term probability distribution over states
- **Hidden Markov Model (HMM)**: Markov model with unobservable states
- **Emission Probability**: Probability of observing symbol given hidden state

4.1 Markov Chains

4.1.1 First-Order Markov Chain

The Markov property states:

$$P(X_{t+1} = s_j | X_t = s_i, X_{t-1} = s_{t-1}, \ldots, X_1 = s_1) = P(X_{t+1} = s_j | X_t = s_i) \tag{15}$$

4.1.2 Transition Matrix

For N states, the transition matrix $\mathbf{A}$ is:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix} \tag{16}$$

where $a_{ij} = P(X_{t+1} = s_j | X_t = s_i)$ and $\sum_{j=1}^{N} a_{ij} = 1$

4.1.3 Chapman-Kolmogorov Equation

For n-step transitions:

$$P(X_{t+n} = s_j | X_t = s_i) = (\mathbf{A}^n)_{ij} \tag{17}$$

4.1.4 Stationary Distribution

The stationary distribution $\boldsymbol{\pi}$ satisfies:

$$\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{A} \tag{18}$$

This is the left eigenvector of $\mathbf{A}$ with eigenvalue 1.

Key Formulae:

- **Forward Equation**: $\boldsymbol{\pi}_{t+1} = \boldsymbol{\pi}_t\mathbf{A}$
- **Ergodic Theorem**: $\lim_{n\to\infty} \frac{1}{n}\sum_{t=1}^{n} \nVdash[X_t = s_i] = \pi_i$
- **First Passage Time**: $T_j = \min\{t > 0 : X_t = j | X_0 = i\}$

4.2 Hidden Markov Models (HMMs)

HMMs extend Markov models by separating hidden states from observable emissions.

4.2.1 HMM Components

An HMM is defined by the tuple $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$:

- **State transition matrix**: $\mathbf{A} = \{a_{ij}\}$ where $a_{ij} = P(q_{t+1} = S_j | q_t = S_i)$
- **Emission matrix**: $\mathbf{B} = \{b_j(k)\}$ where $b_j(k) = P(O_t = v_k | q_t = S_j)$
- **Initial state distribution**: $\boldsymbol{\pi} = \{\pi_i\}$ where $\pi_i = P(q_1 = S_i)$

4.2.2 Three Fundamental Problems

Problem 1: Evaluation Given HMM λ and observation sequence $\mathbf{O} = O_1O_2\dots O_T$, compute $P(\mathbf{O}|\lambda)$.
Forward Algorithm:

$$\alpha_t(i) = P(O_1O_2\dots O_t, q_t = S_i|\lambda) \tag{19}$$

$$\alpha_1(i) = \pi_i b_i(O_1) \tag{20}$$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i)a_{ij}\right] b_j(O_{t+1}) \tag{21}$$

$$P(\mathbf{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{22}$$

Problem 2: Decoding Find the most likely state sequence given observations.
Viterbi Algorithm:

$$\delta_1(i) = \pi_i b_i(O_1) \tag{23}$$

$$\delta_t(j) = \max_{1\le i\le N}[\delta_{t-1}(i)a_{ij}]b_j(O_t) \tag{24}$$

$$\psi_t(j) = \arg\max_{1\le i\le N}[\delta_{t-1}(i)a_{ij}] \tag{25}$$

Backtracking:

$$q_T^* = \arg\max_{1\leq i\leq N} \delta_T(i) \tag{26}$$

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \tag{27}$$

Problem 3: Learning Estimate HMM parameters given observation sequences.
Baum-Welch Algorithm (EM for HMMs):
E-step - Compute forward and backward variables:

$$\beta_t(i) = P(O_{t+1}O_{t+2}\dots O_T|q_t = S_i, \lambda) \tag{28}$$

$$\beta_T(i) = 1 \tag{29}$$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}b_j(O_{t+1})\beta_{t+1}(j) \tag{30}$$

Compute $\gamma_t(i)$ and $\xi_t(i,j)$:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N}\alpha_t(j)\beta_t(j)} \tag{31}$$

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(\mathbf{O}|\lambda)} \tag{32}$$

M-step - Update parameters:

$$\bar{\pi}_i = \gamma_1(i) \tag{33}$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\gamma_t(i)} \tag{34}$$

$$\bar{b}_j(k) = \frac{\sum_{t=1,O_t=v_k}^{T}\gamma_t(j)}{\sum_{t=1}^{T}\gamma_t(j)} \tag{35}$$

Key Formulae:

- **Forward Variable**: $\alpha_t(i) = P(O_1 \dots O_t, q_t = i|\lambda)$
- **Backward Variable**: $\beta_t(i) = P(O_{t+1} \dots O_T|q_t = i, \lambda)$
- **State Posterior**: $\gamma_t(i) = P(q_t = i|\mathbf{O}, \lambda)$
- **Transition Posterior**: $\xi_t(i,j) = P(q_t = i, q_{t+1} = j|\mathbf{O}, \lambda)$

4.3 Applications in NLP

4.3.1 Part-of-Speech Tagging

- **Hidden States**: POS tags (NN, VB, JJ, etc.)
- **Observations**: Words in the sentence
- **Transition Probabilities**: $P(\text{tag}_i|\text{tag}_{i-1})$
- **Emission Probabilities**: $P(\text{word}_i|\text{tag}_i)$

4.3.2 Speech Recognition

- **Hidden States**: Phonemes or sub-phoneme units
- **Observations**: Acoustic feature vectors (MFCC)
- **Left-to-Right Models**: States can only transition forward

4.3.3 Named Entity Recognition

Use BIO tagging scheme:

- **Hidden States**: B-PER, I-PER, B-LOC, I-LOC, B-ORG, I-ORG, O
- **Observations**: Words and their features
- **Constraints**: I-tags can only follow B-tags of same type

Key Points:

- HMMs provide structured approach to sequence modeling
- Forward-Backward algorithm efficiently computes marginals
- Viterbi algorithm finds optimal state sequence
- EM algorithm (Baum-Welch) learns parameters from data
- Widely used in traditional NLP before neural approaches

Practical Pointers:

- Initialize parameters carefully to avoid local minima in EM
- Use smoothing for transition and emission probabilities
- Consider higher-order Markov models for better context
- Implement in log-space to avoid numerical underflow
- Modern neural approaches often outperform HMMs for most tasks

5 One-Hot Encoding

One-hot encoding is a method to represent categorical data as binary vectors.

Key Terms & Definitions:

- **One-hot Vector**: Binary vector with single 1 and rest 0s
- **Sparse Representation**: Vector with mostly zero values
- **Categorical Variable**: Variable with discrete, unordered categories

- **Embedding Dimension**: Size of the vector representation

5.1 Mathematical Definition

For a vocabulary $V = \{w_1, w_2, \ldots, w_n\}$, the one-hot encoding of word w_i is:

$$\mathbf{e}_i = [0, 0, \ldots, \underbrace{1}_{i\text{-th position}}, \ldots, 0]^T \in \mathbb{R}^{|V|} \tag{36}$$

5.2 Properties

- **Orthogonality**: $\mathbf{e}_i^T \mathbf{e}_j = \delta_{ij}$ (Kronecker delta)
- **Sparsity**: Only one element is 1, rest are 0
- **Dimensionality**: Vector size equals vocabulary size

Key Formulae:

- **Orthogonality**: $\mathbf{e}_i^T \mathbf{e}_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$
- **L2 Norm**: $||\mathbf{e}_i||_2 = 1$ for all i
- **Dot Product**: $\mathbf{e}_i^T \mathbf{e}_j = 0$ for $i \neq j$ (no similarity)

5.3 Limitations

- High dimensionality for large vocabularies
- No semantic relationships captured
- Curse of dimensionality in high-dimensional spaces
- Memory inefficient for large vocabularies

Practical Pointers:

- Use for small vocabularies or as input to embedding layers
- Consider alternatives like word embeddings for better semantic representation
- Implement efficiently using sparse matrices
- Good baseline for simple classification tasks

6 Word Embeddings

Word embeddings are dense vector representations of words that capture semantic relationships in continuous vector space.

Key Terms & Definitions:

- **Word Embedding**: Dense vector representation of words in continuous space
- **Distributional Hypothesis**: Words with similar contexts have similar meanings
- **Context Window**: Number of surrounding words considered
- **Skip-gram**: Predicting context words from center word
- **CBOW**: Continuous Bag of Words, predicting center word from context
- **Negative Sampling**: Efficient approximation to hierarchical softmax
- **Subsampling**: Randomly removing frequent words to balance training

6.1 Word2Vec

Word2Vec learns word representations by predicting word co-occurrences in large text corpora.

6.1.1 Skip-gram Model

Predicts context words given a center word:

$$P(w_{t+j}|w_t) = \frac{\exp(\mathbf{v}_{w_{t+j}}^T \mathbf{u}_{w_t})}{\sum_{w=1}^{|V|} \exp(\mathbf{v}_w^T \mathbf{u}_{w_t})} \tag{37}$$

Objective function:

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log P(w_{t+j}|w_t) \tag{38}$$

6.1.2 Continuous Bag of Words (CBOW)

Predicts center word from context words:

$$P(w_t|w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c}) = \frac{\exp(\mathbf{v}_{w_t}^T \bar{\mathbf{h}})}{\sum_{w=1}^{|V|} \exp(\mathbf{v}_w^T \bar{\mathbf{h}})} \tag{39}$$

where $\bar{\mathbf{h}} = \frac{1}{2c} \sum_{-c \leq j \leq c, j \neq 0} \mathbf{u}_{w_{t+j}}$

6.1.3 Negative Sampling

Approximates softmax using negative examples:

$$\log \sigma(\mathbf{v}_{w_O}^T \mathbf{u}_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-\mathbf{v}_{w_i}^T \mathbf{u}_{w_I})] \tag{40}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ and $P_n(w) = \frac{f(w)^{3/4}}{\sum_{w'} f(w')^{3/4}}$

Key Formulae:

- **Skip-gram Objective**: $\mathcal{J} = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j}\log P(w_{t+j}|w_t)$
- **CBOW Objective**: $\mathcal{J} = -\frac{1}{T}\sum_{t=1}^{T}\log P(w_t|\text{context}(w_t))$
- **Negative Sampling**: $\mathcal{J} = -\log\sigma(\mathbf{u}_{w_c}^T\mathbf{v}_{w_t}) - \sum_{i=1}^{k}\log\sigma(-\mathbf{u}_{w_i}^T\mathbf{v}_{w_t})$

6.2 GloVe (Global Vectors)

GloVe combines global matrix factorization with local context window methods.

6.2.1 Co-occurrence Matrix

Build matrix X where X_{ij} represents how often word i appears in context of word j:

$$X_{ij} = \sum_{k=1}^{|C|} \nVdash[w_i \text{ in context of } w_j \text{ in corpus position } k] \tag{41}$$

6.2.2 GloVe Objective

$$\mathcal{J} = \sum_{i,j=1}^{|V|} f(X_{ij})(\mathbf{w}_i^T\tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij})^2 \tag{42}$$

where the weighting function is:

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases} \tag{43}$$

Typical values: $x_{\max} = 100, \alpha = 3/4$

6.3 FastText

FastText extends Word2Vec by incorporating subword information through character n-grams.

6.3.1 Subword Representation

Each word w is represented as bag of character n-grams:

$$\mathbf{v}_w = \sum_{g\in\mathcal{G}_w} \mathbf{z}_g \tag{44}$$

where $\mathcal{G}_w$ is the set of n-grams for word w and $\mathbf{z}_g$ is the vector for n-gram g.

6.3.2 Character N-grams

For word ”where” with n-grams of size 3-6:

$$\begin{aligned} \text{3-grams:} \quad & \langle \text{wh, whe, her, ere, re} \rangle && (45) \\ \text{4-grams:} \quad & \langle \text{whe, wher, here, ere} \rangle && (46) \\ \text{5-grams:} \quad & \langle \text{wher, where, here} \rangle && (47) \\ \text{6-grams:} \quad & \langle \text{where} \rangle && (48) \end{aligned}$$

Key Points:

- Word embeddings capture semantic and syntactic relationships
- Skip-gram works better for infrequent words, CBOW for frequent words
- GloVe leverages global co-occurrence statistics
- FastText handles out-of-vocabulary words through subword modeling
- Pre-trained embeddings provide good initialization for downstream tasks

6.4 Evaluation of Word Embeddings

6.4.1 Intrinsic Evaluation

- **Word Similarity**: Correlation with human similarity judgments
- **Word Analogy**: Solving analogy tasks like ”king - man + woman = queen”
- **Clustering**: Quality of word clusters in embedding space

6.4.2 Analogy Task

Solve $\mathbf{a} - \mathbf{b} + \mathbf{c} = \mathbf{d}$ by finding:

$$\mathbf{d}^* = \arg \max_{\mathbf{d} \in V \setminus \{\mathbf{a},\mathbf{b},\mathbf{c}\}} \cos(\mathbf{d}, \mathbf{a} - \mathbf{b} + \mathbf{c}) \tag{49}$$

6.4.3 Extrinsic Evaluation

Evaluate embeddings on downstream tasks:

- Named Entity Recognition
- Sentiment Analysis
- Part-of-Speech Tagging
- Text Classification

Practical Pointers:

- Use pre-trained embeddings (Word2Vec, GloVe, FastText) as starting point
- Fine-tune embeddings during training if sufficient data available
- Consider domain-specific embeddings for specialized tasks
- FastText is best for morphologically rich languages
- Normalize embeddings for cosine similarity computations

7 Bag of Words (BoW)

The **Bag of Words** model represents text as a collection of words, disregarding grammar and word order.

Key Terms & Definitions:

- **Bag of Words**: Document representation as word frequency vector
- **Document-Term Matrix**: Matrix with documents as rows, terms as columns
- **Frequency Vector**: Vector containing word occurrence counts
- **Binary BoW**: Uses 1/0 for presence/absence instead of counts

7.1 Mathematical Formulation

For a document d and vocabulary $V = \{w_1, w_2, \dots, w_n\}$, the BoW representation is:

$$\mathbf{x}_d = [c_1, c_2, \dots, c_n]^T \tag{50}$$

where c_i is the count of word w_i in document d.

7.2 Example

Documents:

$$d_1 : \text{"The cat sat"} \tag{51}$$

$$d_2 : \text{"The cat ran"} \tag{52}$$

Vocabulary: $V = \{\text{The, cat, sat, ran}\}$
BoW representations:

$$\mathbf{x}_{d_1} = [1, 1, 1, 0]^T \tag{53}$$

$$\mathbf{x}_{d_2} = [1, 1, 0, 1]^T \tag{54}$$

Key Points:

- Simple and computationally efficient
- Works well for document classification and clustering
- Loses word order and grammatical information
- Suitable for topic modeling and information retrieval

7.3 Variants

7.3.1 Binary BoW

$$\mathbf{x}_{d,\text{binary}} = [\nVdash(c_1 > 0), \nVdash(c_2 > 0), \ldots, \nVdash(c_n > 0)]^T \tag{55}$$

7.3.2 Normalized BoW

$$\mathbf{x}_{d,\text{norm}} = \frac{\mathbf{x}_d}{||\mathbf{x}_d||_2} \tag{56}$$

8 TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF is a numerical statistic that reflects the importance of a word in a document relative to a collection of documents.

Key Terms & Definitions:

- **Term Frequency (TF)**: How often a term appears in a document
- **Inverse Document Frequency (IDF)**: Measure of term's rarity across corpus
- **Corpus**: Collection of documents
- **Stop Words**: Common words with low discriminative power

8.1 Mathematical Definition

8.1.1 Term Frequency (TF)

$$\text{TF}(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d} \tag{57}$$

8.1.2 Inverse Document Frequency (IDF)

$$\text{IDF}(t, D) = \log\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right) \tag{58}$$

where $|D|$ is the total number of documents and $|\{d \in D : t \in d\}|$ is the number of documents containing term t.

8.1.3 TF-IDF Score

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D) \tag{59}$$

Key Formulae:

- **Raw Term Frequency**: $\text{TF}_{\text{raw}}(t, d) = f_{t,d}$ (raw count)
- **Boolean TF**: $\text{TF}_{\text{bool}}(t, d) = \begin{cases} 1 & \text{if } f_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$
- **Log-normalized TF**: $\text{TF}_{\log}(t, d) = 1 + \log(f_{t,d})$ if $f_{t,d} > 0$, else 0

8.2 Enhancements

8.2.1 Sublinear TF Scaling

$$\text{TF}_{\text{sublinear}}(t, d) = 1 + \log(\text{TF}(t, d)) \tag{60}$$

8.2.2 Smoothing

$$\text{IDF}_{\text{smooth}}(t, D) = \log\left(\frac{|D|}{1 + |\{d \in D : t \in d\}|}\right) + 1 \tag{61}$$

8.2.3 Normalization

L2 normalization of TF-IDF vectors:

$$\mathbf{x}_{\text{norm}} = \frac{\mathbf{x}}{||\mathbf{x}||_2} \tag{62}$$

Practical Pointers:

- Use TF-IDF for document similarity and information retrieval
- Apply sublinear TF scaling to reduce impact of very frequent terms
- Combine with stop word removal for better performance
- Consider using scikit-learn's TfidfVectorizer with appropriate parameters

9 Part-of-Speech (POS) Tagging

POS Tagging is the process of assigning grammatical categories to words in a sentence.

Key Terms & Definitions:

- **Part-of-Speech**: Grammatical category of a word (noun, verb, adjective)
- **Tag Set**: Predefined set of grammatical categories
- **Ambiguity**: Words that can have multiple POS tags depending on context

- **Sequence Labeling**: Assigning labels to sequences of input elements

9.1 Common POS Tags

- **NN**: Noun (singular)
- **NNS**: Noun (plural)
- **VB**: Verb (base form)
- **VBD**: Verb (past tense)
- **VBG**: Verb (gerund/present participle)
- **VBN**: Verb (past participle)
- **VBP**: Verb (present, not 3rd person singular)
- **VBZ**: Verb (present, 3rd person singular)
- **JJ**: Adjective
- **JJR**: Adjective (comparative)
- **JJS**: Adjective (superlative)
- **RB**: Adverb
- **RBR**: Adverb (comparative)
- **RBS**: Adverb (superlative)
- **DT**: Determiner
- **IN**: Preposition
- **CC**: Coordinating conjunction
- **PRP**: Personal pronoun
- **WP**: Wh-pronoun

9.2 Mathematical Model

POS tagging can be modeled as a sequence labeling problem:

$$\hat{y}_1^n = \arg\max_{y_1^n} P(y_1^n|x_1^n) \tag{63}$$

where x_1^n is the word sequence and y_1^n is the POS tag sequence.

9.2.1 Hidden Markov Model (HMM) for POS Tagging

$$P(y_1^n, x_1^n) = \prod_{i=1}^{n} P(y_i|y_{i-1})P(x_i|y_i) \tag{64}$$

$$P(\text{tag}|\text{previous tag}) = \text{Transition Probability} \tag{65}$$

$$P(\text{word}|\text{tag}) = \text{Emission Probability} \tag{66}$$

Key Formulae:

- **Viterbi Algorithm**: $\delta_t(j) = \max_{1\leq i\leq N}[\delta_{t-1}(i) \cdot a_{ij}] \cdot b_j(o_t)$
- **Forward Algorithm**: $\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) \cdot a_{ij} \cdot b_j(o_t)$
- **Backward Algorithm**: $\beta_t(i) = \sum_{j=1}^{N} a_{ij} \cdot b_j(o_{t+1}) \cdot \beta_{t+1}(j)$

Practical Pointers:

- Use NLTK's pos_tag() for quick English POS tagging
- Consider spaCy for more accurate and faster tagging
- For domain-specific text, train custom taggers
- Use POS tags as features for downstream tasks

10 Named Entity Recognition (NER)

Named Entity Recognition identifies and classifies named entities in text into predefined categories.

Key Terms & Definitions:

- **Named Entity**: Real-world objects with proper names (people, places, organizations)
- **Entity Mention**: Textual reference to a named entity
- **Entity Linking**: Connecting mentions to knowledge base entries
- **Coreference Resolution**: Identifying when different expressions refer to same entity

10.1 Common Entity Types

- **PERSON**: Names of people (John Smith, Einstein)
- **ORGANIZATION**: Companies, institutions (Google, MIT)

- **LOCATION**: Cities, countries, landmarks (Paris, USA, Mount Everest)
- **DATE**: Temporal expressions (January 1st, 2023)
- **TIME**: Time expressions (3:30 PM, noon)
- **MONEY**: Monetary values ($100, 50 euros)
- **PERCENT**: Percentages (25%, half)
- **GPE**: Geopolitical entities (countries, states, cities)
- **FACILITY**: Buildings, airports, highways
- **PRODUCT**: Objects, vehicles, foods
- **EVENT**: Named hurricanes, battles, wars
- **WORK_OF_ART**: Titles of books, songs
- **LAW**: Named documents made into laws
- **LANGUAGE**: Any named language

10.2 BIO Tagging Scheme

- **B-**: Beginning of an entity
- **I-**: Inside an entity (continuation)
- **O**: Outside any entity

10.2.1 BILOU Tagging Scheme

Extended version with more precision:

- **B-**: Beginning of entity
- **I-**: Inside entity
- **L-**: Last token of multi-token entity
- **O-**: Outside any entity
- **U-**: Unit-length entity (single token)

10.3 Example

John Smith works at Google in Mountain View.
BIO Tags: `B-PERSON I-PERSON O O B-ORG O B-LOC I-LOC`

Key Formulae:

- **Conditional Random Field (CRF)**: $P(y|x) = \frac{1}{Z(x)} \exp\left(\sum_{i,k} \lambda_k f_k(y_i, y_{i-1}, x, i)\right)$
- **BiLSTM-CRF**: Combines bidirectional LSTM with CRF layer

- **F1 Score**: $F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

Practical Pointers:

- Use spaCy's built-in NER for general purposes
- Fine-tune BERT-based models for domain-specific NER
- Consider fuzzy matching for handling variations in entity names
- Evaluate using entity-level F1 score, not token-level

11 Stop Word Removal

Stop words are common words that carry little semantic meaning and are often removed during preprocessing.

Key Terms & Definitions:

- **Stop Words**: High-frequency words with low semantic content
- **Function Words**: Grammatical words (articles, prepositions, conjunctions)
- **Content Words**: Words carrying semantic meaning (nouns, verbs, adjectives)
- **Stop List**: Predefined list of stop words for a language

11.1 Examples of Stop Words

the, is, at, which, on, a, an, and, or, but, in, with, to, for, of, as, by, be, been, have, has, had, do, does, did, will, would, could, should, may, might, can, must, shall, this, that, these, those, i, you, he, she, it, we, they, me, him, her, us, them

11.2 Mathematical Impact

Removing stop words reduces vocabulary size:

$$|V'| = |V| - |S| \tag{67}$$

where S is the set of stop words.

11.3 Impact on Document Representation

Original TF-IDF vector:

$$\mathbf{x}_{\text{original}} = [\text{TF-IDF}(w_1), \text{TF-IDF}(w_2), \ldots, \text{TF-IDF}(w_{|V|})] \tag{68}$$

After stop word removal:

$$\mathbf{x}_{\text{filtered}} = [\text{TF-IDF}(w_i) : w_i \notin S] \tag{69}$$

Key Points:

- Reduces dimensionality and computational complexity
- May improve performance for some tasks (information retrieval, classification)
- Can hurt performance for tasks requiring full context (sentiment analysis, translation)
- Language-dependent: different stop lists for different languages

Practical Pointers:

- Use NLTK's stop word lists: `from nltk.corpus import stopwords`
- Create domain-specific stop word lists
- Consider frequency-based stop word detection
- Be cautious with negation words in sentiment analysis

12 Stemming and Lemmatization

12.1 Stemming

Stemming reduces words to their root form by removing suffixes.

Key Terms & Definitions:

- **Stemming**: Crude heuristic process to chop off word endings
- **Stem**: Root form of word after removing affixes
- **Over-stemming**: Removing too much, creating non-words
- **Under-stemming**: Removing too little, keeping distinct stems

12.1.1 Porter Stemmer Rules

$$\text{running} \to \text{run} \tag{70}$$
$$\text{flies} \to \text{fli} \tag{71}$$
$$\text{dogs} \to \text{dog} \tag{72}$$
$$\text{studies} \to \text{studi} \tag{73}$$
$$\text{ponies} \to \text{poni} \tag{74}$$

12.1.2 Porter Stemmer Algorithm Steps

1. **Step 1a**: Remove plurals (sses → ss, ies → i, ss → ss, s →)

2. **Step 1b**: Remove past tense (eed → ee, ed → , ing →)
3. **Step 2**: Remove derivational suffixes (ational → ate, izer → ize)
4. **Step 3**: Remove additional suffixes (icate → ic, ative →)
5. **Step 4**: Remove final suffixes (al → , ance → , er →)
6. **Step 5**: Remove final e and double consonants

Key Formulae:

- **Measure Function**: m = number of VC patterns (V=vowel, C=consonant)
- **Condition**: Apply rule only if $m > k$ for some threshold k
- **Vowel in Stem**: *v* - stem contains a vowel
- **Ends with Double Consonant**: *d - stem ends with double consonant

12.2 Lemmatization

Lemmatization reduces words to their dictionary form (lemma) considering morphological analysis.

Key Terms & Definitions:

- **Lemmatization**: Morphological analysis to find dictionary form
- **Lemma**: Canonical/dictionary form of word
- **Inflection**: Grammatical variation (tense, number, person)
- **Derivation**: Formation of new words by adding affixes
- **Morphology**: Study of word structure and formation

12.2.1 Examples

$$\text{running} \rightarrow \text{run} \tag{75}$$

$$\text{mice} \rightarrow \text{mouse} \tag{76}$$

$$\text{went} \rightarrow \text{go} \tag{77}$$

$$\text{children} \rightarrow \text{child} \tag{78}$$

$$\text{feet} \rightarrow \text{foot} \tag{79}$$

$$\text{geese} \rightarrow \text{goose} \tag{80}$$

$$\text{am/is/are} \rightarrow \text{be} \tag{81}$$

Key Points:

- Stemming is faster but less accurate than lemmatization
- Lemmatization preserves meaning better but requires POS information
- Choice depends on application requirements and computational constraints
- Both reduce vocabulary size and handle morphological variations

Practical Pointers:

- Use NLTK's PorterStemmer for basic stemming
- Use spaCy or NLTK's WordNetLemmatizer for lemmatization
- Consider language-specific stemmers (Snowball stemmers)
- Provide POS tags to lemmatizer for better accuracy

12.3 Comparison Table

Word	**Stemming**	**Lemmatization**
running	run	run
flies	fli	fly
studies	studi	study
went	went	go
mice	mice	mouse

Table 1: Stemming vs Lemmatization Comparison

13 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks are designed to handle sequential data by maintaining hidden states that capture information from previous time steps.

Key Terms & Definitions:

- **Recurrent Neural Network (RNN)**: Neural network designed for sequential data
- **Hidden State**: Internal memory that carries information across time steps
- **Temporal Dependencies**: Relationships between elements at different time steps
- **Parameter Sharing**: Same weights used across all time steps
- **Unfolding**: Expanding RNN through time for visualization/computation

13.1 RNN Architecture

13.1.1 Mathematical Formulation

At time step t, the RNN computes:

$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}_h) \tag{82}$$

$$\mathbf{y}_t = \mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y \tag{83}$$

where:

- $\mathbf{x}_t \in \mathbb{R}^d$: Input at time t
- $\mathbf{h}_t \in \mathbb{R}^h$: Hidden state at time t
- $\mathbf{y}_t \in \mathbb{R}^o$: Output at time t
- $\mathbf{W}_h \in \mathbb{R}^{h \times h}$: Hidden-to-hidden weight matrix
- $\mathbf{W}_x \in \mathbb{R}^{h \times d}$: Input-to-hidden weight matrix
- $\mathbf{W}_y \in \mathbb{R}^{o \times h}$: Hidden-to-output weight matrix
- $\mathbf{b}_h, \mathbf{b}_y$: Bias vectors

Key Formulae:

- **RNN Update**: $\mathbf{h}_t = f(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b})$
- **Output**: $\mathbf{y}_t = g(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y)$
- **Initial State**: $\mathbf{h}_0 = \mathbf{0}$ or learned parameter
- **Activation**: $f = \tanh, \text{ReLU}, \text{or sigmoid}$

13.2 RNN Variants

13.2.1 Many-to-One (Sequence Classification)

$$\mathbf{y} = \text{softmax}(\mathbf{W}_y \mathbf{h}_T + \mathbf{b}_y) \tag{84}$$

13.2.2 One-to-Many (Sequence Generation)

$$\mathbf{y}_t = \text{softmax}(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y) \tag{85}$$

13.2.3 Many-to-Many (Sequence-to-Sequence)

$$\text{Encoder: } \mathbf{h}_t^{enc} = f(\mathbf{W}^{enc} \mathbf{h}_{t-1}^{enc} + \mathbf{U}^{enc} \mathbf{x}_t) \tag{86}$$

$$\text{Decoder: } \mathbf{h}_t^{dec} = f(\mathbf{W}^{dec} \mathbf{h}_{t-1}^{dec} + \mathbf{U}^{dec} \mathbf{y}_{t-1}) \tag{87}$$

13.3 Backpropagation Through Time (BPTT)

The gradient of the loss with respect to parameters is computed using BPTT:

$$\frac{\partial L}{\partial \mathbf{W}_h} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial \mathbf{W}_h} \tag{88}$$

13.3.1 Gradient Flow

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{i=k+1}^{t} \mathbf{W}_h^T \text{diag}(\tanh'(\mathbf{z}_i)) \tag{89}$$

Key Formulae:

- **Chain Rule:** $\frac{\partial L}{\partial \mathbf{W}_h} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_h}$
- **Temporal Gradient:** $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_s} = \prod_{i=s+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}}$
- **Truncated BPTT**: Limit backpropagation to k time steps: $\frac{\partial L_t}{\partial \mathbf{W}} \approx \sum_{i=\max(1,t-k+1)}^{t} \frac{\partial L_t}{\partial \mathbf{h}_i} \frac{\partial \mathbf{h}_i}{\partial \mathbf{W}}$

13.4 Vanishing and Exploding Gradients

13.4.1 Vanishing Gradients

When $||\mathbf{W}_h|| < 1$ and $|\tanh'| < 1$:

$$\left|\left|\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}\right|\right| \leq ||\mathbf{W}_h||^{t-k} \to 0 \text{ as } (t-k) \to \infty \tag{90}$$

13.4.2 Exploding Gradients

When $||\mathbf{W}_h|| > 1$:

$$\left|\left|\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}\right|\right| \geq ||\mathbf{W}_h||^{t-k} \to \infty \text{ as } (t-k) \to \infty \tag{91}$$

13.4.3 Gradient Clipping

To prevent exploding gradients:

$$\mathbf{g} \leftarrow \begin{cases} \mathbf{g} & \text{if } ||\mathbf{g}|| \leq \theta \\ \frac{\theta \mathbf{g}}{||\mathbf{g}||} & \text{otherwise} \end{cases} \tag{92}$$

Practical Pointers:

- Initialize weights carefully (Xavier or He initialization)
- Use gradient clipping to prevent exploding gradients
- Consider skip connections or residual connections

- Use more sophisticated architectures (LSTM, GRU) for long sequences

14 Long Short-Term Memory (LSTM)

LSTM networks address the vanishing gradient problem through gating mechanisms that control information flow.

Key Terms & Definitions:

- **LSTM**: Long Short-Term Memory network with gating mechanisms
- **Cell State**: Long-term memory component that flows through network
- **Gate**: Neural mechanism that controls information flow
- **Forget Gate**: Decides what information to discard from cell state
- **Input Gate**: Controls what new information to store in cell state
- **Output Gate**: Controls what parts of cell state to output

14.1 LSTM Architecture

14.1.1 Cell State and Gates

The LSTM maintains a cell state $\mathbf{C}_t$ and uses three gates:

Forget Gate Decides what information to throw away from the cell state:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \tag{93}$$

Input Gate Decides what new information to store in the cell state:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \tag{94}$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C) \tag{95}$$

Output Gate Controls what parts of the cell state to output:

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \tag{96}$$

14.1.2 State Updates

$$\mathbf{C}_t = \mathbf{f}_t * \mathbf{C}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{C}}_t \tag{97}$$

$$\mathbf{h}_t = \mathbf{o}_t * \tanh(\mathbf{C}_t) \tag{98}$$

where $*$ denotes element-wise multiplication and σ is the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{99}$$

Key Formulae:

- **Sigmoid Gate**: $\sigma(x) = \frac{1}{1+e^{-x}}$ (outputs values between 0 and 1)
- **Tanh Activation**: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ (outputs values between -1 and 1)
- **Element-wise Multiplication**: $\mathbf{a} * \mathbf{b} = [a_1 b_1, a_2 b_2, \ldots, a_n b_n]$
- **Concatenation**: $[\mathbf{a}, \mathbf{b}] = [a_1, \ldots, a_m, b_1, \ldots, b_n]$

14.2 LSTM Information Flow

14.2.1 Step-by-Step Process

1. **Forget**: $\mathbf{f}_t$ determines what to forget from $\mathbf{C}_{t-1}$
2. **Input**: $\mathbf{i}_t$ and $\tilde{\mathbf{C}}_t$ determine what new info to store
3. **Update**: Combine old and new information in $\mathbf{C}_t$
4. **Output**: $\mathbf{o}_t$ determines what to output based on $\mathbf{C}_t$

14.3 LSTM Variants

14.3.1 Peephole Connections

Gates can also look at the cell state:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot [\mathbf{C}_{t-1}, \mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \tag{100}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \cdot [\mathbf{C}_{t-1}, \mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \tag{101}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \cdot [\mathbf{C}_t, \mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \tag{102}$$

14.3.2 Coupled Forget and Input Gates

Simplification where input gate is complement of forget gate:

$$\mathbf{i}_t = 1 - \mathbf{f}_t \tag{103}$$

14.3.3 Layer Normalization LSTM

Apply layer normalization to improve training:

$$\mathbf{h}_t = \text{LayerNorm}(\mathbf{o}_t * \tanh(\text{LayerNorm}(\mathbf{C}_t))) \tag{104}$$

Key Points:

- LSTM solves vanishing gradient problem through additive cell state updates
- Three gates (forget, input, output) control information flow
- Cell state acts as a ”conveyor belt” carrying information across time

- More parameters than vanilla RNN but better long-term memory

15 Gated Recurrent Unit (GRU)

GRU is a simplified version of LSTM with fewer parameters and computational requirements.

Key Terms & Definitions:

- **GRU**: Gated Recurrent Unit, simplified LSTM variant
- **Reset Gate**: Controls how much past information to forget
- **Update Gate**: Controls how much new information to accept
- **Candidate State**: Proposed new hidden state

15.1 GRU Architecture

15.1.1 Reset Gate

Controls how much of the past state to use when computing candidate state:

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_r) \tag{105}$$

15.1.2 Update Gate

Controls how much of the past state to keep and how much of candidate state to accept:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_z) \tag{106}$$

15.1.3 Candidate Activation

Proposed new hidden state:

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W} \cdot [\mathbf{r}_t * \mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}) \tag{107}$$

15.1.4 Hidden State Update

Final hidden state as interpolation between past and candidate:

$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * \tilde{\mathbf{h}}_t \tag{108}$$

Key Formulae:

- **Reset Gate**: $\mathbf{r}_t = \sigma(\mathbf{W}_r[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_r)$
- **Update Gate**: $\mathbf{z}_t = \sigma(\mathbf{W}_z[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_z)$
- **Linear Interpolation**: $\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$

15.2 GRU vs LSTM Comparison

Aspect	**LSTM**	**GRU**
Gates	3 (forget, input, output)	2 (reset, update)
Parameters	More	Fewer (25% less)
Memory	Separate cell state	Single hidden state
Computation	Higher	Lower
Performance	Better on complex tasks	Comparable on simpler tasks
Training Speed	Slower	Faster
Long-term Dependencies	Excellent	Good
Overfitting	More prone	Less prone

Table 2: Detailed LSTM vs GRU Comparison

Practical Pointers:

- Start with GRU for faster experimentation
- Use LSTM for tasks requiring longer memory
- GRU often performs similarly to LSTM with less computation
- Consider ensemble methods combining both architectures

16 Bidirectional RNNs

Bidirectional RNNs process sequences in both forward and backward directions to capture complete context.

Key Terms & Definitions:

- **Bidirectional RNN**: RNN processing sequence in both directions
- **Forward Pass**: Left-to-right processing
- **Backward Pass**: Right-to-left processing
- **Context Vector**: Combined representation from both directions

16.1 Mathematical Formulation

16.1.1 Forward Pass

$$\overrightarrow{\mathbf{h}}_t = f(\mathbf{W}_{\overrightarrow{h}}\overrightarrow{\mathbf{h}}_{t-1} + \mathbf{W}_{\overrightarrow{x}}\mathbf{x}_t + \mathbf{b}_{\overrightarrow{h}}) \tag{109}$$

16.1.2 Backward Pass

$$\overleftarrow{\mathbf{h}}_t = f(\mathbf{W}_{\overleftarrow{h}}\overleftarrow{\mathbf{h}}_{t+1} + \mathbf{W}_{\overleftarrow{x}}\mathbf{x}_t + \mathbf{b}_{\overleftarrow{h}}) \tag{110}$$

16.1.3 Output Combination Methods

$$\mathbf{h}_t^{\text{concat}} = [\overrightarrow{\mathbf{h}}_t; \overleftarrow{\mathbf{h}}_t] \tag{111}$$

$$\mathbf{h}_t^{\text{sum}} = \overrightarrow{\mathbf{h}}_t + \overleftarrow{\mathbf{h}}_t \tag{112}$$

$$\mathbf{h}_t^{\text{avg}} = \frac{1}{2}(\overrightarrow{\mathbf{h}}_t + \overleftarrow{\mathbf{h}}_t) \tag{113}$$

$$\mathbf{h}_t^{\text{max}} = \max(\overrightarrow{\mathbf{h}}_t, \overleftarrow{\mathbf{h}}_t) \tag{114}$$

Key Formulae:

- **Forward Hidden State**: $\overrightarrow{\mathbf{h}}_t = f(\mathbf{W}_f \overrightarrow{\mathbf{h}}_{t-1} + \mathbf{U}_f \mathbf{x}_t)$
- **Backward Hidden State**: $\overleftarrow{\mathbf{h}}_t = f(\mathbf{W}_b \overleftarrow{\mathbf{h}}_{t+1} + \mathbf{U}_b \mathbf{x}_t)$
- **Final Representation**: $\mathbf{h}_t = g(\overrightarrow{\mathbf{h}}_t, \overleftarrow{\mathbf{h}}_t)$

16.2 Applications and Benefits

Key Points:

- Provides complete context for each position in sequence
- Particularly useful for sequence labeling (POS tagging, NER)
- Doubles the number of parameters and computation time
- Cannot be used for online/streaming applications
- Excellent for tasks where future context is important

17 Deep RNNs

Deep RNNs stack multiple RNN layers to increase model capacity and representational power.

Key Terms & Definitions:

- **Deep RNN**: Multi-layer RNN architecture
- **Vertical Stacking**: Layers stacked on top of each other
- **Layer Depth**: Number of RNN layers in the network
- **Representational Capacity**: Ability to model complex patterns

17.1 Architecture

For an L-layer deep RNN:

$$\mathbf{h}_t^{(1)} = f(\mathbf{W}_h^{(1)}\mathbf{h}_{t-1}^{(1)} + \mathbf{W}_x^{(1)}\mathbf{x}_t + \mathbf{b}^{(1)}) \tag{115}$$

$$\mathbf{h}_t^{(l)} = f(\mathbf{W}_h^{(l)}\mathbf{h}_{t-1}^{(l)} + \mathbf{W}_x^{(l)}\mathbf{h}_t^{(l-1)} + \mathbf{b}^{(l)}) \quad \text{for } l = 2, \ldots, L \tag{116}$$

17.2 Information Flow

- **Temporal flow**: Information flows through time within each layer
- **Depth flow**: Information flows vertically between layers

17.2.1 Deep Bidirectional RNN

Combining depth and bidirectionality:

$$\overrightarrow{\mathbf{h}}_t^{(l)} = f(\mathbf{W}_f^{(l)}\overrightarrow{\mathbf{h}}_{t-1}^{(l)} + \mathbf{U}_f^{(l)}\mathbf{h}_t^{(l-1)}) \tag{117}$$

$$\overleftarrow{\mathbf{h}}_t^{(l)} = f(\mathbf{W}_b^{(l)}\overleftarrow{\mathbf{h}}_{t+1}^{(l)} + \mathbf{U}_b^{(l)}\mathbf{h}_t^{(l-1)}) \tag{118}$$

$$\mathbf{h}_t^{(l)} = [\overrightarrow{\mathbf{h}}_t^{(l)}; \overleftarrow{\mathbf{h}}_t^{(l)}] \tag{119}$$

Practical Pointers:

- Start with 2-3 layers; more layers don't always help
- Apply dropout between layers to prevent overfitting
- Consider residual connections for very deep networks
- Monitor gradient flow in deep architectures

18 Attention Mechanism

Attention allows models to focus on relevant parts of the input sequence when generating outputs.

Key Terms & Definitions:

- **Attention Mechanism**: Method to focus on relevant input parts
- **Query**: The current state asking for information
- **Key**: The states being queried against
- **Value**: The actual information to be retrieved
- **Attention Weights**: Probabilities indicating relevance
- **Context Vector**: Weighted combination of input representations

- **Alignment**: How well query matches with keys

18.1 Basic Attention

18.1.1 Attention Score

Measures compatibility between query and key:

$$e_{ij} = a(\mathbf{s}_{i-1}, \mathbf{h}_j) \tag{120}$$

where $\mathbf{s}_{i-1}$ is the decoder state (query) and $\mathbf{h}_j$ is the encoder hidden state (key).

18.1.2 Attention Weights

Normalized attention scores using softmax:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})} \tag{121}$$

18.1.3 Context Vector

Weighted sum of encoder states:

$$\mathbf{c}_i = \sum_{j=1}^{T} \alpha_{ij} \mathbf{h}_j \tag{122}$$

Key Formulae:

- **Attention Score**: $e_{ij} = \text{score}(\mathbf{q}_i, \mathbf{k}_j)$
- **Softmax Normalization**: $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})}$
- **Context Vector**: $\mathbf{c}_i = \sum_{j=1}^{T} \alpha_{ij} \mathbf{v}_j$
- **Output**: $\mathbf{o}_i = f(\mathbf{s}_i, \mathbf{c}_i)$

18.2 Attention Functions

18.2.1 Additive Attention (Bahdanau)

$$a(\mathbf{s}, \mathbf{h}) = \mathbf{v}^T \tanh(\mathbf{W}_s \mathbf{s} + \mathbf{W}_h \mathbf{h}) \tag{123}$$

18.2.2 Multiplicative Attention (Luong)

$$a(\mathbf{s}, \mathbf{h}) = \mathbf{s}^T \mathbf{W} \mathbf{h} \tag{124}$$

18.2.3 Scaled Dot-Product Attention

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{125}$$

18.2.4 Multi-Head Attention

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O \tag{126}$$

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \tag{127}$$

Key Points:

- Attention solves the bottleneck problem of encoder-decoder architectures
- Provides interpretability by showing what the model focuses on
- Different attention mechanisms suit different tasks
- Foundation for Transformer architecture

18.3 Self-Attention

When queries, keys, and values come from the same sequence:

$$\mathbf{Q} = \mathbf{K} = \mathbf{V} = \mathbf{X}\mathbf{W} \tag{128}$$

Practical Pointers:

- Use attention visualization for model interpretability
- Consider attention dropout during training
- Scale dot-product attention by $\sqrt{d_k}$ to prevent vanishing gradients
- Multi-head attention captures different types of relationships

19 Machine Translation and Encoder-Decoder Architecture

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

Key Terms & Definitions:

- **Machine Translation (MT)**: Automatic translation between languages
- **Encoder-Decoder**: Architecture with separate encoding and decoding components
- **Source Language**: The input language being translated from
- **Target Language**: The output language being translated to
- **Parallel Corpus**: Aligned sentence pairs in different languages
- **Beam Search**: Decoding algorithm that maintains top-k hypotheses

- **Teacher Forcing**: Training technique using ground truth as decoder input

19.1 Encoder-Decoder Framework

19.1.1 Encoder

The encoder processes the source sequence and produces context representations:

$$\mathbf{c} = q(\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{T_x}\}) \tag{129}$$

Common encoding strategies:

$$\mathbf{c}_{\text{last}} = \mathbf{h}_{T_x} \tag{130}$$

$$\mathbf{c}_{\text{mean}} = \frac{1}{T_x} \sum_{t=1}^{T_x} \mathbf{h}_t \tag{131}$$

$$\mathbf{c}_{\text{max}} = \max_{t=1}^{T_x} \mathbf{h}_t \tag{132}$$

19.1.2 Decoder

The decoder generates the target sequence autoregressively:

$$P(\mathbf{y}) = \prod_{t=1}^{T_y} P(y_t | \{y_1, \ldots, y_{t-1}\}, \mathbf{c}) \tag{133}$$

19.1.3 Decoder with Attention

$$\mathbf{s}_t = f(\mathbf{s}_{t-1}, y_{t-1}, \mathbf{c}_t) \tag{134}$$

$$\mathbf{c}_t = \sum_{j=1}^{T_x} \alpha_{tj} \mathbf{h}_j \tag{135}$$

$$P(y_t | y_{<t}, \mathbf{x}) = g(\mathbf{s}_t, y_{t-1}, \mathbf{c}_t) \tag{136}$$

19.2 Training and Inference

19.2.1 Training Objective

Maximum Likelihood Estimation:

$$\mathcal{L}(\theta) = -\sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \log P(\mathbf{y} | \mathbf{x}; \theta) \tag{137}$$

19.2.2 Teacher Forcing

During training, use ground truth target tokens as input:

$$P(y_t | y_{<t}, \mathbf{x}) \approx P(y_t | y^*_{<t}, \mathbf{x}) \tag{138}$$

where $y^*_{<t}$ are the ground truth tokens.

19.2.3 Beam Search Decoding

Maintain top-k hypotheses at each step:

$$\text{Score}(\mathbf{y}) = \log P(\mathbf{y}|\mathbf{x}) \tag{139}$$

$$= \sum_{t=1}^{T} \log P(y_t|y_{<t}, \mathbf{x}) \tag{140}$$

Length normalization:

$$\text{Score}_{\text{norm}}(\mathbf{y}) = \frac{\log P(\mathbf{y}|\mathbf{x})}{|\mathbf{y}|^{\alpha}} \tag{141}$$

Key Formulae:

- **Cross-Entropy Loss**: $\mathcal{L} = -\sum_{t=1}^{T} \sum_{v=1}^{|V|} y_{t,v} \log \hat{y}_{t,v}$
- **Perplexity**: $\text{PPL} = \exp\left(-\frac{1}{T} \sum_{t=1}^{T} \log P(y_t|y_{<t}, \mathbf{x})\right)$
- **Beam Score**: $\text{score}(\mathbf{y}) = \frac{1}{|\mathbf{y}|^{\alpha}} \sum_{t=1}^{|\mathbf{y}|} \log P(y_t|y_{<t}, \mathbf{x})$

19.3 Challenges in Machine Translation

Key Points:

- Handling rare and out-of-vocabulary words
- Preserving meaning across languages with different structures
- Managing long-distance dependencies
- Dealing with ambiguity and context-dependent translations
- Maintaining fluency while preserving adequacy

Practical Pointers:

- Use byte-pair encoding (BPE) for handling OOV words
- Apply data augmentation (back-translation, paraphrasing)
- Use pre-trained multilingual models when possible
- Implement proper evaluation using multiple metrics

20 Autoregressive Language Models

Autoregressive models predict the next token in a sequence based on all previous tokens, factorizing the joint probability using the chain rule.

Key Terms & Definitions:

- **Autoregressive Model**: Model that predicts next element using previous elements
- **Chain Rule of Probability**: Decomposition of joint probability into conditional probabilities
- **Causal Modeling**: Future predictions depend only on past and present, not future
- **Left-to-Right Generation**: Sequential generation from beginning to end
- **Teacher Forcing**: Using ground truth as input during training
- **Exposure Bias**: Discrepancy between training and inference conditions

20.1 Mathematical Foundation

20.1.1 Chain Rule Decomposition

The joint probability of a sequence $\mathbf{x} = (x_1, x_2, \ldots, x_T)$ is:

$$P(\mathbf{x}) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \cdots P(x_T|x_1, \ldots, x_{T-1}) \tag{142}$$

General form:

$$P(\mathbf{x}) = \prod_{t=1}^{T} P(x_t|x_{<t}) \tag{143}$$

where $x_{<t} = (x_1, x_2, \ldots, x_{t-1})$ represents all tokens before position t.

20.1.2 Autoregressive Modeling Objective

Maximum likelihood training minimizes negative log-likelihood:

$$\mathcal{L}(\theta) = -\sum_{(\mathbf{x})\in\mathcal{D}} \log P(\mathbf{x}; \theta) = -\sum_{(\mathbf{x})\in\mathcal{D}} \sum_{t=1}^{T} \log P(x_t|x_{<t}; \theta) \tag{144}$$

20.1.3 Neural Autoregressive Models

Use neural networks to parameterize conditional distributions:

$$\mathbf{h}_t = f_\theta(x_1, x_2, \ldots, x_{t-1}) \tag{145}$$

$$P(x_t|x_{<t}; \theta) = \text{softmax}(\mathbf{W}\mathbf{h}_t + \mathbf{b})_{x_t} \tag{146}$$

Key Formulae:

- **Conditional Probability**: $P(x_t|x_{<t}) = \frac{\exp(f_\theta(x_{<t})_{x_t})}{\sum_{v\in\mathcal{V}} \exp(f_\theta(x_{<t})_v)}$
- **Cross-Entropy Loss**: $\mathcal{L}_t = -\log P(x_t|x_{<t}; \theta)$

- **Perplexity**: $\text{PPL} = \exp\left(\frac{1}{T}\sum_{t=1}^{T}\mathcal{L}_t\right)$

20.2 RNN-based Autoregressive Models

20.2.1 Vanilla RNN Language Model

$$\mathbf{h}_t = \tanh(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{W}_x\mathbf{e}_{x_t} + \mathbf{b}_h) \tag{147}$$

$$\mathbf{y}_t = \text{softmax}(\mathbf{W}_y\mathbf{h}_t + \mathbf{b}_y) \tag{148}$$

$$P(x_{t+1}|x_{\leq t}) = y_{t,x_{t+1}} \tag{149}$$

20.2.2 LSTM Language Model

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{e}_{x_t}] + \mathbf{b}_f) \tag{150}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{e}_{x_t}] + \mathbf{b}_i) \tag{151}$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_C[\mathbf{h}_{t-1}, \mathbf{e}_{x_t}] + \mathbf{b}_C) \tag{152}$$

$$\mathbf{C}_t = \mathbf{f}_t * \mathbf{C}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{C}}_t \tag{153}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{e}_{x_t}] + \mathbf{b}_o) \tag{154}$$

$$\mathbf{h}_t = \mathbf{o}_t * \tanh(\mathbf{C}_t) \tag{155}$$

20.3 Training Strategies

20.3.1 Teacher Forcing

During training, use ground truth tokens as input:

$$\mathcal{L} = -\sum_{t=1}^{T}\log P(x_t^*|x_1^*, x_2^*, \ldots, x_{t-1}^*; \theta) \tag{156}$$

where x_t^* are ground truth tokens.

20.3.2 Scheduled Sampling

Gradually transition from teacher forcing to model predictions:

$$x_{t-1}^{\text{input}} = \begin{cases} x_{t-1}^* & \text{with probability } \epsilon_t \\ \hat{x}_{t-1} & \text{with probability } 1-\epsilon_t \end{cases} \tag{157}$$

Typical scheduling: $\epsilon_t = \max(0, \epsilon_0 - kt)$ or $\epsilon_t = \epsilon_0 \cdot \gamma^t$

20.3.3 Curriculum Learning

Start training with shorter sequences, gradually increase length:

$$\mathcal{L}_{\text{curriculum}} = \sum_{l=l_{\min}}^{l_t}\sum_{\mathbf{x}\in\mathcal{D}_l}\mathcal{L}(\mathbf{x}; \theta) \tag{158}$$

where l_t increases over training time.

20.4 Generation and Inference

20.4.1 Greedy Decoding

Select most probable token at each step:

$$x_t = \arg\max_{x \in \mathcal{V}} P(x|x_{<t}; \theta) \tag{159}$$

20.4.2 Beam Search

Maintain top-k partial sequences:

$$\text{Score}(\mathbf{x}_{1:t}) = \sum_{i=1}^{t} \log P(x_i|x_{<i}; \theta) \tag{160}$$

$$\text{Normalized Score} = \frac{\text{Score}(\mathbf{x}_{1:t})}{t^\alpha} \tag{161}$$

20.4.3 Sampling Methods

Multinomial Sampling Sample from the full probability distribution:

$$x_t \sim P(\cdot|x_{<t}; \theta) \tag{162}$$

Top-k Sampling Sample from top-k most probable tokens:

$$P'(x_t|x_{<t}) = \begin{cases} \frac{P(x_t|x_{<t})}{\sum_{x \in \text{top-k}} P(x|x_{<t})} & \text{if } x_t \in \text{top-k} \\ 0 & \text{otherwise} \end{cases} \tag{163}$$

Nucleus (Top-p) Sampling Sample from smallest set with cumulative probability p:

$$\mathcal{V}_p = \text{smallest set s.t.} \sum_{x \in \mathcal{V}_p} P(x|x_{<t}) \geq p \tag{164}$$

Temperature Scaling Control randomness of sampling:

$$P_T(x_t|x_{<t}) = \frac{\exp(\log P(x_t|x_{<t})/T)}{\sum_{x \in \mathcal{V}} \exp(\log P(x|x_{<t})/T)} \tag{165}$$

Higher $T \rightarrow$ more random, lower $T \rightarrow$ more deterministic.

Key Formulae:

- **Beam Search Score**: $\text{score}(\mathbf{x}) = \frac{1}{|\mathbf{x}|^\alpha} \sum_{t=1}^{|\mathbf{x}|} \log P(x_t|x_{<t})$
- **Temperature**: $P_T(x) = \frac{\exp(\log P(x)/T)}{\sum_y \exp(\log P(y)/T)}$
- **Repetition Penalty**: $P'(x) = P(x)/\rho^{\text{count}(x)}$ if x appeared before

20.5 Evaluation Metrics

20.5.1 Perplexity

Measures how well the model predicts the test set:

$$\text{PPL} = \exp\left(-\frac{1}{T}\sum_{t=1}^{T} \log P(x_t|x_{<t};\theta)\right) \tag{166}$$

Lower perplexity indicates better model performance.

20.5.2 Bits Per Character (BPC)

For character-level models:

$$\text{BPC} = \frac{1}{T\log 2}\sum_{t=1}^{T} -\log P(x_t|x_{<t};\theta) \tag{167}$$

20.6 Modern Autoregressive Architectures

20.6.1 Transformer-based Models

Modern autoregressive models use Transformer decoders:

$$\mathbf{h}_t^{(0)} = \mathbf{e}_{x_t} + \mathbf{p}_t \tag{168}$$

$$\mathbf{h}_t^{(l)} = \text{TransformerBlock}^{(l)}(\mathbf{h}_t^{(l-1)}) \tag{169}$$

$$P(x_{t+1}|x_{\leq t}) = \text{softmax}(\mathbf{W}_{\text{out}}\mathbf{h}_t^{(L)}) \tag{170}$$

where $\mathbf{p}_t$ is positional encoding.

20.6.2 Causal Self-Attention

Masked attention prevents information leakage from future positions:

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T + \mathbf{M}}{\sqrt{d_k}}\right)\mathbf{V} \tag{171}$$

where mask $\mathbf{M}_{ij} = -\infty$ if $j > i$, else 0.

20.7 Applications and Use Cases

20.7.1 Text Generation

Generate coherent text continuations:

- Story completion
- Code generation
- Poetry and creative writing
- Dialogue systems

20.7.2 Language Modeling Pretraining

Foundation for many downstream tasks:

- Transfer learning to specific domains
- Few-shot learning via prompting
- Fine-tuning for classification tasks

20.7.3 Conditional Generation

Generate text conditioned on prompts:

$$P(\text{completion}|\text{prompt}) = \prod_{t=|\text{prompt}|+1}^{T} P(x_t|\text{prompt}, x_{|\text{prompt}|+1:t-1}) \tag{172}$$

Key Points:

- Autoregressive models are fundamental to modern NLP
- Chain rule decomposition enables tractable training
- Teacher forcing accelerates training but creates exposure bias
- Various decoding strategies trade off quality and diversity
- Transformer architectures have largely replaced RNNs
- Pre-trained autoregressive models enable transfer learning

Practical Pointers:

- Use gradient clipping to stabilize training
- Implement proper masking to prevent information leakage
- Consider different decoding strategies based on application needs
- Monitor perplexity during training as quality metric
- Use pre-trained models when possible for downstream tasks
- Apply regularization techniques to prevent overfitting

21 BLEU Score

BLEU (Bilingual Evaluation Understudy) is a metric for evaluating machine translation quality by comparing n-gram overlap with reference translations.

Key Terms & Definitions:

- **BLEU Score**: Automatic evaluation metric for translation quality
- **N-gram Precision**: Fraction of candidate n-grams found in references
- **Brevity Penalty**: Penalty for translations that are too short
- **Modified Precision**: Clipped n-gram counts to avoid repetition bonus
- **Geometric Mean**: Method for combining different n-gram precisions

21.1 Mathematical Definition

21.1.1 Modified N-gram Precision

$$p_n = \frac{\sum_{C \in \{\text{Candidates}\}} \sum_{\text{n-gram} \in C} \text{Count}_{\text{clip}}(\text{n-gram})}{\sum_{C' \in \{\text{Candidates}\}} \sum_{\text{n-gram}' \in C'} \text{Count}(\text{n-gram}')} \tag{173}$$

where $\text{Count}_{\text{clip}}(\text{n-gram})$ is:

$$\text{Count}_{\text{clip}}(\text{n-gram}) = \min(\text{Count}(\text{n-gram}), \text{Max\_Ref\_Count}(\text{n-gram})) \tag{174}$$

21.1.2 Brevity Penalty

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{175}$$

where c is candidate length and r is reference length.

21.1.3 BLEU Score

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{176}$$

Typically, $N = 4$ and $w_n = 1/N$ for uniform weights.

21.2 BLEU Variants

21.2.1 BLEU-4 Formula

$$\text{BLEU-4} = BP \cdot \sqrt[4]{p_1 \cdot p_2 \cdot p_3 \cdot p_4} \tag{177}$$

21.2.2 Sentence-level BLEU

For individual sentences, smoothing is often applied:

$$p_n^{\text{smooth}} = \frac{\text{Count}_{\text{clip}}(n\text{-grams}) + \epsilon}{\text{Count}(n\text{-grams}) + \epsilon} \tag{178}$$

21.3 BLEU Limitations and Alternatives

Key Points:

- BLEU focuses on precision, not recall
- Doesn't capture semantic equivalence
- Sensitive to tokenization and preprocessing
- Poor correlation with human judgment for single sentences
- Alternatives include METEOR, ROUGE, BERTScore, and human evaluation

Key Formulae:

- **METEOR**: Includes stemming, synonyms, and paraphrases
- **ROUGE-L**: Based on longest common subsequence
- **ChrF**: Character-level n-gram matching
- **BERTScore**: Uses contextual embeddings for semantic similarity

Practical Pointers:

- Report multiple evaluation metrics, not just BLEU
- Use consistent tokenization for fair comparison
- Consider domain-specific evaluation when possible
- Complement automatic metrics with human evaluation

22 Training Large Language Models

Large Language Models (LLMs) are Transformer-based models trained on massive text corpora to learn general language understanding and generation capabilities.

Key Terms & Definitions:

- **Large Language Model (LLM)**: Neural language model with billions of parameters
- **Pre-training**: Initial training on large unlabeled text corpus
- **Fine-tuning**: Adapting pre-trained model to specific tasks
- **Few-shot Learning**: Learning from few examples during inference
- **In-context Learning**: Learning from examples in the input prompt
- **Emergent Abilities**: Capabilities that arise at scale

- **Scaling Laws**: Relationships between model performance and scale

22.1 Model Architecture and Scale

22.1.1 Scaling Dimensions

LLM performance improves with three key factors:

- **Model size**: Number of parameters (N)
- **Dataset size**: Number of training tokens (D)
- **Compute**: Training FLOPs (C)

22.1.2 Scaling Laws

Power law relationships discovered by Kaplan et al.:

$$L(N) \propto N^{-\alpha_N} \quad \alpha_N \approx 0.076 \tag{179}$$
$$L(D) \propto D^{-\alpha_D} \quad \alpha_D \approx 0.095 \tag{180}$$
$$L(C) \propto C^{-\alpha_C} \quad \alpha_C \approx 0.057 \tag{181}$$

where L is cross-entropy loss.

22.1.3 Compute-Optimal Scaling (Chinchilla)

For fixed compute budget, should scale model size and data equally:

$$N_{\text{optimal}} \propto C^{0.5}, \quad D_{\text{optimal}} \propto C^{0.5} \tag{182}$$

22.1.4 Model Architectures

Decoder-Only Models (GPT family):

- Autoregressive generation
- Causal self-attention masking
- Suitable for generation tasks

Encoder-Only Models (BERT family):

- Bidirectional attention
- Good for understanding tasks
- Require task-specific heads

Encoder-Decoder Models (T5, UL2):

- Flexible for various tasks
- Cross-attention between encoder/decoder
- Higher parameter efficiency

Key Formulae:

- **Parameter Count**: $N = 12 \cdot L \cdot d_{\text{model}}^2$ (approximate for Transformer)
- **Training FLOPs**: $C \approx 6ND$ (forward + backward pass)
- **Inference FLOPs**: $C_{\text{inf}} \approx 2N$ per token
- **Memory Usage**: $M \approx 20N$ bytes (with optimizer states)

22.2 Pre-training Objectives

22.2.1 Causal Language Modeling (CLM)

Standard autoregressive objective:

$$\mathcal{L}_{\text{CLM}} = -\sum_{i=1}^{T} \log P(x_i | x_{<i}; \theta) \tag{183}$$

Used by GPT family models.

22.2.2 Masked Language Modeling (MLM)

Bidirectional objective with random masking:

$$\mathcal{L}_{\text{MLM}} = -\sum_{i \in \mathcal{M}} \log P(x_i | x_{\backslash \mathcal{M}}; \theta) \tag{184}$$

where $\mathcal{M}$ is set of masked positions.
Masking Strategy:

- 15% of tokens selected for masking
- 80% replaced with [MASK] token
- 10% replaced with random token
- 10% left unchanged

22.2.3 Prefix LM

Combines bidirectional encoder with autoregressive decoder:

$$\mathcal{L}_{\text{PrefixLM}} = -\sum_{i=k+1}^{T} \log P(x_i | x_{1:k}, x_{k+1:i-1}; \theta) \tag{185}$$

22.2.4 Span Corruption (T5)

Mask contiguous spans and predict them:

$$\text{Input}: \quad \text{"The cat [MASK] on the [MASK]"} \tag{186}$$
$$\text{Target}: \quad \text{"[MASK\_1] sat [MASK\_2] mat [END]"} \tag{187}$$

22.3 Training Infrastructure

22.3.1 Distributed Training

Data Parallelism:

- Replicate model across devices
- Split batch across devices
- Aggregate gradients using AllReduce

Model Parallelism:

- Split model layers across devices
- Pipeline parallelism for sequential processing
- Tensor parallelism within layers

3D Parallelism: Combines data, pipeline, and tensor parallelism:

$$\text{Total GPUs} = \text{DP} \times \text{PP} \times \text{TP} \tag{188}$$

22.3.2 Memory Optimization

Gradient Checkpointing: Trade compute for memory by recomputing activations:

$$\text{Memory} = O(\sqrt{L}) \text{ instead of } O(L) \tag{189}$$

ZeRO (Zero Redundancy Optimizer):

- Stage 1: Partition optimizer states
- Stage 2: Partition gradients
- Stage 3: Partition parameters

Memory reduction:

$$\text{Memory per GPU} = \frac{\text{Model States}}{N_{\text{GPUs}}} + \text{Activations} \tag{190}$$

Mixed Precision Training:

- FP16 for forward/backward pass
- FP32 for optimizer states
- Loss scaling to prevent underflow

Key Formulae:

- **Gradient Accumulation**: $\nabla_{\text{effective}} = \frac{1}{K}\sum_{k=1}^{K} \nabla_k$
- **Learning Rate Scaling**: $\text{lr}_{\text{scaled}} = \text{lr}_{\text{base}} \times \sqrt{\text{batch size}}$
- **Warmup Steps**: $\text{warmup} = \max(10000, 0.01 \times \text{total steps})$

22.4 Data and Preprocessing

22.4.1 Training Data Sources

- **Web Crawls**: Common Crawl, C4 (Colossal Clean Crawled Corpus)
- **Books**: Project Gutenberg, BookCorpus
- **News Articles**: Reuters, CNN, BBC archives
- **Academic Papers**: arXiv, PubMed abstracts
- **Code Repositories**: GitHub, StackOverflow
- **Reference Materials**: Wikipedia, dictionaries

22.4.2 Data Preprocessing Pipeline

1. **Deduplication**: Remove exact and near-duplicate content
2. **Language Detection**: Filter for target languages
3. **Quality Filtering**: Remove low-quality content
4. **Privacy Filtering**: Remove personal information (PII)
5. **Toxicity Filtering**: Remove harmful or offensive content
6. **Tokenization**: Convert to model input format

22.4.3 Data Quality Metrics

$$\begin{align}
\text{Perplexity Filter} &: \text{Remove high-perplexity text} \tag{191}\\
\text{Length Filter} &: \text{Remove very short/long documents} \tag{192}\\
\text{Language Score} &: \text{Confidence in language identification} \tag{193}\\
\text{Content Score} &: \text{Educational/informational value} \tag{194}
\end{align}$$

22.5 Fine-tuning and Alignment

22.5.1 Supervised Fine-tuning (SFT)

Adapt pre-trained model to specific tasks:

$$\mathcal{L}_{\text{SFT}} = -\sum_{(x,y)\in\mathcal{D}_{\text{task}}} \log P(y|x;\theta) \tag{195}$$

Common techniques:

- Lower learning rates than pre-training
- Task-specific data formatting
- Instruction tuning for general capabilities

22.5.2 Reinforcement Learning from Human Feedback (RLHF)

Step 1: Reward Model Training Train model to predict human preferences:

$$r_\phi(x, y) = \text{scalar reward for response } y \text{ to prompt } x \tag{196}$$

Step 2: RL Fine-tuning Optimize policy using PPO:

$$\mathcal{L}_{\text{RLHF}} = \mathbb{E}_{x,y}[r_\phi(x, y)] - \beta \text{KL}(\pi_\theta(y|x)||\pi_{\text{ref}}(y|x)) \tag{197}$$

where π_{ref} is reference model and β controls KL penalty.

22.5.3 Constitutional AI

Self-improvement through AI-generated critiques:

1. Generate initial responses
2. Critique responses against constitutional principles
3. Revise responses based on critiques
4. Train on revised responses

22.6 Emergent Abilities and Capabilities

22.6.1 In-Context Learning

LLMs can learn from examples in the input prompt:

$$P(y|x, \text{examples}) = \prod_{i=1}^{|y|} P(y_i|x, \text{examples}, y_{<i}) \tag{198}$$

Few-shot Prompting:

$$\text{Prompt} : \text{Task description + examples + test input} \tag{199}$$
$$\text{Format} : \text{"Input: } x_1 \text{ Output: } y_1 \text{ ... Input: } x_{\text{test}} \text{ Output:"} \tag{200}$$

22.6.2 Chain-of-Thought Reasoning

Intermediate reasoning steps improve complex problem solving:

$$P(\text{answer}|\text{question}) = P(\text{answer}|\text{reasoning})P(\text{reasoning}|\text{question}) \tag{201}$$

22.6.3 Instruction Following

Models trained to follow natural language instructions:

$$\text{Instruction} : \text{"Translate the following to French:"} \tag{202}$$
$$\text{Input} : \text{"Hello, how are you?"} \tag{203}$$
$$\text{Output} : \text{"Bonjour, comment allez-vous?"} \tag{204}$$

22.7 Evaluation and Benchmarking

22.7.1 Language Understanding Benchmarks

- **GLUE/SuperGLUE**: General language understanding
- **MMLU**: Massive multitask language understanding
- **HellaSwag**: Commonsense reasoning
- **ARC**: AI2 Reasoning Challenge

22.7.2 Generation Quality Metrics

- **Perplexity**: Model confidence on held-out data
- **BLEU/ROUGE**: N-gram overlap with references
- **BERTScore**: Semantic similarity using embeddings
- **Human Evaluation**: Relevance, coherence, factuality

22.7.3 Safety and Alignment Evaluation

- **Truthfulness**: Factual accuracy of responses
- **Harmlessness**: Avoiding harmful or toxic content
- **Helpfulness**: Providing useful assistance
- **Robustness**: Performance under adversarial inputs

Key Formulae:

- **Model FLOPs**: $C = 6ND$ for training, $2N$ per token for inference
- **Scaling Law**: $L(N) = aC^{-\alpha} + L_\infty$
- **Optimal Allocation**: $N_{\text{opt}} \propto C^a$, $D_{\text{opt}} \propto C^b$ where $a + b = 1$
- **RLHF Objective**: $\max_\pi \mathbb{E}[r(x, y)] - \beta D_{\text{KL}}(\pi || \pi_0)$

22.8 Challenges and Limitations

22.8.1 Technical Challenges

- **Computational Cost**: Training requires massive resources
- **Memory Requirements**: Storing billion-parameter models
- **Training Instability**: Large models can be difficult to train
- **Evaluation Complexity**: Assessing general capabilities

22.8.2 Capability Limitations

- **Factual Accuracy**: Models can hallucinate information
- **Reasoning**: Struggles with complex logical reasoning
- **Grounding**: Limited connection to real-world knowledge
- **Consistency**: Responses may vary across similar prompts

22.8.3 Ethical and Safety Concerns

- **Bias**: Reflecting biases present in training data
- **Misuse**: Potential for generating harmful content
- **Privacy**: Possible memorization of training data
- **Environmental**: Large carbon footprint from training

Key Points:

- LLMs achieve impressive capabilities through scale and data
- Pre-training on diverse text creates general language understanding
- Fine-tuning and RLHF align models with human preferences
- Emergent abilities appear at sufficient scale
- In-context learning enables few-shot task performance
- Significant challenges remain in safety, alignment, and efficiency

Practical Pointers:

- Use existing pre-trained models rather than training from scratch
- Focus on effective fine-tuning strategies for specific tasks
- Implement proper safety filtering and evaluation protocols
- Monitor for hallucination and factual errors
- Consider computational costs and environmental impact
- Stay updated on latest developments in alignment research

23 Applications and Case Studies

23.1 Sentiment Analysis

Classification of text into positive, negative, or neutral sentiments using RNNs:

23.1.1 Architecture

$$P(\text{sentiment}|\text{text}) = \text{softmax}(\mathbf{W}\mathbf{h}_T + \mathbf{b}) \tag{205}$$

23.1.2 Feature Engineering

- **Bag of Words**: Simple baseline with TF-IDF
- **N-grams**: Capture local context
- **Lexicon Features**: Sentiment dictionaries (VADER, SentiWordNet)
- **Syntactic Features**: POS tags, dependency parsing

Key Terms & Definitions:

- **Sentiment Polarity**: Positive, negative, or neutral emotion
- **Subjectivity**: Objective facts vs. subjective opinions
- **Aspect-based Sentiment**: Sentiment toward specific aspects
- **Emotion Detection**: Fine-grained emotional categories

23.2 Language Modeling

Predicting the next word in a sequence:

23.2.1 Mathematical Formulation

$$P(w_{t+1}|w_1, \ldots, w_t) = \text{softmax}(\mathbf{W}\mathbf{h}_t + \mathbf{b}) \tag{206}$$

23.2.2 Perplexity

Measure of how well the model predicts the test set:

$$\text{PPL} = \exp\left(-\frac{1}{T}\sum_{t=1}^{T} \log P(w_t|w_1, \ldots, w_{t-1})\right) \tag{207}$$

Lower perplexity indicates better model performance.

Practical Pointers:

- Use techniques like dropout and weight tying
- Consider character-level models for morphologically rich languages
- Apply adaptive softmax for large vocabularies
- Use pre-trained language models when available

23.3 Text Summarization

Sequence-to-sequence task generating concise summaries:

23.3.1 Extractive Summarization

Select important sentences from the original text:

$$\text{Score}(s_i) = f(\text{position}(s_i), \text{tf-idf}(s_i), \text{length}(s_i)) \tag{208}$$

23.3.2 Abstractive Summarization

Generate new sentences that capture the essence:

$$P(\text{summary}|\text{document}) = \prod_{t=1}^{T_s} P(s_t|s_1, \ldots, s_{t-1}, \text{document}) \tag{209}$$

Key Terms & Definitions:

- **Extractive**: Selecting existing sentences
- **Abstractive**: Generating new sentences
- **Coverage**: Avoiding repetition and ensuring completeness
- **Compression Ratio**: Length of summary relative to original

23.4 Speech Processing

23.4.1 Speech-to-Text (STT)

Converting audio signals to text using RNNs:

$$\mathbf{X} = \text{Audio Features (MFCC, spectrograms)} \tag{210}$$

$$P(\text{text}|\text{audio}) = \prod_{t=1}^{T} P(w_t|w_1, \ldots, w_{t-1}, \mathbf{X}) \tag{211}$$

23.4.2 Text-to-Speech (TTS)

Generating audio from text input:

$$P(\text{audio}|\text{text}) = \prod_{t=1}^{T} P(a_t|a_1, \ldots, a_{t-1}, \text{text}) \tag{212}$$

Key Terms & Definitions:

- **MFCC**: Mel-Frequency Cepstral Coefficients
- **Spectrogram**: Visual representation of audio frequencies

- **Vocoder**: Converts intermediate representation to audio
- **Prosody**: Rhythm, stress, and intonation of speech

23.5 Spam Detection

23.5.1 Logistic Regression Approach

$$P(\text{spam}|\mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}} \tag{213}$$

23.5.2 Feature Engineering for Spam Detection

- **Word Frequency**: Count of suspicious words
- **Capitalization**: Ratio of uppercase letters
- **Punctuation**: Excessive use of exclamation marks
- **URL Count**: Number of hyperlinks
- **Email Headers**: Sender reputation, domain age

23.5.3 Feature Extraction Methods

- **CountVectorizer**: Creates BoW representation
- **TF-IDF Vectorizer**: Applies TF-IDF weighting
- **Hashing Vectorizer**: Memory-efficient feature hashing

Key Formulae:

- **Precision**: $P = \frac{TP}{TP+FP}$
- **Recall**: $R = \frac{TP}{TP+FN}$
- **F1-Score**: $F1 = \frac{2PR}{P+R}$
- **False Positive Rate**: $FPR = \frac{FP}{FP+TN}$

24 Semi-supervised Learning

Semi-supervised learning leverages both labeled and unlabeled data to improve model performance when labeled data is scarce.

Key Terms & Definitions:

- **Semi-supervised Learning**: Learning with both labeled and unlabeled data
- **Consistency Regularization**: Enforcing similar outputs for perturbed versions of input

- **Pseudo-labeling**: Using model predictions as labels for unlabeled data
- **Self-training**: Iteratively adding confident predictions to training set
- **Co-training**: Training multiple models on different views of data
- **Manifold Assumption**: Similar examples lie on same low-dimensional manifold
- **Cluster Assumption**: Decision boundary should lie in low-density regions

24.1 Problem Formulation

Given:

- Labeled dataset: $\mathcal{D}_L = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n_l}$
- Unlabeled dataset: $\mathcal{D}_U = \{\mathbf{x}_j\}_{j=1}^{n_u}$
- Typically $n_u \gg n_l$

Objective: Learn function $f : \mathcal{X} \to \mathcal{Y}$ using both datasets.

24.1.1 General SSL Loss Function

$$\mathcal{L}_{\text{SSL}} = \mathcal{L}_{\text{supervised}} + \lambda \mathcal{L}_{\text{unsupervised}} \tag{214}$$

where:

$$\mathcal{L}_{\text{supervised}} = \sum_{(\mathbf{x},y)\in\mathcal{D}_L} \ell(f(\mathbf{x}), y) \tag{215}$$

$$\mathcal{L}_{\text{unsupervised}} = \sum_{\mathbf{x}\in\mathcal{D}_U} \ell_{\text{SSL}}(\mathbf{x}) \tag{216}$$

24.2 Consistency Regularization Methods

24.2.1 -Model (Pi-Model)

Enforce consistency between predictions on original and augmented inputs:

$$\mathcal{L}_{\Pi} = \sum_{\mathbf{x}\in\mathcal{D}_L\cup\mathcal{D}_U} ||\text{softmax}(f(\mathbf{x})) - \text{softmax}(f(\tilde{\mathbf{x}}))||_2^2 \tag{217}$$

where $\tilde{\mathbf{x}}$ is an augmented version of $\mathbf{x}$.

24.2.2 Temporal Ensembling

Use exponential moving average of predictions:

$$\mathbf{z}_i^{(t)} = \alpha \mathbf{z}_i^{(t-1)} + (1-\alpha) f(\mathbf{x}_i) \tag{218}$$

$$\tilde{\mathbf{z}}_i^{(t)} = \frac{\mathbf{z}_i^{(t)}}{1-\alpha^t} \tag{219}$$

$$\mathcal{L}_{\text{temp}} = ||\text{softmax}(f(\mathbf{x}_i)) - \text{softmax}(\tilde{\mathbf{z}}_i^{(t)})||_2^2 \tag{220}$$

24.2.3 Mean Teacher

Student model learns from teacher model (EMA of student):

$$\theta^{(t)}_{\text{teacher}} = \alpha\theta^{(t-1)}_{\text{teacher}} + (1-\alpha)\theta^{(t)}_{\text{student}} \tag{221}$$

$$\mathcal{L}_{\text{consistency}} = ||\text{softmax}(f_{\text{student}}(\mathbf{x})) - \text{softmax}(f_{\text{teacher}}(\tilde{\mathbf{x}}))||_2^2 \tag{222}$$

Key Formulae:

- **Consistency Loss**: $\mathcal{L}_{\text{cons}} = \mathbb{E}_{\mathbf{x}}[||f(\mathbf{x}) - f(\text{augment}(\mathbf{x}))||^2]$
- **EMA Update**: $\theta_{\text{EMA}} \leftarrow \alpha\theta_{\text{EMA}} + (1-\alpha)\theta$
- **Ramp-up Function**: $w(t) = \exp(-5(1-\min(t/T, 1))^2)$ for unsupervised weight

24.3 Pseudo-labeling Methods

24.3.1 Basic Pseudo-labeling

Use model's confident predictions as pseudo-labels:

$$\hat{y}_i = \arg\max_c P(y = c|\mathbf{x}_i; \theta) \tag{223}$$

$$\mathcal{L}_{\text{pseudo}} = \sum_{\mathbf{x}_i \in \mathcal{D}_U} \nVdash[\max_c P(y = c|\mathbf{x}_i) > \tau]\ell(\hat{y}_i, y_i^*) \tag{224}$$

where τ is confidence threshold and y_i^* is pseudo-label.

24.3.2 Self-training Algorithm

1. Train model on labeled data $\mathcal{D}_L$
2. Generate predictions on unlabeled data $\mathcal{D}_U$
3. Select confident predictions above threshold τ
4. Add pseudo-labeled examples to training set
5. Repeat until convergence

24.3.3 Co-training

Train multiple models on different feature views:

$$f_1(\mathbf{x}_1) : \mathcal{X}_1 \to \mathcal{Y} \tag{225}$$

$$f_2(\mathbf{x}_2) : \mathcal{X}_2 \to \mathcal{Y} \tag{226}$$

where $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2]$ are different views of the same input.

24.4 Generative Models for SSL

24.4.1 Variational Autoencoders (VAE) for SSL

Joint model of inputs and labels:

$$p(\mathbf{x}, y) = p(\mathbf{x}|y)p(y) \tag{227}$$

$$\text{ELBO} = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] - D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) \tag{228}$$

For labeled data:

$$\mathcal{L}_{\text{labeled}} = -\log q(y|\mathbf{x}) - \text{ELBO}(\mathbf{x}) \tag{229}$$

For unlabeled data:

$$\mathcal{L}_{\text{unlabeled}} = -\sum_{y} q(y|\mathbf{x})\text{ELBO}(\mathbf{x}, y) \tag{230}$$

24.4.2 Generative Adversarial Networks for SSL

Discriminator distinguishes between:

- Real labeled samples: $(x, y) \sim p_{\text{data}}$
- Real unlabeled samples: $x \sim p_{\text{data}}$
- Fake samples: $x \sim p_{\text{generator}}$

Loss function:

$$\mathcal{L}_D = \mathbb{E}_{x,y\sim p_{\text{data}}}[\log p(y|x)] + \mathbb{E}_{x\sim p_{\text{data}}}[\log(1 - p(\text{fake}|x))] \tag{231}$$

$$+ \mathbb{E}_{x\sim p_G}[\log p(\text{fake}|x)] \tag{232}$$

24.5 Graph-based Semi-supervised Learning

24.5.1 Label Propagation

Construct graph where nodes are examples and edges represent similarity:

$$W_{ij} = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right) \tag{233}$$

Label propagation update:

$$\mathbf{Y}^{(t+1)} = \alpha\mathbf{S}\mathbf{Y}^{(t)} + (1 - \alpha)\mathbf{Y}^{(0)} \tag{234}$$

where $\mathbf{S} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ is normalized adjacency matrix.

24.5.2 Graph Neural Networks for SSL

Use GNNs to propagate information through graph structure:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{W}^{(l)}\mathbf{h}_i^{(l)} + \sum_{j\in\mathcal{N}(i)} \alpha_{ij}\mathbf{h}_j^{(l)}\right) \tag{235}$$

$$\alpha_{ij} = \text{softmax}\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i||\mathbf{W}\mathbf{h}_j]\right)\right) \tag{236}$$

24.6 Recent Advances: MixMatch and FixMatch

24.6.1 MixMatch

Combines multiple SSL techniques:

1. Generate multiple augmented versions of unlabeled samples
2. Average predictions to get "guess" labels
3. Apply MixUp to both labeled and unlabeled data
4. Use separate losses for labeled and unlabeled data

MixUp operation:

$$\lambda \sim \text{Beta}(\alpha, \alpha) \tag{237}$$

$$\tilde{\mathbf{x}} = \lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2 \tag{238}$$

$$\tilde{\mathbf{y}} = \lambda \mathbf{y}_1 + (1 - \lambda)\mathbf{y}_2 \tag{239}$$

24.6.2 FixMatch

Combines consistency regularization with pseudo-labeling:

$$\mathcal{L}_s = \frac{1}{B} \sum_{b=1}^{B} \mathbb{H}(y_b, p_m(\hat{y}|\alpha(\mathbf{x}_b))) \tag{240}$$

$$\mathcal{L}_u = \frac{1}{\mu B} \sum_{b=1}^{\mu B} \nVdash(\max(q_b) \geq \tau)\mathbb{H}(\hat{q}_b, p_m(\hat{y}|\mathcal{A}(\mathbf{u}_b))) \tag{241}$$

where:

- $\alpha(\cdot)$ is weak augmentation
- $\mathcal{A}(\cdot)$ is strong augmentation
- τ is confidence threshold
- $q_b = p_m(\hat{y}|\alpha(\mathbf{u}_b))$ are weak augmentation predictions

Key Points:

- SSL is crucial when labeled data is expensive or scarce
- Consistency regularization enforces smooth predictions
- Pseudo-labeling leverages model's confident predictions
- Graph-based methods exploit similarity structure
- Modern methods combine multiple SSL techniques effectively
- Data augmentation is critical for SSL success

24.7 Applications in NLP

24.7.1 Text Classification with Limited Labels

- Use pre-trained language models for initialization
- Apply consistency regularization with text augmentation
- Generate pseudo-labels for confident predictions

24.7.2 Named Entity Recognition

- Self-training with confident entity predictions
- Cross-lingual transfer using multilingual models
- Distant supervision from knowledge bases

24.7.3 Machine Translation

- Back-translation for monolingual data utilization
- Multi-lingual models trained on related languages
- Iterative self-training approaches

Practical Pointers:

- Start with strong data augmentation strategies
- Carefully tune confidence thresholds for pseudo-labeling
- Use exponential moving averages for stable training
- Apply gradual ramp-up of unsupervised loss weight
- Consider pre-trained models as strong baselines
- Validate on held-out labeled data, not pseudo-labeled data

25 Reinforcement Learning

Reinforcement Learning (RL) is a learning paradigm where an agent learns to make decisions by interacting with an environment to maximize cumulative reward.

Key Terms & Definitions:

- **Agent**: The learner or decision maker
- **Environment**: The world the agent interacts with
- **State**: Current situation of the agent in the environment

- **Action**: Choice made by the agent
- **Reward**: Numerical feedback from the environment
- **Policy**: Strategy that defines agent's behavior
- **Value Function**: Expected cumulative reward from a state
- **Markov Decision Process (MDP)**: Mathematical framework for RL

25.1 Markov Decision Processes (MDPs)

An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$:

- $\mathcal{S}$: State space
- $\mathcal{A}$: Action space
- $P(s'|s,a)$: Transition probability function
- $R(s,a,s')$: Reward function
- $\gamma \in [0,1]$: Discount factor

25.1.1 Markov Property

$$P(S_{t+1}=s', R_{t+1}=r|S_t, A_t, S_{t-1}, A_{t-1}, \ldots) = P(S_{t+1}=s', R_{t+1}=r|S_t, A_t) \tag{242}$$

25.1.2 Return and Value Functions

Return (discounted cumulative reward):

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{243}$$

State Value Function:

$$v^\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right] \tag{244}$$

Action-State Value Function (Q-function):

$$q^\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{245}$$

25.1.3 Bellman Equations

Bellman Equation for v^π:

$$v^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v^\pi(s')] \tag{246}$$

Bellman Equation for q^π:

$$q^\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \sum_{a'} \pi(a'|s') q^\pi(s',a')] \tag{247}$$

Bellman Optimality Equations:

$$v^*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v^*(s')] \tag{248}$$

$$q^*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q^*(s',a')] \tag{249}$$

Key Formulae:

- **Policy**: $\pi(a|s) = P(A_t = a|S_t = s)$
- **Optimal Policy**: $\pi^*(a|s) = \arg\max_a q^*(s,a)$
- **Advantage Function**: $A^\pi(s,a) = q^\pi(s,a) - v^\pi(s)$
- **TD Error**: $\delta_t = R_{t+1} + \gamma v(S_{t+1}) - v(S_t)$

25.2 Dynamic Programming

25.2.1 Policy Evaluation

Iteratively compute v^π for a given policy π:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')] \tag{250}$$

25.2.2 Policy Improvement

Improve policy greedily:

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v^\pi(s')] \tag{251}$$

25.2.3 Policy Iteration

Alternate between policy evaluation and improvement:

1. Initialize π_0
2. Evaluate: $v^{\pi_k} \leftarrow$ solve Bellman equation for π_k
3. Improve: $\pi_{k+1} \leftarrow \text{greedy}(v^{\pi_k})$
4. Repeat until convergence

25.2.4 Value Iteration

Direct optimization of value function:

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')] \tag{252}$$

25.3 Monte Carlo Methods

Learn from complete episodes without model knowledge.

25.3.1 First-Visit MC Policy Evaluation

$$v^\pi(s) = \text{average of returns following first visits to } s \tag{253}$$

25.3.2 Every-Visit MC Policy Evaluation

$$v^\pi(s) = \text{average of returns following all visits to } s \tag{254}$$

25.3.3 Monte Carlo Control

On-policy Monte Carlo Control:

1. Generate episode using π
2. For each state-action pair (s, a) in episode:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[G - Q(s, a)] \tag{255}$$

3. Improve policy: $\pi(s) \leftarrow \arg\max_a Q(s, a)$

Off-policy Monte Carlo with Importance Sampling:

$$v^\pi(s) = \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|} \tag{256}$$

where $\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$ is importance sampling ratio.

25.4 Temporal Difference Learning

Learn from incomplete episodes using bootstrapping.

25.4.1 TD(0) for State Values

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{257}$$

25.4.2 SARSA (On-policy TD Control)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{258}$$

25.4.3 Q-Learning (Off-policy TD Control)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{259}$$

25.4.4 Expected SARSA

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{260}$$

25.4.5 TD(λ)

Uses eligibility traces for credit assignment:

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma V(S_{t+1}) - V(S_t) && (261) \\ E_t(s) &= \gamma\lambda E_{t-1}(s) + \nVdash[S_t = s] && (262) \\ V(s) &\leftarrow V(s) + \alpha\delta_t E_t(s) \quad \forall s && (263) \end{aligned}$$

Key Formulae:

- **TD Error**: $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
- **Q-Learning Update**: $Q(s,a) \leftarrow Q(s,a) + \alpha[\text{target} - Q(s,a)]$
- **Eligibility Trace**: $E_t(s) = \gamma\lambda E_{t-1}(s) + \nVdash[S_t = s]$
- **ϵ-greedy**: $\pi(a|s) = \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}| & \text{if } a = \arg\max Q(s,a) \\ \epsilon/|\mathcal{A}| & \text{otherwise} \end{cases}$

25.5 Multi-Armed Bandit Problem

Simplified RL problem with single state and multiple actions.

25.5.1 Problem Setup

- K arms (actions), each with unknown reward distribution
- Goal: maximize cumulative reward over T time steps
- Trade-off between exploration and exploitation

25.5.2 Action Value and Regret

True action value:

$$q^*(a) = \mathbb{E}[R_t | A_t = a] \tag{264}$$

Regret:

$$\text{Regret}_T = T \cdot q^*(a^*) - \sum_{t=1}^{T} q^*(A_t) \tag{265}$$

where $a^* = \arg\max_a q^*(a)$.

25.5.3 Action Value Estimation

Sample Average Method:

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i \nVdash[A_i = a]}{\sum_{i=1}^{t-1} \nVdash[A_i = a]} \tag{266}$$

Incremental Update:

$$Q_{t+1}(A_t) = Q_t(A_t) + \frac{1}{N_t(A_t)}[R_t - Q_t(A_t)] \tag{267}$$

25.5.4 Exploration Strategies

ϵ-greedy:

$$A_t = \begin{cases} \arg\max_a Q_t(a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases} \tag{268}$$

Upper Confidence Bound (UCB):

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right] \tag{269}$$

where $c > 0$ controls exploration level and $N_t(a)$ is number of times action a was selected.
Thompson Sampling: Sample action according to probability it's optimal:

$$\theta_a^{(t)} \sim P(\theta_a | \text{data}_t) \tag{270}$$

$$A_t = \arg\max_a \mathbb{E}[R | \theta_a^{(t)}] \tag{271}$$

Gradient Bandit Algorithm: Maintain preferences $H_t(a)$ and use softmax policy:

$$\pi_t(a) = \frac{e^{H_t(a)}}{\sum_{b=1}^k e^{H_t(b)}} \tag{272}$$

$$H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)) \tag{273}$$

$$H_{t+1}(a) = H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a) \quad \forall a \neq A_t \tag{274}$$

where $\bar{R}_t$ is average reward up to time t.

25.6 Policy Gradient Methods

Directly optimize parameterized policies $\pi(a|s, \boldsymbol{\theta})$.

25.6.1 Policy Gradient Theorem

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[\sum_a q^\pi(s,a) \nabla \pi(a|s, \boldsymbol{\theta}) \right] \tag{275}$$

REINFORCE Algorithm:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \nabla \ln \pi(A_t | S_t, \boldsymbol{\theta}_t) \tag{276}$$

REINFORCE with Baseline:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(G_t - b(S_t)) \nabla \ln \pi(A_t | S_t, \boldsymbol{\theta}_t) \tag{277}$$

where $b(s)$ is baseline (often $v^\pi(s)$).

25.6.2 Actor-Critic Methods

Combine policy gradient (actor) with value function approximation (critic):

$$\text{Critic Update: } \mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_w \delta_t \nabla \hat{v}(S_t, \mathbf{w}_t) \tag{278}$$

$$\text{Actor Update: } \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_\theta \delta_t \nabla \ln \pi(A_t | S_t, \boldsymbol{\theta}_t) \tag{279}$$

where $\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t)$.

25.7 Deep Reinforcement Learning

25.7.1 Deep Q-Networks (DQN)

Use neural networks to approximate Q-functions:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[(r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}^-) - Q(s, a; \boldsymbol{\theta}))^2\right] \tag{280}$$

Key innovations:

- Experience replay buffer $\mathcal{D}$
- Target network with parameters $\boldsymbol{\theta}^-$
- ϵ-greedy exploration

25.7.2 Policy Gradient with Function Approximation

Advantage Actor-Critic (A2C):

$$\begin{aligned} \nabla_{\boldsymbol{\theta} J(\boldsymbol{\theta})} &= \mathbb{E}_\pi[\nabla_{\boldsymbol{\theta} \ln \pi(a|s,\boldsymbol{\theta}) A^\pi(s,a)]} && (281) \\ A^\pi(s, a) &= Q^\pi(s, a) - V^\pi(s) && (282) \\ &\approx r + \gamma V(s') - V(s) && (283) \end{aligned}$$

Proximal Policy Optimization (PPO):

$$\mathcal{L}^{\text{CLIP}}(\boldsymbol{\theta}) = \mathbb{E}_t\left[\min(r_t(\boldsymbol{\theta})\hat{A}_t, \text{clip}(r_t(\boldsymbol{\theta}), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)\right] \tag{284}$$

where $r_t(\boldsymbol{\theta}) = \frac{\pi(a_t|s_t,\boldsymbol{\theta})}{\pi_{\text{old}}(a_t|s_t)}$ is probability ratio.

Key Formulae:

- **Policy Gradient**: $\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi[G_t \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})]$
- **DQN Loss**: $\mathcal{L} = (r + \gamma \max_{a'} Q_{\text{target}}(s', a') - Q(s, a))^2$
- **PPO Clipping**: $r(\boldsymbol{\theta}) = \frac{\pi(a|s,\boldsymbol{\theta})}{\pi_{\text{old}}(a|s)}$
- **GAE**: $\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$

25.8 Applications in NLP

25.8.1 Neural Machine Translation

Use RL to optimize BLEU score directly:

$$\begin{aligned} \text{Reward: } r &= \text{BLEU}(\text{generated}, \text{reference}) && (285) \\ \text{Policy: } \pi(y_t|y_{<t}, x) &= \text{decoder output distribution} && (286) \end{aligned}$$

25.8.2 Dialogue Systems

- **State**: Conversation history
- **Action**: Response generation
- **Reward**: User satisfaction, task completion
- **Policy**: Response selection strategy

25.8.3 Text Summarization

- **State**: Document and partial summary
- **Action**: Next word/sentence selection
- **Reward**: ROUGE score, readability metrics

25.8.4 Information Extraction

- **State**: Text with current extraction state
- **Action**: Extract entity/relation or move to next position
- **Reward**: Precision/recall of extractions

Key Points:

- RL provides framework for sequential decision making
- Exploration-exploitation trade-off is fundamental challenge
- Value functions and policies are core concepts
- Deep RL combines neural networks with RL algorithms
- Applications in NLP often involve optimizing non-differentiable metrics
- Sample efficiency remains major challenge in RL

Practical Pointers:

- Start with simple tabular methods for understanding
- Use experience replay and target networks for stable deep RL
- Carefully design reward functions to avoid unintended behavior
- Consider pre-training with supervised learning before RL fine-tuning
- Monitor training stability and use appropriate regularization
- Validate on diverse test cases to ensure robustness

26 Advanced Topics and Optimization

26.1 Sequence Batching

Efficient training with variable-length sequences requires special handling:

26.1.1 Padding

Pad shorter sequences to match the longest sequence in batch:

$$\mathbf{x}_{\text{padded}} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T, \mathbf{0}, \ldots, \mathbf{0}] \tag{287}$$

26.1.2 Masking

Apply masks to ignore padded positions:

$$\text{mask}_t = \begin{cases} 1 & \text{if } t \leq \text{actual length} \\ 0 & \text{otherwise} \end{cases} \tag{288}$$

26.1.3 Packed Sequences

More memory-efficient representation for RNNs:

$$\text{PackedSequence} = (\text{data}, \text{batch\_sizes}, \text{sorted\_indices}) \tag{289}$$

Practical Pointers:

- Sort sequences by length for efficient batching
- Use dynamic padding to minimize wasted computation
- Implement custom collate functions for DataLoader
- Monitor memory usage with large batch sizes

26.2 Regularization Techniques

26.2.1 Dropout

Randomly set activations to zero during training:

$$\mathbf{h}_{\text{dropout}} = \mathbf{h} \odot \mathbf{m} \tag{290}$$

where $\mathbf{m} \sim \text{Bernoulli}(1-p)$ and $\odot$ is element-wise multiplication.

26.2.2 Recurrent Dropout

Apply dropout to recurrent connections:

$$\mathbf{h}_t = \tanh(\mathbf{W}_h(\mathbf{h}_{t-1} \odot \mathbf{m}_h) + \mathbf{W}_x\mathbf{x}_t + \mathbf{b}) \tag{291}$$

26.2.3 Variational Dropout

Use the same dropout mask across all time steps:

$$\mathbf{h}_t = \tanh(\mathbf{W}_h(\mathbf{h}_{t-1} \odot \mathbf{m}) + \mathbf{W}_x(\mathbf{x}_t \odot \mathbf{m}_x) + \mathbf{b}) \tag{292}$$

26.2.4 Weight Decay (L2 Regularization)

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda \sum_i \mathbf{W}_i^2 \tag{293}$$

26.2.5 Gradient Clipping

Prevent exploding gradients:

$$\mathbf{g}_{\text{clipped}} = \min\left(1, \frac{\text{threshold}}{||\mathbf{g}||}\right) \mathbf{g} \tag{294}$$

Key Formulae:

- **L1 Regularization**: $\mathcal{L}_{\text{L1}} = \lambda \sum_i |\mathbf{W}_i|$
- **Elastic Net**: $\mathcal{L}_{\text{EN}} = \lambda_1 \sum_i |\mathbf{W}_i| + \lambda_2 \sum_i \mathbf{W}_i^2$
- **Early Stopping**: Stop when validation loss stops improving

26.3 Learning Rate Scheduling

26.3.1 Exponential Decay

$$\text{lr}_t = \text{lr}_0 \cdot \gamma^{t/\text{step\_size}} \tag{295}$$

26.3.2 Cosine Annealing

$$\text{lr}_t = \text{lr}_{\min} + \frac{1}{2}(\text{lr}_{\max} - \text{lr}_{\min}) \left(1 + \cos\left(\frac{\pi t}{T_{\max}}\right)\right) \tag{296}$$

26.3.3 Reduce on Plateau

$$\text{lr}_{t+1} = \begin{cases} \text{lr}_t \cdot \text{factor} & \text{if validation loss plateaued} \\ \text{lr}_t & \text{otherwise} \end{cases} \tag{297}$$

26.3.4 Warmup Scheduling

$$\text{lr}_t = \begin{cases} \text{lr}_{\max} \cdot \frac{t}{t_{\text{warmup}}} & \text{if } t < t_{\text{warmup}} \\ \text{lr}_{\max} \cdot \text{decay}(t - t_{\text{warmup}}) & \text{otherwise} \end{cases} \tag{298}$$

Practical Pointers:

- Start with simple step decay, then try more sophisticated schedules
- Use learning rate finder to determine good initial learning rate
- Monitor both training and validation metrics
- Save checkpoints at best validation performance

27 Evaluation Metrics

27.1 Classification Metrics

27.1.1 Confusion Matrix

	Predicted 0	**Predicted 1**
Actual 0	TN	FP
Actual 1	FN	TP

27.1.2 Basic Metrics

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{299}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{300}$$

$$\text{Recall (Sensitivity)} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{301}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{302}$$

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{303}$$

27.1.3 Multi-class Extensions

$$\text{Macro F1} = \frac{1}{K} \sum_{k=1}^{K} \text{F1}_k \tag{304}$$

$$\text{Micro F1} = 2 \cdot \frac{\sum_k \text{TP}_k \cdot \sum_k \text{TP}_k}{\sum_k \text{TP}_k + \sum_k \text{FP}_k + \sum_k \text{TP}_k + \sum_k \text{FN}_k} \tag{305}$$

$$\text{Weighted F1} = \sum_{k=1}^{K} \frac{n_k}{n} \text{F1}_k \tag{306}$$

27.2 Ranking Metrics

27.2.1 ROC AUC

Area Under the Receiver Operating Characteristic curve:

$$\text{AUC} = \int_0^1 \text{TPR}(\text{FPR}^{-1}(x))dx \tag{307}$$

where TPR = True Positive Rate, FPR = False Positive Rate.

27.2.2 Average Precision (AP)

$$\text{AP} = \sum_{k=1}^{n} P(k) \cdot \Delta R(k) \tag{308}$$

where $P(k)$ is precision at rank k and $\Delta R(k)$ is change in recall.

27.3 Sequence Labeling Metrics

27.3.1 Entity-Level F1

For NER tasks, evaluate at entity level rather than token level:

$$\text{Entity F1} = 2 \cdot \frac{P_{\text{entity}} \cdot R_{\text{entity}}}{P_{\text{entity}} + R_{\text{entity}}} \tag{309}$$

27.3.2 Exact Match

Percentage of sequences where all labels are correct:

$$\text{EM} = \frac{1}{N} \sum_{i=1}^{N} \nVdash[\hat{\mathbf{y}}_i = \mathbf{y}_i] \tag{310}$$

27.4 Language Generation Metrics

27.4.1 ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

For text summarization:

ROUGE-N

$$\text{ROUGE-N} = \frac{\sum_{S \in \{\text{Reference}\}} \sum_{\text{n-gram} \in S} \text{Count}_{\text{match}}(\text{n-gram})}{\sum_{S \in \{\text{Reference}\}} \sum_{\text{n-gram} \in S} \text{Count}(\text{n-gram})} \tag{311}$$

ROUGE-L Based on Longest Common Subsequence:

$$\text{ROUGE-L} = \frac{(1 + \beta^2) R_{\text{lcs}} P_{\text{lcs}}}{R_{\text{lcs}} + \beta^2 P_{\text{lcs}}} \tag{312}$$

27.4.2 Perplexity for Language Models

$$\text{PPL} = \exp\left(-\frac{1}{T}\sum_{t=1}^{T}\log P(w_t|w_{<t})\right) \tag{313}$$

Key Points:

- Choose metrics appropriate for your task and dataset characteristics
- Report multiple metrics to get a complete picture
- Use statistical significance tests when comparing models
- Consider domain-specific evaluation criteria

28 Practical Tips and Best Practices

28.1 Model Selection Guidelines

Key Points:

- **Task Complexity**: Use simpler models (GRU, shallow networks) for straightforward tasks
- **Data Size**: Larger datasets can support more complex architectures
- **Computational Resources**: Balance model complexity with available computing power
- **Interpretability Requirements**: Simpler models are generally more interpretable

28.1.1 Architecture Decision Tree

- **Short sequences (¡50 tokens)**: Consider CNN or simple RNN
- **Medium sequences (50-200 tokens)**: GRU or LSTM work well
- **Long sequences (¿200 tokens)**: LSTM with attention or Transformer
- **Sequential labeling**: Bidirectional RNN + CRF
- **Sequence-to-sequence**: Encoder-decoder with attention

28.2 Hyperparameter Tuning

Parameter	Typical Range	Tuning Strategy
Hidden size	64-1024	Start small, increase if underfitting
Learning rate	1e-4 to 1e-2	Use learning rate finder
Batch size	16-128	Larger is generally better (memory permitting)
Dropout	0.1-0.5	Start with 0.2, increase if overfitting
Layers	1-6	Start with 2, add more if needed
Sequence length	Task-dependent	Analyze your data distribution

Table 3: Hyperparameter Tuning Guidelines

Practical Pointers:

- Use random search or Bayesian optimization over grid search
- Start with reasonable defaults and tune one parameter at a time
- Monitor both training and validation metrics
- Use early stopping to prevent overfitting

28.3 Data Preprocessing Best Practices

28.3.1 Text Cleaning Pipeline

1. **Encoding**: Convert to UTF-8 for consistent character handling
2. **Case Normalization**: Lowercase unless case carries meaning
3. **Punctuation**: Remove or normalize based on task requirements
4. **Special Characters**: Handle URLs, emails, mentions consistently
5. **Whitespace**: Normalize multiple spaces and remove leading/trailing
6. **Numbers**: Decide whether to keep, remove, or normalize

28.3.2 Tokenization Considerations

- **Language-specific**: Use appropriate tokenizers for each language
- **Domain-specific**: Medical, legal, and technical texts need special handling
- **Consistency**: Same preprocessing for training, validation, and test
- **Subword Units**: Consider BPE or SentencePiece for morphologically rich languages

28.3.3 Vocabulary Management

$$\begin{aligned}\text{Vocab Size} &= \text{Trade-off between coverage and efficiency} \quad (314)\\ \text{Coverage} &= \frac{\text{Known tokens}}{\text{Total tokens}} \quad (315)\\ \text{OOV Rate} &= 1 - \text{Coverage} \quad (316)\end{aligned}$$

Practical Pointers:

- Keep vocabulary size between 10K-50K for most tasks
- Use frequency-based filtering: keep words appearing 5 times
- Reserve special tokens: `<PAD>`, `<UNK>`, `<SOS>`, `<EOS>`
- Consider domain adaptation when using pre-trained embeddings

28.4 Training Strategies

28.4.1 Curriculum Learning

Start with easier examples, gradually increase difficulty:

$$\mathcal{L}_{\text{curriculum}}(\theta, t) = \sum_i w_i(t)\mathcal{L}(f(\mathbf{x}_i; \theta), \mathbf{y}_i) \quad (317)$$

where $w_i(t)$ increases over time for harder examples.

28.4.2 Teacher Forcing vs. Scheduled Sampling

$$\begin{aligned}\text{Teacher Forcing:} \quad & P(y_t | y^*_{<t}, \mathbf{x}) \quad (318)\\ \text{Scheduled Sampling:} \quad & P(y_t | \text{mix}(y^*_{<t}, \hat{y}_{<t}), \mathbf{x}) \quad (319)\end{aligned}$$

28.4.3 Transfer Learning Strategies

1. **Feature Extraction**: Freeze pre-trained layers, train classifier
2. **Fine-tuning**: Train all layers with lower learning rate
3. **Gradual Unfreezing**: Unfreeze layers progressively
4. **Domain Adaptation**: Adapt to specific domain/task

28.4.4 Multi-task Learning

Joint training on related tasks:

$$\mathcal{L}_{\text{total}} = \sum_{k=1}^{K} \lambda_k \mathcal{L}_k(\theta_{\text{shared}}, \theta_k) \quad (320)$$

Key Points:

- Use early stopping based on validation performance
- Monitor gradient norms to detect training issues
- Save checkpoints regularly for recovery
- Use learning rate warmup for stable training

28.5 Debugging and Troubleshooting

28.5.1 Common Training Issues

Problem	Symptoms	Solutions
Vanishing Gradients	Training loss plateaus early	Use LSTM/GRU, gradient clipping, better initialization
Exploding Gradients	Loss becomes NaN or very large	Gradient clipping, lower learning rate
Overfitting	Large gap between train/val loss	Dropout, regularization, more data
Underfitting	Both train/val loss high	Increase model capacity, reduce regularization
Slow Convergence	Loss decreases very slowly	Increase learning rate, better optimizer

Table 4: Common Training Problems and Solutions

28.5.2 Model Diagnostic Techniques

- **Learning Curves**: Plot training/validation loss over time
- **Gradient Analysis**: Monitor gradient norms and distributions
- **Activation Analysis**: Check for dead neurons or saturation
- **Attention Visualization**: Examine attention patterns (if applicable)
- **Error Analysis**: Categorize and analyze prediction errors

29 Common Challenges and Solutions

29.1 Vanishing Gradients

Problem: Gradients become too small in long sequences, preventing learning.

Key Points:

- **Cause**: Repeated multiplication of small gradients through time

- **Impact**: Model cannot learn long-term dependencies
- **Detection**: Gradient norms decrease exponentially with sequence length

Solutions:

- Use LSTM or GRU instead of vanilla RNN
- Apply residual connections: $\mathbf{h}_t = \mathbf{h}_{t-1} + f(\mathbf{h}_{t-1}, \mathbf{x}_t)$
- Use gradient clipping: $\mathbf{g} \leftarrow \text{clip}(\mathbf{g}, \theta)$
- Initialize weights carefully (Xavier/He initialization)
- Consider skip connections between non-adjacent time steps

Key Formulae:

- **Xavier Initialization**: $\mathbf{W} \sim \mathcal{N}(0, \frac{2}{n_{\text{in}}+n_{\text{out}}})$
- **He Initialization**: $\mathbf{W} \sim \mathcal{N}(0, \frac{2}{n_{\text{in}}})$
- **Gradient Norm**: $||\nabla\mathcal{L}|| = \sqrt{\sum_i (\frac{\partial \mathcal{L}}{\partial \theta_i})^2}$

29.2 Exploding Gradients

Problem: Gradients become too large, causing unstable training.

Key Points:

- **Cause**: Repeated multiplication of large gradients
- **Impact**: Training loss oscillates or becomes NaN
- **Detection**: Sudden spikes in loss, very large gradient norms

Solutions:

- Gradient clipping (most effective):

$$\mathbf{g}_{\text{clipped}} = \begin{cases} \mathbf{g} & \text{if } ||\mathbf{g}|| \leq \theta \\ \frac{\theta \mathbf{g}}{||\mathbf{g}||} & \text{otherwise} \end{cases} \tag{321}$$

- Reduce learning rate
- Better weight initialization
- Use batch normalization or layer normalization

29.3 Overfitting

Problem: Model performs well on training but poorly on validation data.

Key Points:

- **Symptoms**: Large gap between training and validation performance
- **Causes**: Model too complex, insufficient data, memorization
- **Detection**: Monitor validation metrics during training

Solutions:

- Apply dropout regularization:

$$\mathbf{h}_{\text{train}} = \mathbf{h} \odot \mathbf{m}, \quad \mathbf{m} \sim \text{Bernoulli}(1-p) \tag{322}$$

- Use weight decay (L2 regularization):

$$\mathcal{L}_{\text{regularized}} = \mathcal{L}_{\text{data}} + \lambda \sum_i \mathbf{W}_i^2 \tag{323}$$

- Reduce model complexity (fewer parameters, layers)
- Increase training data or use data augmentation
- Early stopping based on validation performance
- Cross-validation for hyperparameter selection

29.4 Out-of-Vocabulary (OOV) Words

Problem: Model encounters words not seen during training.

Key Points:

- **Impact**: Poor performance on unseen text
- **Common in**: Domain adaptation, low-resource settings
- **Measurement**: OOV rate = $\frac{\text{Unknown tokens}}{\text{Total tokens}}$

Solutions:

- Use subword tokenization (BPE, WordPiece, SentencePiece):

$$\text{Input:} \quad \text{"unhappiness"} \tag{324}$$

$$\text{BPE:} \quad [\text{"un"}, \text{"happy"}, \text{"ness"}] \tag{325}$$

- Reserve special tokens for unknown words (`<UNK>`)
- Use character-level models for morphologically rich languages
- Apply pre-trained embeddings (Word2Vec, GloVe, FastText)
- Implement fuzzy matching for named entities

29.5 Computational Efficiency

Problem: RNNs are slow to train due to sequential nature.

Key Points:

- **Sequential Bottleneck**: Cannot parallelize across time steps
- **Memory Usage**: Storing hidden states for all time steps
- **Long Sequences**: Quadratic complexity with attention

Solutions:

- Use packed sequences for variable-length inputs:

$$\text{Efficiency Gain} = \frac{\text{Max Length} \times \text{Batch Size}}{\text{Total Valid Tokens}} \tag{326}$$

- Apply gradient accumulation for large effective batch sizes
- Consider Transformer architectures for parallelization
- Use mixed precision training (FP16):

$$\text{Memory Reduction} \approx 50\%, \quad \text{Speed Increase} \approx 30\% \tag{327}$$

- Implement efficient attention mechanisms (sparse, local, etc.)
- Use model parallelism for very large models

29.6 Data Imbalance

Problem: Unequal distribution of classes in training data.

Key Points:

- **Impact**: Model biased toward majority class
- **Metrics**: Accuracy misleading, use F1, precision, recall
- **Detection**: Check class distribution in your dataset

Solutions:

- Weighted loss functions:

$$\mathcal{L}_{\text{weighted}} = \sum_i w_{y_i} \mathcal{L}(f(\mathbf{x}_i), y_i) \tag{328}$$

- Data resampling (oversampling minority, undersampling majority)
- Synthetic data generation (SMOTE, back-translation)
- Focal loss for extreme imbalance:

$$\text{FL}(p_t) = -\alpha_t (1 - p_t)^\gamma \log(p_t) \tag{329}$$

- Ensemble methods with balanced subsets

30 Future Directions and Advanced Architectures

30.1 Transformer Architecture

While not covered in detail in this RNN-focused handbook, Transformers have largely replaced RNNs for many NLP tasks.

Key Points:

- **Self-Attention**: Process sequences in parallel
- **Positional Encoding**: Inject sequence order information
- **Scalability**: Train much larger models efficiently
- **Transfer Learning**: Pre-train on large corpora, fine-tune on tasks

30.1.1 Key Advantages over RNNs

- **Parallelization**: All positions processed simultaneously
- **Long-range Dependencies**: Direct connections between any positions
- **Interpretability**: Attention weights show model focus
- **Scalability**: Efficient training on large datasets

Key Formulae:

- **Self-Attention**: $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}})\mathbf{V}$
- **Positional Encoding**: $PE_{(pos,2i)} = \sin(\frac{pos}{10000^{2i/d}})$
- **Feed-Forward**: $\text{FFN}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$

30.2 Pre-trained Language Models

Modern NLP heavily relies on pre-trained models fine-tuned for specific tasks.

30.2.1 BERT (Bidirectional Encoder Representations from Transformers)

- **Architecture**: Bidirectional Transformer encoder
- **Pre-training**: Masked Language Model + Next Sentence Prediction
- **Fine-tuning**: Add task-specific layers and train end-to-end

30.2.2 GPT (Generative Pre-trained Transformer)

- **Architecture**: Unidirectional Transformer decoder
- **Pre-training**: Autoregressive language modeling
- **Applications**: Text generation, completion, few-shot learning

30.2.3 T5 (Text-to-Text Transfer Transformer)

- **Approach**: All NLP tasks as text-to-text
- **Architecture**: Encoder-decoder Transformer
- **Versatility**: Same model for translation, summarization, QA

Practical Pointers:

- Use pre-trained models as starting point for most NLP tasks
- Fine-tune with lower learning rates than training from scratch
- Consider prompt engineering for few-shot learning scenarios
- Monitor for catastrophic forgetting during fine-tuning

30.3 Multi-modal Learning

Integration of text with other modalities for richer understanding.

30.3.1 Vision-Language Models

- **Applications**: Image captioning, visual question answering
- **Architecture**: CNN for vision + RNN/Transformer for language
- **Attention**: Cross-modal attention between visual and textual features

30.3.2 Speech-Text Models

- **End-to-End ASR**: Direct audio to text without intermediate representations
- **Speech Synthesis**: Neural vocoders for high-quality audio generation
- **Joint Training**: Multi-task learning on speech and text

30.3.3 Multi-modal Transformers

$$\mathbf{h}_{\text{joint}} = \text{Transformer}([\mathbf{h}_{\text{text}}; \mathbf{h}_{\text{vision}}; \mathbf{h}_{\text{audio}}]) \tag{330}$$

30.4 Efficient Architectures

Developing models that maintain performance while reducing computational requirements.

30.4.1 Knowledge Distillation

Transfer knowledge from large teacher to small student model:

$$\mathcal{L}_{\text{distill}} = \alpha\mathcal{L}_{\text{CE}}(y, \hat{y}_s) + (1-\alpha)\mathcal{L}_{\text{KD}}(\hat{y}_t, \hat{y}_s) \tag{331}$$

30.4.2 Model Compression

- **Pruning**: Remove unimportant weights/neurons
- **Quantization**: Reduce precision of weights and activations
- **Low-rank Factorization**: Decompose weight matrices

30.4.3 Efficient Attention

- **Sparse Attention**: Attend to subset of positions
- **Local Attention**: Restrict attention to nearby positions
- **Hierarchical Attention**: Multi-scale attention mechanisms

31 Implementation Guidelines and Code Patterns

31.1 PyTorch Implementation Patterns

31.1.1 Basic RNN Cell

```
class BasicRNNCell(nn.Module):
    def __init__(self, input_size, hidden_size):
        super().__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.W_ih = nn.Linear(input_size, hidden_size)
        self.W_hh = nn.Linear(hidden_size, hidden_size)

    def forward(self, x, h_prev):
        return torch.tanh(self.W_ih(x) + self.W_hh(h_prev))
```

31.1.2 LSTM Implementation Tips

- Use `nn.LSTM` for standard implementations
- Set `batch_first=True` for easier tensor handling
- Initialize hidden states properly for variable-length sequences
- Use `pack_padded_sequence` for efficiency

31.1.3 Attention Mechanism

```
class Attention(nn.Module):
    def __init__(self, hidden_size):
        super().__init__()
        self.W = nn.Linear(hidden_size * 2, hidden_size)
        self.v = nn.Linear(hidden_size, 1)

    def forward(self, query, keys):
        # query: [batch, hidden_size]
        # keys: [batch, seq_len, hidden_size]
        query = query.unsqueeze(1)  # [batch, 1, hidden_size]
        energy = torch.tanh(self.W(torch.cat([
            query.expand(-1, keys.size(1), -1), keys
        ], dim=2)))
        attention = F.softmax(self.v(energy).squeeze(2), dim=1)
        context = torch.bmm(attention.unsqueeze(1), keys)
        return context.squeeze(1), attention
```

31.2 Training Loop Best Practices

31.2.1 Standard Training Loop

```
def train_epoch(model, dataloader, optimizer, criterion):
    model.train()
    total_loss = 0

    for batch in dataloader:
        optimizer.zero_grad()

        # Forward pass
        outputs = model(batch.input)
        loss = criterion(outputs, batch.target)

        # Backward pass
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        optimizer.step()

        total_loss += loss.item()

    return total_loss / len(dataloader)
```

31.2.2 Validation and Early Stopping

```
class EarlyStopping:
    def __init__(self, patience=7, min_delta=0):
        self.patience = patience
        self.min_delta = min_delta
        self.counter = 0
        self.best_loss = float(’inf’)

    def __call__(self, val_loss):
        if val_loss < self.best_loss - self.min_delta:
            self.best_loss = val_loss
            self.counter = 0
        else:
            self.counter += 1
        return self.counter >= self.patience
```

32 Key Takeaways and Summary

Key Points:

- **Preprocessing is Crucial**: Quality text preprocessing significantly impacts model performance
- **Architecture Selection**: Choose RNN variants based on task complexity and sequence length
- **Gradient Management**: Use LSTM/GRU and gradient clipping to handle gradient problems
- **Attention Mechanisms**: Improve long-range dependency modeling and interpretability
- **Evaluation Strategy**: Use appropriate metrics and multiple evaluation approaches
- **Regularization**: Essential for preventing overfitting in complex models
- **Hyperparameter Tuning**: Systematic approach to finding optimal configurations
- **Transfer Learning**: Leverage pre-trained models when available

32.1 Essential Formula Reference

Key Formulae:

- **RNN Update**: $\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b})$
- **LSTM Gates**:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \tag{332}$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \tag{333}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \tag{334}$$

- **Attention**: $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}})\mathbf{V}$
- **BLEU Score**: $\text{BLEU} = BP \cdot \exp(\sum_{n=1}^{N} w_n \log p_n)$
- **Cross-Entropy Loss**: $\mathcal{L} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$

32.2 Common Pitfalls to Avoid

- **Data Leakage**: Ensure proper train/validation/test splits

- **Inconsistent Preprocessing**: Same preprocessing for all data splits
- **Ignoring Baselines**: Always compare against simple baselines
- **Overfitting to Validation**: Use separate test set for final evaluation
- **Inadequate Error Analysis**: Understand model failures systematically
- **Premature Optimization**: Focus on model architecture before micro-optimizations
- **Ignoring Domain Knowledge**: Incorporate linguistic insights when possible

Module D Handbook

33 Applications of Generative AI

33.1 AI in Search and Recommendation Systems

Definition: Recommendation systems are information filtering systems that predict user preferences and suggest relevant items based on user behavior, item characteristics, or collaborative patterns.
Filtering Techniques:

- **Content-based filtering:** Identifies item similarity based on features such as genre, keywords, attributes, or metadata. Uses item profiles and user profiles to make recommendations.
- **Collaborative filtering:** Measures similarity between users (user-based) or items (item-based) using historical interaction data. Assumes users with similar preferences will like similar items.
- **Hybrid filtering:** Combines content-based and collaborative methods to overcome individual limitations such as cold start problems and data sparsity.
- **Knowledge-based filtering:** Uses domain knowledge and explicit user requirements to generate recommendations.
- **Demographic filtering:** Makes recommendations based on user demographic profiles such as age, gender, location, and occupation.

Key Terms:

- **Cold Start Problem:** Difficulty in making recommendations for new users or items with limited data.
- **Data Sparsity:** When user-item interaction matrix has very few ratings compared to total possible interactions.
- **Serendipity:** Ability to recommend unexpected but relevant items that users might not have discovered otherwise.
- **Diversity:** Providing varied recommendations to avoid repetitive suggestions.
- **Long Tail:** Recommending niche or less popular items to improve catalog coverage.

Similarity Metrics:

- **Cosine similarity:** $\cos(\theta) = \frac{A \cdot B}{||A|| \times ||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$
- **Pearson correlation:** $r = \frac{\sum_{i=1}^{n} (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n} (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^{n} (Y_i - \bar{Y})^2}}$
- **Jaccard similarity:** $J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$
- **Euclidean distance:** $d(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$
- **Manhattan distance:** $d(p, q) = \sum_{i=1}^{n} |p_i - q_i|$

Evaluation Metrics:

- **Mean Absolute Error (MAE):** $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
- **Root Mean Square Error (RMSE):** $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
- **Precision:** $P = \frac{|Relevant \cap Retrieved|}{|Retrieved|}$
- **Recall:** $R = \frac{|Relevant \cap Retrieved|}{|Relevant|}$
- **F1-Score:** $F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$
- **Normalized Discounted Cumulative Gain (NDCG):** $NDCG@k = \frac{DCG@k}{IDCG@k}$ where $DCG@k = \sum_{i=1}^{k}\frac{rel_i}{\log_2(i+1)}$
- **Mean Average Precision (MAP):** $MAP@k = \frac{1}{|Q|}\sum_{q=1}^{|Q|}AP@k(q)$ where $AP@k = \frac{1}{\min(m,k)}\sum_{i=1}^{k}P@i \times rel_i$
- **Mean Reciprocal Rank (MRR):** $MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$
- **Hit Rate:** $HitRate@k = \frac{\text{Number of users with at least one relevant item in top-k}}{|Users|}$

Example: For a movie recommendation system, if we recommend 5 movies to a user and 3 are relevant:

- Precision@5 = 3/5 = 0.6
- If the user had 10 total relevant movies, Recall@5 = 3/10 = 0.3
- F1@5 = 2 × (0.6 × 0.3)/(0.6 + 0.3) = 0.4

Netflix & Amazon Pipelines:

- These platforms use deep learning models (autoencoders, neural collaborative filtering), reinforcement learning for sequential recommendations, and hybrid recommendation systems.
- **Matrix Factorization:** Decomposes user-item interaction matrix into latent factors.
- **Deep Learning Approaches:** Neural Collaborative Filtering, Variational Autoencoders, Recurrent Neural Networks.
- **Multi-Armed Bandit:** Balances exploration vs exploitation in recommendation delivery.

33.2 NLP: Sentiment Analysis & Chatbots

Definition: Natural Language Processing involves computational techniques to analyze, understand, and generate human language for various applications including sentiment analysis and conversational AI.
Key Concepts:

- **Text preprocessing:** Involves lowercasing, stopword removal, tokenization, stemming, lemmatization, and noise removal.
- **Tokenization:** Process of breaking text into individual words, subwords, or characters (tokens).
- **Named Entity Recognition (NER):** Identifies and classifies named entities like persons, locations, organizations.
- **Part-of-Speech (POS) Tagging:** Assigns grammatical categories to words (noun, verb, adjective, etc.).
- **Dependency Parsing:** Analyzes grammatical structure and relationships between words.

Transformer Models:

- **BERT (Bidirectional Encoder Representations from Transformers):** Bidirectional context understanding, masked language modeling.
- **GPT (Generative Pre-trained Transformer):** Autoregressive language generation, left-to-right context.
- **T5 (Text-to-Text Transfer Transformer):** Treats all NLP tasks as text generation problems.
- **RoBERTa:** Robustly optimized BERT with improved training methodology.
- **DistilBERT:** Lightweight version of BERT with 97% performance retention.

Attention Mechanism Formula:

- **Scaled Dot-Product Attention:** $Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$
- **Multi-Head Attention:** $MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$
- Where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$

Sentiment Analysis:

- **Classification levels:** Document-level, sentence-level, aspect-level sentiment analysis.
- **Approaches:** Lexicon-based (using sentiment dictionaries), machine learning (SVM, Naive Bayes), deep learning (LSTM, CNN, Transformers).
- **Challenges:** Sarcasm detection, context dependency, domain adaptation, multilingual sentiment analysis.

- **Applications:** Social media monitoring, product reviews, customer feedback analysis, brand monitoring.

Chatbot Types:

- **Rule-based chatbots:** Use predefined rules, decision trees, and if-else logic. Limited flexibility but predictable responses.
- **Retrieval-based chatbots:** Select responses from predefined repository using similarity matching.
- **Generative chatbots:** Create new responses using language models like GPT, T5.
- **Hybrid chatbots:** Combine rule-based and AI approaches for optimal performance.

Key NLP Metrics:

- **BLEU Score:** $BLEU = BP \times \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$ for translation quality.
- **ROUGE Score:** Measures overlap between generated and reference summaries.
- **Perplexity:** $PP = \exp\left(-\frac{1}{N}\sum_{i=1}^{N} \log p(w_i)\right)$ where lower values indicate better language model performance. For a probability distribution, $PP = 2^{H(p)}$ where $H(p)$ is entropy.

33.3 AI in Computer Vision (CV)

Definition: Computer Vision enables machines to interpret and understand visual information from images and videos using deep learning and traditional image processing techniques.
Applications:

- **Healthcare:** Tumor detection in MRI/X-ray/CT scans, diabetic retinopathy screening, skin cancer diagnosis, medical image segmentation.
- **Surveillance:** Face recognition, person re-identification, anomaly detection, crowd analysis, behavior recognition.
- **Autonomous Vehicles:** Object detection, lane detection, traffic sign recognition, depth estimation.
- **Manufacturing:** Quality control, defect detection, assembly line automation.
- **Agriculture:** Crop monitoring, disease detection, yield estimation, precision farming.

Key Models:

- **CNNs (Convolutional Neural Networks):** ResNet, VGG, Inception, DenseNet, EfficientNet.

- **Object Detection:** YOLO (You Only Look Once), R-CNN, Fast R-CNN, Faster R-CNN, SSD.
- **Segmentation:** U-Net, Mask R-CNN, DeepLab, FCN (Fully Convolutional Networks).
- **Transfer learning:** Utilizes pretrained models on ImageNet for feature extraction and fine-tuning.

Key Computer Vision Metrics:

- **Intersection over Union (IoU):** $IoU = \frac{AreaofOverlap}{AreaofUnion}$
- **Mean Average Precision (mAP):** $mAP = \frac{1}{n}\sum_{i=1}^{n} AP_i$
- **Dice Coefficient:** $Dice = \frac{2|X \cap Y|}{|X|+|Y|}$
- **Peak Signal-to-Noise Ratio (PSNR):** $PSNR = 10\log_{10}\left(\frac{MAX_I^2}{MSE}\right)$

Image Processing Fundamentals:

- **Convolution Operation:** $(f * g)(t) = \sum_m f(m)g(t-m)$
- **Pooling Operations:** Max pooling, average pooling, global average pooling for dimensionality reduction.
- **Activation Functions:** ReLU, Sigmoid, Tanh, Leaky ReLU, Swish.
- **Normalization:** Batch normalization, layer normalization, instance normalization.

Teachable Machine Workflow:

- The comprehensive steps are: Data Collection → Data Preprocessing → Model Training → Validation → Model Export → Deployment → Prediction
- **Best Practices:** Balanced dataset, data augmentation, proper validation split, iterative improvement.

33.4 AI in Finance

Definition: Financial AI applies machine learning and data analytics to automate financial processes, assess risks, detect fraud, and optimize trading strategies.
Credit Risk Scoring:

- **Models:** Logistic Regression, Random Forest, XGBoost, Neural Networks, Support Vector Machines.
- **Key Features:** Income, age, credit score, debt-to-income ratio, employment history, payment history, credit utilization.
- **Risk Metrics:** Default probability, Expected Loss, Value at Risk (VaR), Credit Score ranges (300-850).

- **Regulatory Compliance:** Fair Credit Reporting Act (FCRA), Equal Credit Opportunity Act (ECOA).

Fraud Detection:

- **Anomaly Detection:** Isolation Forest, One-Class SVM, Autoencoders, Local Outlier Factor (LOF).
- **Supervised Models:** Random Forest, Logistic Regression, Gradient Boosting, Neural Networks.
- **Feature Engineering:** Transaction patterns, velocity checks, geolocation analysis, behavioral analytics.
- **Real-time Processing:** Stream processing, rule engines, ensemble methods.

Algorithmic Trading:

- **Strategies:** Mean reversion, momentum trading, arbitrage, market making, high-frequency trading.
- **Technical Indicators:** Moving averages, RSI, MACD, Bollinger Bands, Stochastic Oscillator.
- **Risk Management:** Position sizing, stop-loss, portfolio optimization, correlation analysis.

Financial Formulas:

- **Sharpe Ratio:** $S = \frac{R_p - R_f}{\sigma_p}$ where R_p is portfolio return, R_f is risk-free rate.
- **Value at Risk (VaR):** $VaR_\alpha = \inf\{l \in \mathbb{R} : P(L > l) \leq 1 - \alpha\}$
- **Black-Scholes Formula:**

$$C = S_0 N(d_1) - K e^{-rT} N(d_2) \tag{335}$$

$$d_1 = \frac{\ln(S_0/K) + (r + \sigma^2/2)T}{\sigma\sqrt{T}} \tag{336}$$

$$d_2 = d_1 - \sigma\sqrt{T} \tag{337}$$

  where C is call option price, S_0 is current stock price, K is strike price, r is risk-free rate, T is time to maturity, σ is volatility, and $N(\cdot)$ is cumulative standard normal distribution.

- **Beta Coefficient:** $\beta = \frac{Cov(R_i, R_m)}{Var(R_m)}$

33.5 AI in Agriculture

Definition: Agricultural AI leverages machine learning, computer vision, and IoT sensors to optimize farming practices, increase yields, and promote sustainable agriculture.
Vegetation Indices:

- **NDVI Formula:** $NDVI = \frac{NIR-RED}{NIR+RED}$ where NIR is Near-Infrared and RED is Red wavelength.
- **NDVI Range:** Values range from -1 to +1, with healthy vegetation typically showing 0.3-0.8.
- **Enhanced Vegetation Index (EVI):** $EVI = G \times \frac{NIR-RED}{NIR+C_1 \times RED-C_2 \times BLUE+L}$
- **Soil Adjusted Vegetation Index (SAVI):** $SAVI = \frac{NIR-RED}{NIR+RED+L} \times (1 + L)$

Yield Prediction Models:

- **Statistical Models:** Linear Regression, Multiple Regression, Polynomial Regression.
- **Machine Learning:** Random Forest, XGBoost, Support Vector Regression, Ensemble Methods.
- **Deep Learning:** LSTM for time series, CNN for image analysis, CNN-LSTM hybrid models.
- **Input Variables:** Weather data, soil properties, crop phenology, satellite imagery, historical yields.

Precision Agriculture Components:

- **GPS Technology:** Sub-meter accuracy for field mapping and variable rate applications.
- **Remote Sensing:** Satellite imagery, drone surveillance, multispectral and hyperspectral imaging.
- **IoT Sensors:** Soil moisture, temperature, pH, nutrient levels, weather stations.
- **Variable Rate Technology (VRT):** Site-specific application of seeds, fertilizers, pesticides.

Irrigation Planning:

- **Inputs:** Soil moisture content, weather forecasts, evapotranspiration rates, crop type and growth stage.
- **AI Models:** Decision Trees for rule-based systems, LSTM for temporal patterns, Reinforcement Learning for optimization.
- **Water Management:** Deficit irrigation, drip irrigation optimization, flood irrigation scheduling.
- **Efficiency Metrics:** Water Use Efficiency (WUE), Crop Water Productivity, Irrigation Efficiency.

Tools and Platforms:

- **Programming:** Python, R, MATLAB for data analysis and modeling.

- **Frameworks:** TensorFlow, PyTorch, Scikit-learn for machine learning implementation.
- **Remote Sensing:** Google Earth Engine, QGIS, ArcGIS, ENVI for geospatial analysis.
- **Cloud Platforms:** AWS, Google Cloud, Microsoft Azure for scalable computing.

33.6 AI in Smart Cities

Definition: Smart Cities integrate AI, IoT, and data analytics to optimize urban services, improve quality of life, and enhance sustainability through intelligent infrastructure management.
Traffic Signal Optimization:

- **RL Algorithms:** Deep Q-Network (DQN), Proximal Policy Optimization (PPO), Actor-Critic methods.
- **Rule-based Methods:** Fixed-time control, actuated control, adaptive control systems.
- **Optimization Objectives:** Minimize delay, reduce fuel consumption, maximize throughput, minimize emissions.
- **Sensors:** LiDAR for 3D mapping, cameras for vehicle detection, GPS for traffic monitoring, inductive loop detectors.
- **Performance Metrics:** Average delay, queue length, level of service, intersection capacity.

Energy Management:

- **Load Forecasting:** LSTM for sequential patterns, ARIMA for time series, Prophet for seasonality.
- **Smart Grid Components:** Advanced Metering Infrastructure (AMI), Demand Response (DR), Distributed Energy Resources (DER).
- **Optimization Goals:** Peak load reduction, renewable energy integration, grid stability, cost minimization.
- **Energy Storage:** Battery management systems, pumped hydro storage, compressed air energy storage.

Waste Management:

- **Fill-level Prediction:** LSTM for temporal patterns, ARIMA for forecasting, Neural Networks for complex patterns.
- **Route Optimization:** Genetic Algorithms, Ant Colony Optimization, Vehicle Routing Problem (VRP) solutions.
- **IoT Integration:** Smart bins with ultrasonic sensors, GPS tracking, real-time monitoring systems.

- **Sustainability Metrics:** Waste diversion rate, recycling efficiency, carbon footprint reduction.

Urban Planning Applications:

- **Population Dynamics:** Migration pattern analysis, demographic forecasting, urban sprawl modeling.
- **Air Quality Monitoring:** Pollutant dispersion modeling, sensor network deployment, health impact assessment.
- **Noise Management:** Sound level monitoring, traffic noise prediction, urban acoustic planning.
- **Emergency Response:** Incident detection, resource allocation, evacuation planning.

33.7 Robotics & Automation

Definition: Robotics combines AI, mechanical engineering, and control systems to create autonomous machines capable of performing complex tasks in various environments.
Path Planning Algorithms:

- **Classical Algorithms:** A* (heuristic search), Dijkstra's algorithm (shortest path), Rapidly-exploring Random Trees (RRT).
- **AI-based Algorithms:** Deep Q-Network (DQN), Proximal Policy Optimization (PPO), Asynchronous Advantage Actor-Critic (A3C).
- **Optimization Criteria:** Shortest path, minimum time, energy efficiency, safety constraints.
- **Environment Types:** Static vs dynamic, known vs unknown, continuous vs discrete.

SLAM (Simultaneous Localization and Mapping):

- **Problem Definition:** Estimating robot pose while building environment map simultaneously.
- **Approaches:** Feature-based SLAM, grid-based SLAM, particle filter SLAM.
- **Tools:** ORB-SLAM (visual SLAM), RTAB-Map (real-time appearance-based mapping), GMapping, Hector SLAM.
- **Sensors:** LiDAR, stereo cameras, RGB-D sensors, IMU (Inertial Measurement Unit).
- **Feature Extraction:** CNNs for visual features, SIFT, SURF, ORB for keypoint detection.

Control Systems:

- **Real-time Control:** LSTM for temporal dependencies, DDPG (Deep Deterministic Policy Gradient), PPO for continuous control.
- **Visual Servoing:** CNNs for object recognition, pose estimation, end-effector control.
- **Control Theory:** PID control, Model Predictive Control (MPC), adaptive control.
- **Actuator Types:** Servo motors, stepper motors, hydraulic actuators, pneumatic systems.

Robotic Applications:

- **Industrial Automation:** Assembly line robots, pick-and-place systems, welding robots.
- **Service Robots:** Healthcare assistants, cleaning robots, delivery robots, elderly care.
- **Autonomous Vehicles:** Self-driving cars, drones, underwater vehicles, space exploration.
- **Human-Robot Interaction:** Collaborative robots (cobots), social robots, telepresence systems.

Key Formulas:

- **Forward Kinematics:** $T_n^0 = \prod_{i=1}^{n} T_i^{i-1}$ (transformation matrices).
- **Jacobian Matrix:** $J = \frac{\partial f}{\partial q}$ for velocity relationships.
- **PID Control:** $u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$

33.8 AI in Learning & Creativity

Definition: AI-powered educational technology and creative tools that personalize learning experiences and generate artistic content across multiple modalities.
Educational AI Tools:

- **Language Models:** GPT for text generation, question answering, essay feedback.
- **Adaptive Learning:** Personalized curriculum, difficulty adjustment, learning path optimization.
- **Intelligent Tutoring Systems:** One-on-one instruction, immediate feedback, progress tracking.
- **Assessment Automation:** Automated grading, plagiarism detection, competency evaluation.

Creative AI Applications:

- **Image Generation:** DALL·E, Midjourney, Stable Diffusion for visual art creation.

- **Music Composition:** MusicGen, AIVA, OpenAI Jukebox for audio synthesis.
- **Text Creation:** GPT models for creative writing, poetry, storytelling.
- **Video Generation:** Runway ML, Synthesia for video content creation.
- **3D Modeling:** AI-assisted 3D object generation, texture synthesis.

Key Technologies:

- **Generative Adversarial Networks (GANs):** $\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$
- **Variational Autoencoders (VAEs):** $\mathcal{L} = -\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + KL(q_\phi(z|x)||p(z))$
- **Diffusion Models:** Denoising process for high-quality image generation.
- **Transformer Architecture:** Self-attention mechanisms for sequence modeling.

Learning Analytics:

- **Key Uses:** Adaptive quizzes, personalized recommendations, learning outcome prediction, dropout prevention.
- **Data Sources:** Clickstream data, assignment submissions, discussion forums, peer interactions.
- **Metrics:** Learning efficiency, engagement levels, knowledge retention, skill acquisition.
- **Privacy Considerations:** FERPA compliance, data anonymization, consent management.

33.9 AI for Climate & Sustainability

Definition: Application of AI technologies to address climate change challenges, environmental monitoring, and sustainable resource management.
Environmental Monitoring:

- **NDVI Applications:** Vegetation health assessment, deforestation monitoring, agricultural productivity.
- **Climate Modeling:** Weather prediction, climate change projections, extreme event forecasting.
- **Satellite Analysis:** Land use change detection, carbon sequestration estimation, biodiversity monitoring.
- **Ocean Monitoring:** Sea level rise, ocean temperature, marine ecosystem health.

Predictive Applications:

- **Deforestation Prediction:** Time series analysis, satellite image classification, risk assessment models.

- **Drought Forecasting:** Meteorological data analysis, soil moisture modeling, agricultural impact assessment.
- **Wildfire Risk:** Weather pattern analysis, fuel load assessment, evacuation planning.
- **Flood Management:** Hydrological modeling, early warning systems, infrastructure protection.

Carbon Management:

- **Carbon Footprint Calculation:** $CO_2 = Activity \times EmissionFactor$
- **Carbon Sequestration:** Forest carbon stock estimation, soil carbon monitoring.
- **Green Technology:** Renewable energy optimization, energy efficiency improvements.
- **Supply Chain Optimization:** Transportation efficiency, sustainable sourcing, waste reduction.

Key Platforms and Tools:

- **Google Earth Engine:** Planetary-scale geospatial analysis, satellite data processing.
- **Climate Data Services:** Copernicus Climate Change Service (C3S), NASA climate data.
- **Environmental APIs:** Air quality APIs, weather data services, biodiversity databases.
- **Sustainability Frameworks:** UN Sustainable Development Goals (SDGs), ESG reporting standards.

33.10 AI Bias, Fairness, and Regulation

Definition: Ensuring AI systems are fair, transparent, and free from discriminatory bias while complying with emerging regulatory frameworks.
Types of Bias:

- **Historical Bias:** Existing societal inequalities reflected in training data.
- **Representation Bias:** Underrepresentation of certain groups in datasets.
- **Measurement Bias:** Systematic errors in data collection and labeling.
- **Algorithmic Bias:** Discriminatory outcomes due to model design or optimization.
- **Confirmation Bias:** Seeking information that confirms preexisting beliefs.

Bias Mitigation Strategies:

- **Data-level:** Balanced datasets, data augmentation, synthetic data generation, representative sampling.
- **Algorithm-level:** Fair ML algorithms, adversarial debiasing, multi-objective optimization.
- **Post-processing:** Output calibration, threshold optimization, fairness-aware predictions.
- **Continuous Monitoring:** Bias detection systems, performance auditing, feedback loops.

Fairness Metrics:

- **Disparate Impact:** $\frac{P(\hat{Y}=1|A=0)}{P(\hat{Y}=1|A=1)} \geq 0.8$ where A is protected attribute.
- **Equal Opportunity:** $P(\hat{Y} = 1|Y = 1, A = 0) = P(\hat{Y} = 1|Y = 1, A = 1)$ (equal true positive rates).
- **Equalized Odds:** $P(\hat{Y} = 1|Y = y, A = 0) = P(\hat{Y} = 1|Y = y, A = 1)$ for $y \in \{0, 1\}$.
- **Demographic Parity:** $P(\hat{Y} = 1|A = 0) = P(\hat{Y} = 1|A = 1)$.
- **Fairness through Unawareness:** Excluding protected attributes from model inputs.
- **Individual Fairness:** Similar individuals should receive similar treatment.
- **Counterfactual Fairness:** Decisions remain same in counterfactual world without protected attributes.

Regulatory Frameworks:

- **EU AI Act:** Risk-based approach with prohibited, high-risk, limited-risk, and minimal-risk categories.
- **India's DPDP Act (Digital Personal Data Protection):** Data protection, consent management, cross-border data transfer.
- **GDPR (General Data Protection Regulation):** Right to explanation, data portability, algorithmic transparency.
- **US Executive Order on AI:** Safety standards, security guidelines, civil rights protection.
- **IEEE Standards:** IEEE 2857 for AI system transparency, IEEE 2858 for algorithmic bias.

Explainable AI (XAI):

- **Model-Agnostic Methods:** LIME (Local Interpretable Model-agnostic Explanations), SHAP (SHapley Additive exPlanations).

- **Model-Specific Methods:** Attention visualization, gradient-based explanations, decision trees.
- **Global vs Local:** Global explanations for entire model behavior, local explanations for individual predictions.
- **Interpretability Levels:** Simulatability, decomposability, algorithmic transparency.

Governance and Ethics:

- **AI Ethics Principles:** Beneficence, non-maleficence, autonomy, justice, explicability.
- **Stakeholder Involvement:** Multi-disciplinary teams, community engagement, user feedback.
- **Risk Assessment:** Impact assessment, vulnerability analysis, mitigation strategies.
- **Audit and Compliance:** Regular bias audits, performance monitoring, regulatory compliance checks.

Technical Implementation:

- **Fairness Libraries:** Fairlearn, AIF360 (AI Fairness 360), What-If Tool, FairML.
- **Testing Frameworks:** Adversarial testing, stress testing, A/B testing for fairness.
- **Documentation:** Model cards, datasheets for datasets, algorithmic impact assessments.
- **Monitoring Systems:** Real-time bias detection, drift monitoring, performance degradation alerts.

Mathematical Foundations:

- **SHAP Values:** $\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N|-|S|-1)!}{|N|!}[v(S \cup \{i\}) - v(S)]$
- **Mutual Information:** $I(X;Y) = \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$ for feature dependence.
- **Statistical Parity Difference:** $SPD = P(\hat{Y} = 1|A = 1) - P(\hat{Y} = 1|A = 0)$
- **Calibration:** $P(Y = 1|\hat{P} = p) = p$ for all prediction probabilities p.

Industry Best Practices:

- **Design Phase:** Inclusive design, diverse development teams, bias-aware data collection.
- **Development Phase:** Fairness-aware algorithms, regular bias testing, cross-validation across groups.

- **Deployment Phase:** A/B testing, gradual rollout, monitoring systems, feedback mechanisms.
- **Maintenance Phase:** Continuous monitoring, regular retraining, stakeholder engagement, impact assessment.

Evaluation Methodologies:

- **Confusion Matrix Analysis:** Group-specific precision, recall, F1-score comparison.
- **ROC Curve Analysis:** AUC comparison across different demographic groups.
- **Calibration Plots:** Reliability diagrams to assess prediction calibration across groups.
- **Fairness-Accuracy Trade-offs:** Pareto frontier analysis, multi-objective optimization results.

Emerging Challenges:

- **Intersectionality:** Multiple protected attributes, compound discrimination effects.
- **Dynamic Fairness:** Fairness over time, concept drift, evolving societal norms.
- **Global Perspectives:** Cultural differences in fairness definitions, international compliance.
- **Foundation Models:** Large language models, generative AI, prompt engineering bias.

Case Studies and Applications:

- **Hiring Systems:** Resume screening, interview scheduling, candidate ranking fairness.
- **Criminal Justice:** Risk assessment tools, bail decisions, sentencing recommendations.
- **Healthcare:** Diagnostic algorithms, treatment recommendations, resource allocation.
- **Financial Services:** Credit scoring, insurance pricing, loan approval processes.
- **Education:** Admissions processes, student assessment, personalized learning systems.

Future Directions:

- **Federated Fairness:** Bias mitigation in distributed learning systems.
- **Causal Fairness:** Using causal inference for fair decision making.
- **Participatory AI:** Community involvement in AI system design and evaluation.

- **Algorithmic Recourse:** Actionable recommendations for individuals affected by AI decisions.
- **Fair Representation Learning:** Learning unbiased representations from biased data.

34 Elective 2: TinyML

34.1 Introduction to TinyML

34.1.1 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.

34.1.2 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.

34.1.3 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.

34.1.4 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.

34.1.5 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.

34.1.6 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.

34.2 Hardware for TinyML

34.2.1 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

34.2.2 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.

34.2.3 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.

34.2.4 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.

34.3 ML Model Design for Embedded Systems

34.3.1 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)

34.3.2 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

34.3.3 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

34.3.4 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.

34.4 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.

34.4.1 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

34.4.2 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

34.4.3 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

34.4.4 Key Metrics

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

34.4.5 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

34.4.6 Applications

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

34.5 TinyML Model Deployment with Edge Impulse

34.5.1 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

34.5.2 Workflow in Edge Impulse

The development process typically follows these stages:
Step-by-Step Pipeline

1. Create an Edge Impulse account and log in
2. Start a new project with proper naming and sensor settings
3. Collect Data:
   (a) Use phone IMU sensors for experiments like fall detection
   (b) Scan a QR code to connect phone sensors with Edge Impulse
4. Label Data:
   (a) Assign labels like ”Safe” or ”Fall” to collected samples
   (b) Ensure diversity in the dataset (e.g., 30–40 samples per class)
5. Split Data into training and test sets
6. Create Impulse:
   (a) Add signal processing block (e.g., Spectral Features)
   (b) Add learning block (e.g., Decision Tree or Neural Net Classifier)
7. Generate Features and visualize them
8. Train the Model
   (a) Use default or tuned hyperparameters
9. Model Evaluation:
   (a) Test on unseen data or live samples
10. Deploy the Model:
    (a) Export firmware library
    (b) Deploy to physical devices or simulate

34.5.3 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

34.5.4 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

34.6 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.

34.6.1 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.

34.6.2 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.

34.6.3 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.

34.6.4 Other Tools

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

34.7 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:

34.7.1 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.

34.7.2 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.

34.7.3 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.

34.7.4 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.

34.7.5 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.

34.7.6 Evaluation On Device

Finally, evaluate the model on-device:

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

34.8 Advanced Topics in TinyML

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

34.8.1 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

34.8.2 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.

34.8.3 Tiny AutoML

AutoML tools automatically:

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

Tools: Edge Impulse AutoML, Google EdgeTPU Compiler.

34.8.4 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.

34.8.5 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.

34.9 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.

34.9.1 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.

34.9.2 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.

34.9.3 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

34.9.4 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.

35 Elective 3: Internet of Things

35.1 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.

35.1.1 Sensor Classifications

- **Analogue Sensors**: These produce a continuous output signal that varies smoothly in proportion to the measured parameter.
  - Output range: Continuous voltage/current signal
  - Resolution: Theoretically infinite (limited by ADC)
  - Formula: $V_{out} = k \cdot P + V_{offset}$ where k is sensitivity, P is measured parameter
  - Example: Temperature sensor outputting 0.5V-4.5V for 0°C-100°C
  - Sensitivity: $S = \frac{4.5V - 0.5V}{100C - 0C} = 0.04V/C$
- **Digital Sensors**: These output discrete, binary signals (like ON/OFF or HIGH/LOW) or digital codes (like binary numbers).
  - Output: Binary states (0/1, LOW/HIGH) or digital words
  - Resolution: Depends on bit depth (n-bit = 2^n levels)
  - Example: 12-bit ADC provides $2^{12} = 4096$ discrete levels
  - PIR motion sensor: HIGH (5V) = motion detected, LOW (0V) = no motion
- **Active Sensors**: Require external power source and emit energy into environment.
  - Power consumption: Typically 10-100mW
  - Response time: Generally faster (μs to ms)
  - Range equation for LiDAR: $d = \frac{c \cdot t}{2}$ where c = speed of light, t = time of flight
  - Examples: Ultrasonic (40kHz), LiDAR (905nm laser), Radar (24GHz)
- **Passive Sensors**: Generate output using ambient energy, no external excitation required.
  - Power consumption: Near zero (self-powered)
  - Thermocouple voltage: $V = \alpha \cdot (T_1 - T_2)$ where α is Seebeck coefficient
  - Photodiode current: $I_{ph} = R \cdot A \cdot \Phi$ where R = responsivity, A = area, Φ = incident light
  - Examples: Thermocouples ($\alpha \approx 40\mu V/C$), Solar cells, Piezoelectric sensors

35.1.2 Sensor Performance Metrics

- **Accuracy**: $Accuracy = 100\% - \frac{|Reading - True\_Value|}{True\_Value} \times 100\%$
- **Precision**: $Precision = 100\% - \frac{\sigma}{Mean} \times 100\%$ where σ is standard deviation
- **Linearity**: $Linearity\_Error = \frac{Max\_Deviation}{Full\_Scale\_Output} \times 100\%$
- **Hysteresis**: Maximum difference between up-scale and down-scale readings
- **Response Time**: Time to reach 63.2% (1-1/e) of final value for step input

Actuators perform the opposite function of sensors. They receive an electrical control signal and convert it into physical action or movement.

35.1.3 Actuator Types & Specifications

- **Electrical Actuators**: Convert electrical energy directly into mechanical motion.
  - DC Motor torque: $\tau = K_t \cdot I$ where K_t is torque constant, I is current
  - Stepper motor resolution: $\theta_{step} = \frac{360}{Steps\_per\_Revolution}$
  - Servo motor bandwidth: Typically 10-100 Hz
  - Efficiency: 80-95% for brushless DC motors
  - Example: NEMA 17 stepper (1.8° step angle, 200 steps/revolution)
- **Hydraulic Actuators**: Use pressurized fluid to generate powerful force and motion.
  - Force equation: $F = P \cdot A$ where P is pressure, A is piston area
  - Power: $Power = P \cdot Q$ where Q is flow rate
  - Typical pressure: 70-350 bar (1000-5000 PSI)
  - Force-to-weight ratio: 25:1 to 100:1
  - Response time: 50-200ms for industrial applications
- **Pneumatic Actuators**: Use compressed air for motion generation.
  - Operating pressure: 4-10 bar (60-150 PSI)
  - Speed: Up to 10 m/s for linear actuators
  - Force calculation: $F = P \cdot A - F_{friction} - F_{spring}$
  - Air consumption: $Q = \frac{A \cdot L \cdot f \cdot P}{60}$ (L/min) where L is stroke, f is frequency
  - Response time: 10-50ms (faster than hydraulic)

35.2 HTTP/TCP & ThingSpeak

35.2.1 HTTP Methods & Status Codes

HTTP (Hypertext Transfer Protocol) defines request methods and response codes:
Request Methods:

- **GET**: Retrieves data (idempotent, cacheable)
- **POST**: Creates new resources (non-idempotent)
- **PUT**: Updates/creates resources (idempotent)
- **DELETE**: Removes resources (idempotent)
- **PATCH**: Partial updates (non-idempotent)

Key Status Codes:

- **2xx Success**: 200 (OK), 201 (Created), 204 (No Content)
- **4xx Client Error**: 400 (Bad Request), 401 (Unauthorized), 404 (Not Found), 429 (Rate Limited)
- **5xx Server Error**: 500 (Internal Error), 502 (Bad Gateway), 503 (Service Unavailable)

HTTP Performance Metrics:

- Request size: Typically 200-800 bytes
- Response time: ¡100ms for good user experience
- Throughput: Modern servers handle 10K-100K requests/second
- Keep-alive connections reduce overhead by 50-90%

35.2.2 TCP Performance & Reliability

TCP (Transmission Control Protocol) ensures reliable data transfer:
Key Features & Formulas:

- **Sequence Numbers**: 32-bit, wraps at 2^{32} bytes
- **Window Size**: Flow control, maximum 65,535 bytes (standard)
- **Congestion Window**: $cwnd = cwnd + 1/cwnd$ (additive increase)
- **Round Trip Time**: $RTT_{smooth} = \alpha \cdot RTT_{smooth} + (1 - \alpha) \cdot RTT_{sample}$
- **Timeout**: $RTO = RTT_{smooth} + 4 \cdot RTT_{deviation}$
- **Throughput**: $Throughput \leq \frac{Window\_Size}{RTT}$

TCP States: LISTEN → SYN-SENT → SYN-RECEIVED → ESTABLISHED → FIN-WAIT → CLOSED

35.2.3 ThingSpeak Platform Details

API Specifications:

- **Update Rate**: Free tier: 15 seconds minimum, Paid: 1 second minimum
- **Data Retention**: Free: 3 months, Paid: 1+ years
- **Channels**: Up to 8 fields per channel, 4 channels (free)
- **API Endpoint**: `https://api.thingspeak.com/update`
- **Request Format**: `POST field1=value1&field2=value2&api_key=YOUR_WRITE_KEY`

HTTP Request Example:

```
POST /update HTTP/1.1
Host: api.thingspeak.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 60

api_key=XXXXXXXXXXXXXXXX&field1=23.5&field2=65.2&field3=1013.25
```

35.3 MQTT Protocol

MQTT (Message Queuing Telemetry Transport) is a lightweight publish-subscribe protocol designed for low-bandwidth IoT devices.

35.3.1 Protocol Specifications

Efficiency Comparison:

- **Header Size**: MQTT = 2 bytes minimum, HTTP = 200+ bytes typical
- **Bandwidth Usage**: MQTT uses 8x less bandwidth than HTTP
- **Connection Overhead**: MQTT persistent connection vs HTTP request/response
- **Battery Life**: 10x longer on battery-powered devices

Message Structure:

- **Fixed Header**: 2-5 bytes (Message Type + Remaining Length)
- **Variable Header**: Topic name + Message ID
- **Payload**: Actual data (up to 256MB theoretical limit)
- **Total Overhead**: As low as 2 bytes for small messages

35.3.2 QoS Levels & Performance

Quality of Service Levels:

1. **QoS 0 (At most once)**:
   - Delivery guarantee: None (fire and forget)
   - Network overhead: Minimal
   - Use case: Frequent sensor readings where loss is acceptable
   - Packet flow: PUBLISH →
2. **QoS 1 (At least once)**:
   - Delivery guarantee: Message arrives at least once
   - Network overhead: 2x (acknowledgment required)
   - Possible duplicates: Yes
   - Packet flow: PUBLISH → PUBACK
3. **QoS 2 (Exactly once)**:
   - Delivery guarantee: Message arrives exactly once
   - Network overhead: 4x (four-part handshake)
   - Use case: Financial transactions, critical commands
   - Packet flow: PUBLISH → PUBREC → PUBREL → PUBCOMP

35.3.3 Topic Structure & Wildcards

Topic Naming Convention:

- Format: `building/floor/room/device/sensor`
- Example: `factory/floor2/machine5/temperature`
- Case sensitive: `Temperature temperature`
- Max length: 65,535 characters

Wildcard Subscriptions:

- **Single level (+)**: `home/+/temperature` matches `home/kitchen/temperature`
- **Multi-level (#)**: `sensors/#` matches all topics under `sensors/`
- **Performance**: Wildcards increase broker CPU usage by 15-30%

35.3.4 MQTT Broker Specifications

HiveMQ Cloud Features:

- **Ports**: 8883 (MQTT over TLS), 8884 (WebSocket Secure)
- **Connection Limit**: 100 concurrent (free tier)
- **Message Rate**: 10 messages/second (free tier)
- **Persistence**: Session data retained for 4 hours
- **Security**: TLS 1.2+, Username/password + X.509 certificates

Keep-Alive Mechanism:

- Default interval: 60 seconds
- Formula: $Keep\_Alive = 1.5 \times Ping\_Interval$
- Network detection time: $2 \times Keep\_Alive + Network\_Timeout$
- Optimal range: 30-300 seconds depending on network stability

35.4 IoT Analytics & Data Processing

35.4.1 Data Types & Volume Metrics

IoT Data Characteristics:

- **Structured Data**:
  - Format: CSV, JSON, database records
  - Size: 10-1000 bytes per sensor reading
  - Processing speed: 1M+ records/second
  - Example: {`timestamp: 1234567890, temp: 23.5, humidity: 65.2`}
- **Unstructured Data**:
  - Format: Images, video, audio, text logs
  - Size: 100KB - 100MB per data point
  - Processing: Requires ML/AI for analysis
  - Storage: 80% of IoT data volume
- **Data Velocity**:
  - **Batch**: Hours to days latency
  - **Near real-time**: 1-30 seconds latency
  - **Real-time**: ¡100ms latency
  - **Streaming**: Continuous data flow

35.4.2 Statistical Analysis & Outlier Detection

Descriptive Statistics Formulas:

- **Mean**: $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$
- **Variance**: $\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$
- **Standard Deviation**: $\sigma = \sqrt{\sigma^2}$
- **Coefficient of Variation**: $CV = \frac{\sigma}{\bar{x}} \times 100\%$

Quartiles & Outlier Detection:

- **First Quartile (Q1)**: $Q_1 = P_{25}$ (25th percentile)
- **Second Quartile (Q2)**: $Q_2 = P_{50}$ (median)
- **Third Quartile (Q3)**: $Q_3 = P_{75}$ (75th percentile)
- **Interquartile Range**: $IQR = Q_3 - Q_1$
- **Lower Bound**: $LB = Q_1 - 1.5 \times IQR$
- **Upper Bound**: $UB = Q_3 + 1.5 \times IQR$
- **Outlier Condition**: Data point x is outlier if $x < LB$ or $x > UB$
- **Extreme Outliers**: Use $3.0 \times IQR$ instead of $1.5 \times IQR$

Alternative Outlier Detection Methods:

- **Z-Score Method**: $z = \frac{x-\bar{x}}{\sigma}$, outlier if $|z| > 3$
- **Modified Z-Score**: $M_i = 0.6745 \times \frac{x_i-\tilde{x}}{MAD}$, outlier if $|M_i| > 3.5$
- **Median Absolute Deviation**: $MAD = median(|x_i - \tilde{x}|)$

35.4.3 Analytics Categories & Applications

- **Descriptive Analytics**:
  - Purpose: Summarize historical data (”What happened?”)
  - Techniques: Aggregation, visualization, reporting
  - Metrics: Mean, median, mode, percentiles, correlations
  - Example: Monthly energy consumption dashboard showing peak usage at 7 PM
  - Processing time: Minutes to hours
- **Predictive Analytics**:
  - Purpose: Forecast future outcomes (”What will happen?”)
  - Techniques: Regression, time series, machine learning

  - Accuracy metrics: $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
  - Example: Predicting equipment failure 72 hours in advance with 85% accuracy
  - Model types: ARIMA, LSTM, Prophet, Random Forest

- **Prescriptive Analytics**:
  - Purpose: Recommend optimal actions (”What should we do?”)
  - Techniques: Optimization, simulation, decision trees
  - Output: Actionable recommendations with confidence scores
  - Example: Irrigation system adjusting water flow based on soil moisture, weather forecast, and crop growth stage
  - ROI calculation: $ROI = \frac{Benefits - Costs}{Costs} \times 100\%$

35.4.4 Machine Learning Deployment Architectures

- **Cloud ML**:
  - Processing power: 100+ TFLOPS
  - Model complexity: Deep neural networks with millions of parameters
  - Latency: 100-1000ms (including network)
  - Example: CNN for crop disease detection (ResNet-50, 23M parameters)
  - Cost model: $0.01-0.10 per inference
  - Bandwidth requirement: 1-100 Mbps

- **Edge ML**:
  - Processing power: 1-100 GFLOPS
  - Hardware: GPU, TPU, specialized AI chips
  - Latency: 1-50ms
  - Model size: 1-100MB compressed models
  - Example: Real-time face detection on security camera (MobileNet, ¡10MB)
  - Power consumption: 5-50W

- **Tiny ML**:
  - Processing power: 1-1000 MOPS
  - Memory: 1-512KB RAM, 8KB-2MB Flash
  - Power: ¡1mW inference power
  - Model size: ¡100KB (heavily quantized)
  - Example: Keyword spotting on Arduino (accuracy 90%+, 20KB model)
  - Battery life: Months to years on single charge
  - Quantization: 8-bit or even 1-bit weights

35.5 Network Graphs in IoT

35.5.1 Graph Theory Fundamentals

Basic Definitions:

- **Graph**: $G = (V, E)$ where V = vertex set, E = edge set
- **Order**: $|V|$ = number of vertices
- **Size**: $|E|$ = number of edges
- **Degree**: $deg(v)$ = number of edges incident to vertex v
- **Density**: $\rho = \frac{2|E|}{|V|(|V|-1)}$ for undirected graphs
- **Path Length**: Number of edges in shortest path between two nodes

35.5.2 Minimum Spanning Tree (MST) Algorithms

Kruskal's Algorithm:

- **Time Complexity**: $O(E \log E)$ where E = number of edges
- **Space Complexity**: $O(V)$ for Union-Find data structure
- **Steps**:
  1. Sort all edges by weight in ascending order
  2. Initialize each vertex as separate component
  3. For each edge (u, v), if u and v in different components: add edge, merge components
- **Applications**: Network infrastructure optimization, clustering

Prim's Algorithm:

- **Time Complexity**: $O(E \log V)$ with binary heap
- **Better for**: Dense graphs where $E \approx V^2$
- **Memory Usage**: Lower than Kruskal's for dense graphs

MST Properties:

- **Edge Count**: Exactly $|V| - 1$ edges
- **Uniqueness**: Unique if all edge weights are distinct
- **Cut Property**: Minimum weight edge crossing any cut is in MST
- **Cost Reduction**: Can reduce infrastructure costs by 20-40%

35.5.3 Communication Range Models

Bidirectional Communication Condition:
Two devices i and j can communicate if BOTH conditions are satisfied:

$$d(i,j) \leq \min(Tx_i, Rx_j) \tag{338}$$
$$d(i,j) \leq \min(Tx_j, Rx_i) \tag{339}$$

Where:

- $d(i,j)$ = Euclidean distance between devices i and j
- Tx_k = Transmission range of device k
- Rx_k = Reception range of device k

Distance Calculations:

- **2D Euclidean**: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
- **3D Euclidean**: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$
- **Manhattan**: $d = |x_2 - x_1| + |y_2 - y_1|$ (grid-based networks)

Signal Propagation Models:

- **Free Space**: $P_r = P_t \cdot G_t \cdot G_r \cdot \left(\frac{\lambda}{4\pi d}\right)^2$
- **Path Loss**: $PL(dB) = PL_0 + 10n \log_{10}\left(\frac{d}{d_0}\right)$
- **Link Budget**: $P_r = P_t + G_t + G_r - PL - L_{system}$

35.5.4 Network Topology Metrics

Connectivity Measures:

- **Node Connectivity**: Minimum nodes to remove to disconnect graph
- **Edge Connectivity**: Minimum edges to remove to disconnect graph
- **Algebraic Connectivity**: Second smallest eigenvalue of Laplacian matrix
- **Network Diameter**: $D = \max_{i,j} d(i,j)$ (longest shortest path)
- **Average Path Length**: $L = \frac{1}{n(n-1)} \sum_{i \neq j} d(i,j)$

Centrality Measures:

- **Degree Centrality**: $C_D(v) = \frac{deg(v)}{n-1}$
- **Betweenness Centrality**: $C_B(v) = \sum_{s \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$
- **Closeness Centrality**: $C_C(v) = \frac{n-1}{\sum_{u \in V} d(v,u)}$
- **Eigenvector Centrality**: Highest eigenvalue of adjacency matrix

35.6 IoT Network Architecture

35.6.1 Layer-Specific Performance Metrics

- **Edge Layer**:
  - **Device Density**: 10-1000 devices per gateway
  - **Data Rate**: 1-100 kbps per sensor
  - **Power Budget**: 1mW - 1W per device
  - **Communication Range**: 10m - 10km (depending on protocol)
  - **Protocols**: LoRaWAN, Zigbee, BLE, 6LoWPAN
  - **Reliability**: 99.9% uptime requirement
- **Fog Layer**:
  - **Processing Capacity**: 1-100 GFLOPS
  - **Latency Target**: ¡10ms for critical applications
  - **Storage**: 1GB - 1TB local cache
  - **Aggregation Ratio**: 100:1 to 1000:1 data reduction
  - **Geographic Coverage**: 1-50km radius
  - **Protocols**: MQTT, CoAP, HTTP/REST
- **Cloud Layer**:
  - **Processing Capacity**: 1+ TFLOPS
  - **Storage**: Petabyte scale (theoretically unlimited)
  - **Availability**: 99.99% SLA standard
  - **Global Latency**: 50-200ms depending on region
  - **Bandwidth**: 10+ Gbps backbone connections
  - **Analytics**: Complex ML models, historical trend analysis

Data Flow Optimization:

- **Compression Ratios**: 10:1 to 100:1 for sensor data
- **Filtering Efficiency**: 90%+ irrelevant data filtered at edge
- **Caching Hit Rate**: 80%+ for frequently accessed data
- **Load Balancing**: Round-robin, least connections, weighted algorithms

35.7 Protocol Performance Comparison

35.7.1 Quantitative Protocol Analysis

Performance Benchmarks:

- **MQTT Throughput**: 100K+ messages/second on standard broker
- **HTTP Requests**: 50K+ requests/second for simple GET operations
- **TCP Throughput**: Limited by network bandwidth and RTT
- **Battery Life**: MQTT (months), HTTP (days), TCP (weeks)

Metric	**MQTT**	**HTTP**	**TCP**
Min Header Size	2 bytes	200+ bytes	20 bytes
Typical Overhead	2-4%	30-60%	5-15%
Connection Model	Persistent	Request/Response	Persistent
Max Message Size	256 MB	1-8 MB	4 GB
Reliability	QoS 0-2	Request/Response	Guaranteed
Power Efficiency	Excellent	Poor	Good

35.7.2 Protocol Selection Decision Matrix

Network Bandwidth Requirements:

- **Low Bandwidth (¡10 kbps)**: MQTT preferred
- **Medium Bandwidth (10-100 kbps)**: MQTT or HTTP
- **High Bandwidth (¿100 kbps)**: Any protocol suitable

Latency Requirements:

- **Real-time (¡50ms)**: MQTT with QoS 0, UDP-based protocols
- **Near real-time (50-500ms)**: MQTT with QoS 1, HTTP
- **Non-critical (¿500ms)**: HTTP, any protocol

Device Constraints:

- **Memory Limited (¡100KB RAM)**: MQTT, CoAP
- **Processing Limited (¡10 MHz)**: MQTT, simple HTTP
- **Power Critical (¡1mW avg)**: MQTT with sleep modes
- **No Constraints**: HTTP for simplicity and debugging

35.8 Advanced IoT Analytics Techniques

35.8.1 Time Series Analysis

Stationarity Tests:

- **Augmented Dickey-Fuller Test**: H_0: Unit root exists (non-stationary)
- **KPSS Test**: H_0: Series is stationary
- **Phillips-Perron Test**: Alternative to ADF with different assumptions

ARIMA Model Components:

- **AutoRegressive (AR)**: $X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \epsilon_t$
- **Integrated (I)**: d-th order differencing for stationarity
- **Moving Average (MA)**: $X_t = c + \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}$

- **ARIMA(p,d,q)**: Combined model with parameters p, d, q

Seasonal Decomposition:

- **Additive**: $X_t = Trend_t + Seasonal_t + Residual_t$
- **Multiplicative**: $X_t = Trend_t \times Seasonal_t \times Residual_t$
- **STL Decomposition**: Seasonal and Trend decomposition using Loess
- **Fourier Transform**: $X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt$

35.8.2 Anomaly Detection Algorithms

Statistical Methods:

- **Control Charts**: $UCL = \bar{x} + 3\sigma$, $LCL = \bar{x} - 3\sigma$
- **CUSUM**: $C_t = \max(0, C_{t-1} + (x_t - \mu - k))$
- **EWMA**: $z_t = \lambda x_t + (1 - \lambda)z_{t-1}$
- **Grubbs' Test**: $G = \frac{\max|x_i - \bar{x}|}{s}$

Machine Learning Methods:

- **Isolation Forest**: Anomaly score based on path length in trees
- **One-Class SVM**: $f(x) = \text{sgn}\left(\sum_{i=1}^{n} \alpha_i K(x_i, x) - \rho\right)$
- **Local Outlier Factor**: $LOF_k(A) = \frac{\sum_{B \in N_k(A)} \frac{lrd_k(B)}{lrd_k(A)}}{|N_k(A)|}$
- **DBSCAN Clustering**: Points not in any cluster are anomalies

Deep Learning Approaches:

- **Autoencoders**: Reconstruction error as anomaly score
- **LSTM Networks**: Prediction error for time series anomalies
- **GAN-based**: Generator-discriminator framework for outlier detection
- **Variational Autoencoders**: Probabilistic encoding for anomaly scoring

35.8.3 Edge Computing Optimization

Resource Allocation Models:

- **CPU Allocation**: $\sum_{i=1}^{n} c_i \leq C_{total}$ (capacity constraint)
- **Memory Constraint**: $\sum_{i=1}^{n} m_i \leq M_{total}$
- **Energy Budget**: $\sum_{i=1}^{n} e_i \cdot t_i \leq E_{battery}$
- **Latency Optimization**: $\min \sum_{i=1}^{n} \alpha_i \cdot L_i$ (weighted latency)

Task Offloading Decision:

- **Local Processing Time**: $T_{local} = \frac{D_{input}}{f_{local}} + T_{execution}$
- **Remote Processing Time**: $T_{remote} = T_{upload} + T_{cloud} + T_{download}$
- **Energy Consumption**: $E_{local} = P_{CPU} \cdot T_{local}$, $E_{remote} = P_{radio} \cdot T_{comm}$
- **Decision Rule**: Offload if $T_{remote} + \alpha \cdot E_{remote} < T_{local} + \alpha \cdot E_{local}$

Caching Strategies:

- **Least Recently Used (LRU)**: Remove least recently accessed data
- **Least Frequently Used (LFU)**: Remove least frequently accessed data
- **Cache Hit Ratio**: $H = \frac{\text{Cache Hits}}{\text{Total Requests}}$
- **Optimal Cache Size**: Balance between hit ratio and storage cost

35.9 IoT Security & Privacy

35.9.1 Cryptographic Protocols

Symmetric Encryption:

- **AES (Advanced Encryption Standard)**: 128/192/256-bit keys
- **ChaCha20**: Stream cipher, mobile-optimized
- **Performance**: AES-128 10 cycles/byte on modern processors
- **Key Management**: Pre-shared keys, key derivation functions

Asymmetric Encryption:

- **RSA**: Key sizes 2048/3072/4096 bits
- **Elliptic Curve (ECC)**: P-256, P-384, P-521 curves
- **Performance Trade-off**: ECC-256 RSA-3072 security, 10x faster
- **Post-Quantum**: Lattice-based, hash-based signatures

Lightweight Cryptography:

- **PRESENT**: 80/128-bit block cipher for RFID
- **Grain**: Stream cipher for hardware constraints
- **PHOTON**: Hash function for 8-bit microcontrollers
- **Resource Usage**: ¡2000 gate equivalents (GE)

35.9.2 Authentication Protocols

Device Authentication:

- **X.509 Certificates**: PKI-based device identity
- **Device ID**: Unique 128-bit identifier
- **Challenge-Response**: $Response = F(Challenge, Secret\_Key)$
- **Physically Unclonable Functions (PUF)**: Hardware-based secrets

Key Agreement Protocols:

- **Diffie-Hellman**: $K = g^{ab} \bmod p$
- **ECDH**: Elliptic curve variant, more efficient
- **Session Key Lifetime**: 1-24 hours typically
- **Perfect Forward Secrecy**: New keys for each session

35.9.3 Privacy Preservation Techniques

Differential Privacy:

- **Definition**: $\Pr[M(D) \in S] \leq e^{\epsilon} \cdot \Pr[M(D') \in S]$
- **Laplace Mechanism**: $M(D) = f(D) + Lap(\frac{\Delta f}{\epsilon})$
- **Privacy Budget**: $\epsilon = 0.1$ (strong), $\epsilon = 1.0$ (weak)
- **Composition**: $\epsilon_{total} = \sum_i \epsilon_i$ for independent queries

Homomorphic Encryption:

- **Additive**: $Enc(a) + Enc(b) = Enc(a + b)$
- **Multiplicative**: $Enc(a) \times Enc(b) = Enc(a \times b)$
- **Fully Homomorphic**: Supports both addition and multiplication
- **Performance Cost**: 1000-10000x slower than plaintext operations

Secure Multi-party Computation:

- **Secret Sharing**: Split data into n shares, require k to reconstruct
- **Garbled Circuits**: Boolean circuit evaluation on encrypted inputs
- **Applications**: Federated learning, privacy-preserving analytics
- **Communication Overhead**: $O(n^2)$ for n parties

35.10 IoT Communication Protocols Detailed Comparison

35.10.1 Low-Power Wide Area Network (LPWAN) Protocols

LoRaWAN:

- **Frequency**: 433/868/915 MHz (region-dependent)
- **Range**: Up to 15 km (rural), 2-5 km (urban)
- **Data Rate**: 0.3-50 kbps (spreading factor dependent)
- **Battery Life**: 10+ years on AA batteries
- **Network Topology**: Star-of-stars (gateways to network server)
- **Device Classes**: A (lowest power), B (scheduled), C (always listening)
- **Adaptive Data Rate**: Automatically optimizes SF and power

NB-IoT (Narrowband IoT):

- **Spectrum**: 200 kHz bandwidth
- **Coverage**: +20 dB better than GSM/GPRS
- **Capacity**: 100K+ devices per cell
- **Latency**: 1.6-10 seconds
- **Power Saving Mode**: 10+ year battery life
- **Deployment**: In-band, guard-band, or standalone

Sigfox:

- **Frequency**: 868/902 MHz
- **Message Limit**: 140 uplink, 4 downlink messages/day
- **Payload Size**: 12 bytes uplink, 8 bytes downlink
- **Range**: 30-50 km (rural), 3-10 km (urban)
- **Modulation**: DBPSK (Differential Binary Phase Shift Keying)
- **Ultra-narrow band**: 100 Hz channel spacing

35.10.2 Short-Range Protocols Performance

Zigbee (IEEE 802.15.4):

- **Frequency**: 2.4 GHz globally, 915 MHz (US), 868 MHz (EU)
- **Data Rate**: 250 kbps (2.4 GHz), 40 kbps (915 MHz), 20 kbps (868 MHz)
- **Range**: 10-100 meters (power and antenna dependent)
- **Network Size**: 65,000+ nodes per network
- **Mesh Topology**: Self-healing, self-configuring
- **Power Consumption**: 1-100 mW depending on role
- **Security**: AES-128 encryption standard

Bluetooth Low Energy (BLE):

- **Frequency**: 2.4 GHz ISM band (40 channels)
- **Data Rate**: 1-2 Mbps (BLE 4.x), up to 2 Mbps (BLE 5.0+)
- **Range**: 10-100 meters (BLE 5.0 extended range)
- **Power**: ¡15 mA active, ¡3 A sleep mode
- **Connection Interval**: 7.5 ms to 4 seconds
- **Advertising Interval**: 20 ms to 10.24 seconds
- **Mesh Support**: BLE Mesh (up to 32,000 nodes)

Wi-Fi 6 (802.11ax) for IoT:

- **Frequency**: 2.4/5/6 GHz bands
- **Data Rate**: Up to 9.6 Gbps theoretical
- **OFDMA**: Orthogonal Frequency Division Multiple Access
- **Target Wake Time**: Scheduled communication for power saving
- **BSS Coloring**: Reduces interference in dense deployments
- **MU-MIMO**: Up to 8 spatial streams
- **Range Extension**: 20-35% better coverage than 802.11ac

35.11 Advanced Network Performance Analysis

35.11.1 Network Simulation and Modeling

Queuing Theory Models:

- **M/M/1 Queue**: $\rho = \frac{\lambda}{\mu}$, $L = \frac{\rho}{1-\rho}$, $W = \frac{L}{\lambda}$
- **M/M/c Queue**: Multiple servers, $P_0 = \left[\sum_{n=0}^{c-1} \frac{\rho^n}{n!} + \frac{\rho^c}{c!(1-\rho/c)}\right]^{-1}$
- **Little's Law**: $L = \lambda W$ (average items = arrival rate × average wait time)
- **Utilization**: $U = \frac{\lambda}{\mu}$ for single server systems

Network Throughput Analysis:

- **Shannon Capacity**: $C = B \log_2(1 + SNR)$ bits/second
- **ALOHA Efficiency**: Pure ALOHA = 18.4%, Slotted ALOHA = 36.8%
- **CSMA/CA Efficiency**: $S = \frac{P_{success} \cdot E[P]}{P_{success} \cdot E[P] + P_{collision} \cdot E[C] + P_{idle} \cdot \sigma}$
- **Hidden Terminal Problem**: Reduces efficiency by 10-40%

Network Reliability Models:

- **Series System**: $R_{system} = \prod_{i=1}^{n} R_i$
- **Parallel System**: $R_{system} = 1 - \prod_{i=1}^{n}(1 - R_i)$
- **k-out-of-n System**: $R = \sum_{i=k}^{n} \binom{n}{i} R^i (1-R)^{n-i}$
- **Mean Time To Failure**: $MTTF = \int_0^\infty R(t)dt$
- **Availability**: $A = \frac{MTBF}{MTBF+MTTR}$

35.11.2 Quality of Service (QoS) Metrics

Performance Indicators:

- **Throughput**: Effective data transfer rate (bps)
- **Latency**: End-to-end delay (milliseconds)
- **Jitter**: Variation in packet delay (ms standard deviation)
- **Packet Loss Rate**: $PLR = \frac{\text{Lost Packets}}{\text{Total Packets}}$
- **Error Rate**: Bit Error Rate (BER) or Packet Error Rate (PER)

QoS Classes:

- **Real-time Critical**: Latency ¡10ms, Jitter ¡1ms (industrial control)
- **Real-time Non-critical**: Latency ¡100ms (voice, video)
- **Non-real-time**: Latency ¡1s (file transfer, email)
- **Best Effort**: No guarantees (web browsing, general data)

36 Elective 4: Robotics

36.1 Introduction to Robotics

36.1.1 Definition and Domains

Robotics is the interdisciplinary study of programmable machines that sense, plan, and act. It combines mechanical engineering, electrical engineering, computer science, and artificial intelligence to create autonomous or semi-autonomous systems capable of performing tasks in dynamic environments.

Key Characteristics of Robots

- **Programmability**: Can be reprogrammed for different tasks
- **Mechanical Capability**: Physical interaction with environment
- **Flexibility**: Adaptable to various situations and tasks
- **Precision**: Accurate and repeatable operations

Applications

- **Industrial**: Welding, assembly lines, material handling, quality control, painting, packaging
  - Manufacturing robots: 6-DOF articulated arms
  - SCARA robots: Selective Compliance Assembly Robot Arm
  - Delta robots: High-speed pick-and-place operations
- **Healthcare**: Surgical robots (da Vinci system), rehabilitation devices, prosthetics, drug delivery
  - Minimally invasive surgery with enhanced precision
  - Robotic wheelchairs and mobility aids
  - Telepresence robots for remote patient monitoring
- **Defense**: UAVs (Unmanned Aerial Vehicles), UGVs (Unmanned Ground Vehicles), surveillance, bomb disposal
  - Reconnaissance and intelligence gathering
  - Search and rescue operations
  - Autonomous weapon systems (ethical concerns)
- **Service**: Cleaning robots (Roomba), delivery bots, agricultural robots, entertainment
  - Autonomous lawn mowers and vacuum cleaners
  - Restaurant service robots
  - Educational and therapeutic robots
- **Space and Exploration**: Mars rovers, space station maintenance, deep-sea exploration

- **Transportation**: Autonomous vehicles, drone delivery systems

System Components

- **Sensors**: Perceive environment
  - Proprioceptive: Internal state (encoders, IMUs, joint sensors)
  - Exteroceptive: External environment (cameras, LiDAR, ultrasonic)
  - Contact sensors: Force/torque sensors, tactile sensors
- **Actuators**: Execute motion
  - Electric: DC/AC motors, stepper motors, servo motors
  - Hydraulic: High power-to-weight ratio for heavy lifting
  - Pneumatic: Fast response, compliant motion
  - Smart materials: Shape memory alloys, piezoelectric actuators
- **Controllers**: Compute control inputs to guide behavior
  - Microcontrollers (Arduino, Raspberry Pi)
  - Real-time operating systems
  - Distributed control architectures
- **Power Systems**: Energy storage and management
- **Communication Systems**: Data transfer and networking

36.2 Sensors and Actuators

36.2.1 Sensor Types and Characteristics

Performance Metrics

- **Accuracy**: Closeness to true value
- **Precision**: Repeatability of measurements
- **Resolution**: Smallest detectable change
- **Range**: Operating limits (min/max values)
- **Bandwidth**: Frequency response characteristics
- **Noise**: Random variations in output
- **Drift**: Long-term stability

Specific Sensor Types

- **Ultrasonic Sensor**: Measures distance using sound echo
  - Operating principle: Time-of-flight measurement
  - Typical range: 2cm to 400cm

  - Beam angle: Usually 15-30 degrees
  - Limitations: Temperature sensitivity, soft surfaces, angular surfaces
  - Applications: Obstacle avoidance, liquid level sensing

- **IMU (Inertial Measurement Unit)**: Measures orientation and motion
  - Components: 3-axis gyroscope, 3-axis accelerometer, 3-axis magnetometer (9-DOF)
  - Gyroscope: Measures angular velocity (°/s)
  - Accelerometer: Measures linear acceleration (including gravity)
  - Magnetometer: Measures magnetic field strength and direction
  - Sensor fusion: Kalman filters, complementary filters
  - Drift compensation and calibration procedures
- **Encoder**: Measures shaft rotation and position
  - Incremental encoders: Relative position measurement
  - Absolute encoders: Absolute position without reference
  - Optical encoders: Light-based detection through coded disk
  - Magnetic encoders: Hall effect or magnetoresistive sensors
  - Resolution: Pulses per revolution (PPR) or counts per revolution (CPR)
- **IR Sensor**: Detects proximity using infrared light
  - Active IR: Emit and detect reflected IR light
  - Passive IR: Detect IR radiation from objects
  - Sharp GP2Y0A series: Analog distance measurement
  - Applications: Line following, obstacle detection, remote control
- **LiDAR (Light Detection and Ranging)**:
  - 2D LiDAR: Single plane scanning (SICK, Hokuyo)
  - 3D LiDAR: Multi-beam or rotating systems (Velodyne, Ouster)
  - Time-of-flight vs. phase-shift measurement
  - Point cloud data processing and filtering
- **Vision Sensors**:
  - Monocular cameras: 2D image capture
  - Stereo cameras: Depth perception through disparity
  - RGB-D cameras: Color + depth (Kinect, RealSense)
  - Event cameras: Asynchronous pixel-level change detection
- **Force/Torque Sensors**:
  - Strain gauge-based measurements
  - 6-DOF force/torque sensors
  - Applications: Assembly, manipulation, impedance control

36.2.2 Actuator Types and Control

Electric Actuators

- **DC Motor**: Provides continuous rotation
  - Brushed DC motors: Simple control, wear components
  - Brushless DC motors: Higher efficiency, longer life
  - Speed control: PWM (Pulse Width Modulation)
  - Direction control: H-bridge circuits
  - Back-EMF and motor constants (K_t, K_e)
- **Servo Motor**: Provides precise angle control
  - PWM control: 1-2ms pulse width, 50Hz frequency
  - Typical range: 0-180° or continuous rotation
  - Internal feedback loop with potentiometer
  - Digital servos: Higher resolution and faster response
- **Stepper Motor**: Precise positioning without feedback
  - Step angle: Typically 1.8° (200 steps/revolution)
  - Bipolar vs. unipolar configurations
  - Microstepping for smoother motion
  - Holding torque and detent torque characteristics

36.3 Degrees of Freedom, Joints, and Coordinate Frames

36.3.1 Degrees of Freedom (DOF)

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

Fundamental Concepts

- DOF = Number of independent motion variables (translational + rotational)
- Mobility vs. DOF: Distinction between theoretical and practical motion capability
- Redundancy: More DOF than required for task completion
- Singularities: Configurations where DOF are temporarily lost

For a 3D rigid body in free space: Maximum DOF = 6 = 3 (translation: x, y, z) + 3 (rotation: roll, pitch, yaw)

Constraint Analysis

- Each constraint removes one DOF from the system
- Holonomic constraints: Depend only on position and time
- Non-holonomic constraints: Depend on velocity (e.g., rolling without slipping)

Gruebler-Kutzbach Criterion (for planar mechanisms):

$$\text{DOF} = 3(n-1) - 2j_1 - j_2$$

Where:

- n: Number of links (including ground/base)
- j_1: Number of lower pairs (1 DOF joints: revolute, prismatic)
- j_2: Number of higher pairs (2 DOF joints: cam-follower, gear contact)

Gruebler-Kutzbach (spatial/3D version):

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

Where:

- n: Number of links (including base)
- j: Number of joints
- f_i: DOF removed by the ith joint (constraints imposed)

Common Joint Constraints

- Revolute joint: 5 constraints (1 rotational DOF remaining)
- Prismatic joint: 5 constraints (1 translational DOF remaining)
- Cylindrical joint: 4 constraints (1 rotational + 1 translational DOF)
- Spherical joint: 3 constraints (3 rotational DOF)
- Universal joint: 4 constraints (2 rotational DOF)

36.3.2 Joint Types and Characteristics

Lower Pair Joints

- **Revolute Joint (R)**: Allows rotation around a fixed axis
  - Single DOF: Angular position θ
  - Range typically limited to ±180 or continuous
  - Common in robot arms and wheeled vehicles
- **Prismatic Joint (P)**: Allows linear displacement along an axis
  - Single DOF: Linear position d
  - Range limited by physical constraints
  - Used in linear actuators and telescoping mechanisms
- **Helical Joint (H)**: Combined rotation and translation

  - Single DOF with coupled motion
  - Lead screw mechanisms
  - Constant pitch relationship

Higher Pair Joints

- Point, line, or surface contact
- Cam-follower mechanisms
- Gear contacts and belt drives
- Rolling contact constraints

36.3.3 Coordinate Frames and Transformations

Reference Frame Types

- **World Frame (Base Frame)**: Global fixed reference coordinate system
  - Origin typically at robot base or workspace center
  - Right-handed coordinate system convention
  - Fixed throughout robot operation
- **Local Frames**: Attached to each link/joint
  - Move with robot links during motion
  - Denavit-Hartenberg convention for systematic assignment
  - Tool frame: Attached to end-effector
- **Task Frame**: Application-specific coordinate systems
  - Workpiece coordinates
  - Camera coordinates for vision applications
  - Sensor-specific reference frames

Frame Assignment Rules (DH Convention)

- z_i axis: Along joint $i + 1$ axis of rotation or translation
- x_i axis: Common normal to z_{i-1} and z_i
- y_i axis: Completes right-handed coordinate system
- Origin: Intersection of x_i and z_i axes

36.4 Kinematics: Forward and Inverse

36.4.1 Denavit–Hartenberg (DH) Parameters

The DH convention provides a systematic method for assigning coordinate frames and describing the relationship between adjacent links.

Four DH Parameters per Link

- θ_i: Joint angle (rotation about z_{i-1})
- d_i: Link offset (translation along z_{i-1})
- a_i: Link length (translation along x_i)
- α_i: Link twist (rotation about x_i)

DH Transformation Sequence The transformation from frame $i-1$ to frame i:

1. Rotate about z_{i-1} by θ_i
2. Translate along z_{i-1} by d_i
3. Translate along x_i by a_i
4. Rotate about x_i by α_i

The resulting transformation matrix:

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

Modified DH Parameters

- Alternative convention with different parameter definitions
- Frame i attached to joint i instead of link i
- Widely used in robotics software (e.g., ROS, Robotics Toolbox)

36.4.2 Forward Kinematics (FK)

Forward kinematics maps joint space coordinates to Cartesian end-effector pose using homogeneous transformations.

Chain Rule for Transformations

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

Basic Transformation Matrices

Translation Matrix (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 Matrices
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:

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

General Homogeneous Transformation:

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

Where R is the rotation matrix and t is the translation vector.
Rotation Representations

- **Rotation Matrices**: 3×3 orthogonal matrices with determinant +1
- **Euler Angles**: Roll-Pitch-Yaw (ZYX convention)

  $$R = R_z(\psi)R_y(\theta)R_x(\phi)$$

- **Quaternions**: $q = [q_0, q_1, q_2, q_3]^T$ with $|q| = 1$
  - No singularities, compact representation
  - Efficient composition and interpolation
- **Axis-Angle**: Rotation by angle θ about unit vector $\mathbf{k}$

  $$R = I + \sin\theta[\mathbf{k}]_{+(1-\cos\theta)[\mathbf{k}]_2}$$

36.4.3 Inverse Kinematics (IK)

Inverse kinematics solves for joint variables given desired end-effector pose. This is generally more complex than forward kinematics.
Existence and Uniqueness

- Solutions may not exist (target outside workspace)
- Multiple solutions often exist (kinematic redundancy)

- Closed-form solutions exist for specific robot geometries

Analytical (Closed-form) Solutions

- **Geometric Approach**: Uses trigonometry and geometry
  - Decouple position and orientation problems
  - Exploit geometric relationships between joints
  - Example: 6-DOF manipulator with spherical wrist

Numerical (Iterative) Solutions

- **Jacobian-based Methods**:
  - Newton-Raphson iteration
  - Damped least squares (Levenberg-Marquardt)
  - Pseudoinverse methods for redundant manipulators
- **Optimization-based Approaches**:
  - Minimize pose error with constraints
  - Handle joint limits and obstacle avoidance
  - Multi-objective optimization for redundant systems

36.5 Control Systems and PID Feedback

36.5.1 Control System Fundamentals

System Representations

- **Transfer Functions**:
  $G(s) = \frac{Y(s)}{U(s)}$ in Laplace domain
- **State Space**:
  $\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$, $\mathbf{y} = C\mathbf{x} + D\mathbf{u}$
- **Block Diagrams**: Graphical representation of system interconnections

System Properties

- **Stability**: Bounded input produces bounded output
- **Controllability**: Ability to drive system to any state
- **Observability**: Ability to determine state from outputs
- **Steady-state Error**: Final tracking error for step inputs

36.5.2 Open vs Closed Loop Control

Open-loop Control

- No feedback from system output
- Control input determined solely by reference signal
- Advantages: Simple, no stability issues, fast response
- Disadvantages: No disturbance rejection, sensitive to parameter variations
- Applications: Stepper motor positioning, simple timing control

Closed-loop Control

- Uses output feedback to adjust control input
- Error signal: $e(t) = r(t) - y(t)$
- Advantages: Disturbance rejection, reduced sensitivity, improved accuracy
- Disadvantages: Potential instability, more complex design
- Applications: Most robotic control systems

36.5.3 PID Control

PID (Proportional-Integral-Derivative) control is the most widely used control algorithm in robotics and industrial applications.
PID Control Law The control signal is computed as:

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

Component Functions

- **Proportional (P)**: $u_p(t) = K_p e(t)$
  - Provides immediate response to current error
  - Larger K_p reduces steady-state error but may cause instability
  - Always leaves steady-state error for step inputs
- **Integral (I)**: $u_i(t) = K_i \int_0^t e(\tau) d\tau$
  - Eliminates steady-state error for step inputs
  - Accumulates past errors to provide corrective action
  - Can cause integral windup and overshoot
  - May introduce phase lag and reduce stability margins
- **Derivative (D)**: $u_d(t) = K_d \frac{de(t)}{dt}$
  - Provides anticipatory action based on error rate

- Improves transient response and stability
- Sensitive to measurement noise
- Never used alone due to pure differentiator issues

Transfer Function Representation
$G_c(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s}$
Discrete-time PID For digital implementation with sampling time T:
$u(k) = K_p e(k) + K_i T \sum_{j=0}^{k} e(j) + K_d \frac{e(k)-e(k-1)}{T}$
PID Tuning Methods

- **Ziegler-Nichols Method**:
    - Step response method: Based on process reaction curve
    - Ultimate sensitivity method: Based on critical gain and period
    - Provides initial tuning parameters
- **Cohen-Coon Method**: Improved performance for processes with significant dead time
- **Manual Tuning**: Trial and error approach
    - Start with $K_i = K_d = 0$, increase K_p until oscillation
    - Add integral action to eliminate steady-state error
    - Add derivative action to improve transient response

36.6 Introduction to ROS (Robot Operating System)

36.6.1 ROS Philosophy and Architecture

ROS is a flexible framework for writing robot software, providing tools, libraries, and conventions to simplify complex robot behavior across diverse platforms.
Core Principles

- **Peer-to-peer**: Distributed processing across multiple nodes
- **Tools-based**: Rich set of development and debugging tools
- **Multi-lingual**: Support for Python, C++, LISP, and other languages
- **Thin**: Minimal runtime overhead
- **Free and Open-source**: BSD-licensed

ROS Graph Concepts

- **Nodes**: Individual processes that perform computation
    - Single-purpose, modular programs
    - Communicate via ROS topics, services, and parameters
    - Can be written in different programming languages
    - Examples: sensor drivers, motion planners, controllers

- **Topics**: Named buses for asynchronous pub/sub communication
  - Many-to-many communication pattern
  - Typed message passing (geometry_msgs, sensor_msgs, etc.)
  - Buffered communication with configurable queue sizes
  - Examples: /cmd_vel, /scan, /camera/image_raw
- **Services**: Synchronous client-server request/response calls
  - One-to-one communication pattern
  - Blocking calls with return values
  - Suitable for remote procedure calls
  - Examples: /move_base/clear_costmaps, /gazebo/spawn_model
- **Actions**: Long-running, goal-oriented tasks with feedback
  - Asynchronous client-server pattern
  - Preemptable and provide periodic feedback
  - Three message types: Goal, Result, Feedback
  - Examples: /move_base, /arm_controller/follow_joint_trajectory
- **Parameters**: Configuration values stored on Parameter Server
  - Centralized configuration management
  - Runtime parameter modification
  - Hierarchical naming with namespaces
  - Data types: integers, floats, strings, booleans, lists, dictionaries

Master (ROS Core)

- Central coordination service for ROS nodes
- Name registration and lookup service
- Parameter server functionality
- Must be running before other ROS nodes

36.6.2 ROS Workspace and Build System

Catkin Workspace Structure

- **catkin_ws/**: Default ROS workspace directory
  - **src/**: Source code for all packages
  - **build/**: Build artifacts and intermediate files
  - **devel/**: Development space with compiled executables
  - **install/**: Installation space (optional)

Catkin Build System

- Built on top of CMake for C++ packages
- Handles dependencies between packages automatically
- Supports in-source and out-of-source builds
- Commands: `catkin_make`, `catkin build`, `catkin_make_isolated`

Environment Setup

- Source setup files: `source devel/setup.bash`
- ROS_PACKAGE_PATH and other environment variables
- Workspace overlaying for multi-workspace development

36.6.3 ROS Package Structure

Each ROS package is a directory containing related functionality.
Required Files

- **package.xml**: Package manifest with metadata
  - Package name, version, description, maintainer
  - Dependencies: build_depend, exec_depend, test_depend
  - Export information for other packages
- **CMakeLists.txt**: Build configuration for catkin
  - Find package dependencies
  - Declare ROS message/service/action files
  - Specify build targets (executables, libraries)
  - Installation rules

Common Directories

- **src/**: Source code files (C++, Python)
- **include/**: Header files for C++ libraries
- **scripts/**: Executable Python scripts
- **launch/**: XML files for node orchestration
- **msg/**: Custom message definitions
- **srv/**: Custom service definitions
- **action/**: Custom action definitions
- **config/**: Configuration files (YAML, parameters)

- **urdf/**: Robot description files
- **meshes/**: 3D model files for visualization

Launch Files

- XML format for starting multiple nodes
- Parameter setting and namespace management
- Conditional execution and argument passing
- Include other launch files for modularity
- Example launch file structure:

```
<launch>
    <arg name="robot_name" default="robot"/>
    <param name="robot_description" textfile="$(find pkg)/urdf/robot.urdf"/>
    <node name="joint_state_publisher" pkg="joint_state_publisher"
          type="joint_state_publisher"/>
    <include file="$(find other_pkg)/launch/sensors.launch"/>
</launch>
```

36.6.4 ROS Tools and Debugging

Command-line Tools

- **rostopic**: Interact with topics (list, echo, pub, hz)
- **rosservice**: Call and inspect services
- **rosnode**: Node information and management
- **rosparam**: Parameter server operations
- **rosmsg/rossrv**: Message and service type information
- **rospack**: Package management utilities
- **rosbag**: Record and playback message data

Graphical Tools

- **rviz**: 3D visualization environment
- **rqt**: Plugin-based GUI framework
- **rosrun rqt_graph**: Visual representation of ROS graph
- **rosrun rqt_plot**: Real-time data plotting
- **rosrun rqt_console**: Log message viewer

36.6.5 Path Planning Fundamentals

Problem Definition Given:

- Initial configuration q_{start}
- Goal configuration q_{goal}
- Obstacle set Q_{obs}
- Robot kinematic/dynamic constraints

Find: Collision-free path from start to goal.
Configuration Space (C-space)

- Space of all possible robot configurations
- Obstacles are "grown" by robot dimensions
- Reduces robot to a point in C-space
- Dimensionality equals robot's DOF
- Free space: $Q_{free} = Q - Q_{obs}$

Grid-based vs Sampling-based Approaches
Grid-based Methods

- Discretize C-space into regular grid
- Graph search algorithms (Dijkstra, A*)
- Complete for discretized space
- Exponential growth with dimensionality (curse of dimensionality)
- Optimal paths on discrete graph

Sampling-based Methods

- Randomly sample configurations in C-space
- Probabilistically complete
- Handle high-dimensional spaces effectively
- May not find optimal paths
- Examples: PRM (Probabilistic Roadmap), RRT (Rapidly-exploring Random Tree)

36.6.6 A* Algorithm

A* is an optimal graph search algorithm widely used for pathfinding.
Algorithm Components

- **Cost Function**: $f(n) = g(n) + h(n)$
  - $g(n)$: Actual cost from start to current node n
  - $h(n)$: Heuristic estimate of cost from n to goal
  - $f(n)$: Estimated total cost of path through n
- **Open List**: Nodes to be evaluated
- **Closed List**: Nodes already evaluated

Heuristic Functions

- Must be admissible: $h(n) \leq h^*(n)$ (never overestimate)
- Consistency: $h(n) \leq c(n, n') + h(n')$ for all neighbors
- Common heuristics:
  - Euclidean distance: $h = \sqrt{(x_g - x_n)^2 + (y_g - y_n)^2}$
  - Manhattan distance: $h = |x_g - x_n| + |y_g - y_n|$
  - Octile distance: Combination for 8-connected grids

A* Properties

- Optimal: Finds least-cost path if heuristic is admissible
- Complete: Always finds solution if one exists
- Optimally efficient: No other algorithm with same heuristic expands fewer nodes
- Time complexity: $O(b^d)$ where b is branching factor, d is depth

36.6.7 Rapidly-exploring Random Trees (RRT)

RRT is a sampling-based algorithm for motion planning in high-dimensional spaces.
Basic RRT Algorithm

1. Initialize tree T with start configuration q_{init}
2. Repeat until goal reached or max iterations:
   (a) Sample random configuration q_{rand}
   (b) Find nearest node q_{near} in tree to q_{rand}
   (c) Extend tree from q_{near} toward q_{rand} by step size Δq
   (d) If extension is collision-free, add new node q_{new} to tree
3. Extract path from start to goal if connection successful

RRT Properties

- Probabilistically complete: Will find solution if one exists (given infinite time)
- Rapidly explores large portions of search space
- Biased toward unexplored regions
- Handles kinodynamic constraints naturally
- Does not guarantee optimal paths

36.7 Simultaneous Localization and Mapping (SLAM)

36.7.1 SLAM Problem Formulation

SLAM addresses the chicken-and-egg problem: to localize, we need a map; to map, we need localization. The robot must estimate both its pose and the map of the environment simultaneously using sensor observations.
Mathematical Formulation Given:

- Control inputs: $\mathbf{u}_{1:t} = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_t\}$
- Sensor observations: $\mathbf{z}_{1:t} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_t\}$

Estimate:

- Robot trajectory: $\mathbf{x}_{0:t} = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_t\}$
- Map: $\mathbf{m} = \{\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_N\}$ (landmarks or occupancy grid)

Probabilistic Formulation Find posterior probability distribution: $P(\mathbf{x}_{0:t}, \mathbf{m}|\mathbf{z}_{1:t}, \mathbf{u}_{1:t})$
SLAM Challenges

- Data association: Which observation corresponds to which landmark?
- Loop closure: Recognizing previously visited locations
- Computational complexity: State space grows with trajectory length
- Sensor noise and uncertainty accumulation
- Dynamic environments with moving objects

36.7.2 Extended Kalman Filter SLAM (EKF-SLAM)

EKF-SLAM represents the SLAM posterior as a multivariate Gaussian distribution.
State Representation $\mathbf{x}_t = \begin{bmatrix} \mathbf{x}_t^r \\ \mathbf{m} \end{bmatrix}$
Where:

- $\mathbf{x}_t^r$: Robot pose at time t
- $\mathbf{m}$: Map (landmark positions)

Prediction Step State prediction: $\hat{\mathbf{x}}_{t|t-1} = f(\hat{\mathbf{x}}_{t-1|t-1}, \mathbf{u}_t)$
Covariance prediction: $P_{t|t-1} = F_t P_{t-1|t-1} F_t^T + G_t Q_t G_t^T$
Where:

- F_t: Jacobian of motion model with respect to state
- G_t: Jacobian of motion model with respect to control noise
- Q_t: Control noise covariance

Update Step Innovation: $\mathbf{v}_t = \mathbf{z}_t - h(\hat{\mathbf{x}}_{t|t-1})$
Innovation covariance: $S_t = H_t P_{t|t-1} H_t^T + R_t$
Kalman gain: $K_t = P_{t|t-1} H_t^T S_t^{-1}$
State update: $\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + K_t \mathbf{v}_t$
Covariance update: $P_{t|t} = (I - K_t H_t) P_{t|t-1}$
Where:

- H_t: Jacobian of observation model
- R_t: Observation noise covariance

EKF-SLAM Limitations

- Quadratic computational complexity: $O(N^2)$ per update
- Gaussian assumption may not hold
- Data association must be solved separately
- Linearization errors can cause divergence

36.7.3 Particle Filter SLAM (FastSLAM)

FastSLAM uses the Rao-Blackwellized particle filter to decompose the SLAM problem.
Key Insight Given the robot trajectory, landmark positions are conditionally independent: $P(\mathbf{x}_{1:t}, \mathbf{m}_{1:N} | \mathbf{z}_{1:t}, \mathbf{u}_{1:t}) = P(\mathbf{x}_{1:t} | \mathbf{z}_{1:t}, \mathbf{u}_{1:t}) \prod_{n=1}^{N} P(\mathbf{m}_n | \mathbf{x}_{1:t}, \mathbf{z}_{1:t}, \mathbf{u}_{1:t})$
Algorithm Structure Each particle maintains:

- Robot trajectory estimate: $\mathbf{x}_{1:t}^{[k]}$
- Set of landmark EKFs: $\{(\boldsymbol{\mu}_n^{[k]}, \Sigma_n^{[k]})\}_{n=1}^{N}$
- Particle weight: $w_t^{[k]}$

FastSLAM Steps

1. **Sampling**: Draw new robot poses from proposal distribution
2. **Data Association**: Determine landmark correspondences
3. **Measurement Update**: Update relevant landmark EKFs
4. **Weight Update**: Compute particle importance weights
5. **Resampling**: Resample particles based on weights

Advantages

- Computational complexity: $O(M \log N)$ where M is number of particles
- Handles non-Gaussian distributions
- Natural framework for data association
- Can handle loop closures better than EKF-SLAM

36.7.4 Graph-based SLAM

Graph-based SLAM represents the SLAM problem as a graph optimization.
Graph Representation

- **Nodes**: Robot poses and landmark positions
- **Edges**: Spatial constraints from odometry and observations
- **Objective**: Find node positions that best satisfy all constraints

Optimization Formulation Minimize sum of squared errors:
$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{\langle i,j \rangle \in \mathcal{E}} \mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j)^T \Omega_{ij} \mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$
Where:

- $\mathbf{e}_{ij}$: Error function between poses i and j
- Ω_{ij}: Information matrix (inverse covariance)

Solution Methods

- Gauss-Newton iteration
- Levenberg-Marquardt algorithm
- Sparse Cholesky factorization
- Incremental smoothing (iSAM)

36.8 Learning in Robotics

36.8.1 Machine Learning in Robotics

Applications of ML in Robotics

- **Perception**: Object recognition, scene understanding, semantic segmentation
- **Control**: Learning control policies, adaptive control, system identification
- **Planning**: Learning heuristics, motion primitives, task planning
- **Manipulation**: Grasping, dexterous manipulation, force control
- **Navigation**: Path planning in dynamic environments, obstacle avoidance
- **Human-Robot Interaction**: Intent recognition, natural language processing

Challenges in Robotic Learning

- **Sample Efficiency**: Real-world data collection is expensive and time-consuming
- **Safety**: Learning must occur without damaging robot or environment
- **Generalization**: Policies must work across different conditions and environments
- **Real-time Constraints**: Decisions must be made within tight time limits
- **Partial Observability**: Incomplete state information
- **Continuous Spaces**: High-dimensional state and action spaces

36.8.2 Reinforcement Learning in Robotics

Reinforcement Learning (RL) enables robots to learn optimal behavior through interaction with their environment.

Markov Decision Process (MDP) Formulation

- **State Space** $\mathcal{S}$: Robot and environment configurations
- **Action Space** $\mathcal{A}$: Available robot actions
- **Transition Model** $P(s'|s,a)$: Probability of next state
- **Reward Function** $R(s,a,s')$: Immediate feedback signal
- **Discount Factor** $\gamma \in [0,1]$: Future reward weighting

Value Functions

- **State Value**: $V^\pi(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s, \pi]$
- **Action Value**: $Q^\pi(s,a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s, A_0 = a, \pi]$

Q-Learning Algorithm Model-free temporal difference learning:
$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
Where:

- $\alpha \in (0,1]$: Learning rate
- $\gamma \in [0,1]$: Discount factor for future rewards
- r: Immediate reward
- s': Next state after taking action a in state s

Deep Reinforcement Learning

- **Deep Q-Networks (DQN)**: Neural network function approximation
  - Experience replay for stable learning
  - Target network for reduced correlation
  - Applications: Atari games, robotic manipulation

Exploration Strategies

- ϵ-greedy: Random action with probability ϵ
- Boltzmann exploration: Temperature-based action selection
- Upper Confidence Bound (UCB): Optimism in face of uncertainty
- Curiosity-driven exploration: Intrinsic motivation mechanisms

37 Elective 5: Mechanics

37.1 What is Mechanics in Robotics?

Definition: Mechanics is the branch of physics that deals with the motion of objects and the forces that cause motion. In robotics, mechanics provides the fundamental principles for understanding, designing, and controlling robotic systems.

37.1.1 Three Main Branches of Mechanics

1. Kinematics

- Describes motion without considering forces that cause it
- Involves position ($\vec{r}$), velocity ($\vec{v}$), acceleration ($\vec{a}$)
- Used for path planning and trajectory design
- Example: Analyzing joint movements of robotic arm for specific reach

2. Dynamics

- Relates motion to forces that cause it
- Core equations: $\vec{F} = m\vec{a}$ (linear), $\vec{\tau} = I\vec{\alpha}$ (rotational)
- Used in control of actuators and calculating torque
- Example: Forces needed to move robotic arm including gravity and friction

3. Statics

- Deals with systems in equilibrium (no net force or motion)
- Focuses on balance, stability, and support forces
- Used for analyzing stationary robots or balanced poses
- Example: Center of mass and support polygon for standing robot

37.1.2 Why Study Mechanics in Robotics?

- **Motion Control:** Robot movement governed by Newton's laws
- **Safety and Stability:** Prevents failures and ensures safe operation
- **Energy Efficiency:** Optimal trajectory planning and power management
- **Precision Control:** Accurate positioning and force application
- **Environmental Interaction:** Safe contact with objects and humans

38 Physical Quantities, Vectors, and Coordinate Frames

38.1 Classification of Physical Quantities

38.1.1 Scalar Quantities

- Have only magnitude, no directional information
- Examples: Mass ($m = 5$ kg), temperature ($T = 25°$C), speed ($v = 10$ m/s)
- Operations: Standard arithmetic (addition, multiplication)
- Units: SI base units (kg, K, m, s, etc.)

38.1.2 Vector Quantities

- Have both magnitude and direction in space
- Examples: Velocity, acceleration, force, torque
- Answer: "How much and in which direction?"

Vector Representation:

$$\vec{A} = A_x\hat{\imath} + A_y\hat{\jmath} + A_z\hat{k} = [A_x, A_y, A_z]^T \tag{340}$$

$$|\vec{A}| = \sqrt{A_x^2 + A_y^2 + A_z^2} \tag{341}$$

$$\hat{A} = \frac{\vec{A}}{|\vec{A}|} \tag{342}$$

Where: $\hat{\imath}, \hat{\jmath}, \hat{k}$ are unit vectors along x, y, z axes respectively

38.2 Vector Operations

38.2.1 Dot Product (Scalar Product)

$$\vec{A} \cdot \vec{B} = A_xB_x + A_yB_y + A_zB_z \tag{343}$$

$$= |\vec{A}||\vec{B}|\cos\theta \tag{344}$$

Where: θ is angle between vectors $\vec{A}$ and $\vec{B}$

Properties:

- Commutative: $\vec{A} \cdot \vec{B} = \vec{B} \cdot \vec{A}$
- Result is scalar
- $\theta = 0°$: $\vec{A} \cdot \vec{B} = AB$ (parallel)
- $\theta = 90°$: $\vec{A} \cdot \vec{B} = 0$ (perpendicular)
- Applications: Work calculation, angle between vectors

38.2.2 Cross Product (Vector Product)

$$\vec{A} \times \vec{B} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{vmatrix} \tag{345}$$

$$|\vec{A} \times \vec{B}| = |\vec{A}||\vec{B}| \sin\theta \tag{346}$$

Where: Direction given by right-hand rule

Properties:

- Anti-commutative: $\vec{A} \times \vec{B} = -\vec{B} \times \vec{A}$
- Result is vector perpendicular to both $\vec{A}$ and $\vec{B}$
- Applications: Torque, angular momentum, normal vectors

38.3 Coordinate Frame Systems

Coordinate Frame: A reference system with origin and three perpendicular axes used to describe position and orientation in 3D space.

38.3.1 Types of Coordinate Frames in Robotics

1. World Frame (Global)

- Fixed reference frame, typically ground-based
- Shared among multiple robots or systems
- Used to describe robot's environment

2. Base Frame

- Attached to robot base, moves with robot
- Serves as origin for all robot joint calculations
- Reference for robot's local coordinate system

3. Tool Frame

- Attached to end-effector, defines tool orientation
- Critical for precise control during tasks
- Used for welding, painting, picking operations

4. Joint Frames

- Local frames for each joint (Denavit-Hartenberg convention)
- Used in forward and inverse kinematics

5. User-Defined Frames

- Custom frames for specific tasks
- Useful for inclined surfaces or part-specific orientations

38.4 Coordinate System Conventions

Right-Hand Rule:

- Index finger: +x direction
- Middle finger: +y direction
- Thumb: +z direction
- For rotation: Thumb along axis, curl fingers show positive rotation

39 Transformations

39.1 What is Pose in Robotics?

Pose = Position + Orientation

Position: Location in 3D space

$$\vec{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Orientation: Rotation relative to reference frame, represented by:

- Rotation matrix (3×3)
- Euler angles (Roll, Pitch, Yaw)
- Quaternions

39.2 Translation

2D Translation:

$$\vec{p}' = \vec{p} + \vec{d} = \begin{bmatrix} x + d_x \\ y + d_y \end{bmatrix}$$

3D Translation:

$$\vec{p}' = \vec{p} + \vec{d} = \begin{bmatrix} x + d_x \\ y + d_y \\ z + d_z \end{bmatrix}$$

39.3 Rotation Matrices

39.3.1 2D Rotation

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

39.3.2 3D Rotation Matrices

Rotation about Z-axis (Yaw):

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

Rotation about X-axis (Roll):

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

Rotation about Y-axis (Pitch):

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

39.3.3 Properties of Rotation Matrices

- **Orthogonality:** $R^{-1} = R^T$
- **Determinant:** $\det(R) = +1$ (proper rotation)
- **Composition:** $R_{AC} = R_{AB} \cdot R_{BC}$ (order matters; A, B, C are reference frames)
- **Inverse rotation:** $R(-\theta) = R^T(\theta)$

39.4 Homogeneous Transformations

Purpose: Combine rotation and translation in single 4×4 matrix for efficient computation

39.4.1 Homogeneous Coordinates

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

39.4.2 Transformation Matrix Structure

$$\mathrm{T} = \begin{bmatrix} R_{3\times 3} & \vec{d}_{3\times 1} \\ \vec{0}_{1\times 3} & 1 \end{bmatrix}$$

Where: R = rotation matrix, $\vec{d}$ = translation vector

$$R_{3\times 3} \in SO(3), \quad \vec{d} \in \mathbb{R}^3, \quad \vec{0}_{1\times 3} \in \mathbb{R}^{1\times 3}$$

Here, $SO(3) = \{R \in \mathbb{R}^{3\times 3} \mid R^T R = I,\ \det(R) = +1\}$ is the special orthogonal group of 3×3 rotation matrices.

39.4.3 Chaining Transformations

$$T_{AC} = T_{AB} \cdot T_{BC}$$

Example - Car Localization: If Car A sees Car B, and Car B sees Car C, then Car A can find Car C's position using matrix multiplication.

40 Linear and Angular Motion

40.1 Linear Motion

40.1.1 Position, Velocity, and Acceleration

$$\text{Position: } \vec{r}(t) = x(t)\hat{i} + y(t)\hat{j} + z(t)\hat{k} \tag{347}$$

$$\text{Velocity: } \vec{v}(t) = \frac{d\vec{r}(t)}{dt} \tag{348}$$

$$\text{Acceleration: } \vec{a}(t) = \frac{d\vec{v}(t)}{dt} = \frac{d^2\vec{r}(t)}{dt^2} \tag{349}$$

Key Distinctions:

- **Distance:** Scalar, total path length $s = \int |\vec{v}| dt$
- **Displacement:** Vector, straight-line distance between points
- **Speed:** Scalar, $|\vec{v}(t)| = \sqrt{v_x^2 + v_y^2 + v_z^2}$
- **Velocity:** Vector with direction

40.1.2 Kinematic Equations for Constant Acceleration

$$\vec{v}(t) = \vec{v}_0 + \vec{a}t \tag{350}$$

$$\vec{r}(t) = \vec{r}_0 + \vec{v}_0 t + \frac{1}{2}\vec{a}t^2 \tag{351}$$

$$v^2 = v_0^2 + 2\vec{a}\cdot(\vec{r} - \vec{r}_0) \tag{352}$$

$$\vec{r}(t) = \vec{r}_0 + \frac{1}{2}(\vec{v}_0 + \vec{v}(t))t \tag{353}$$

40.2 Angular Motion

40.2.1 Angular Position, Velocity, and Acceleration

$$\text{Angular Position: } \theta(t) \text{ [radians]} \tag{354}$$

$$\text{Angular Velocity: } \omega(t) = \frac{d\theta(t)}{dt} \text{ [rad/s]} \tag{355}$$

$$\text{Angular Acceleration: } \alpha(t) = \frac{d\omega(t)}{dt} \text{ [rad/s}^2\text{]} \tag{356}$$

40.2.2 Measuring Angles: Degrees vs Radians

Radian Definition: Angle subtended when arc length equals radius

$$\theta = \frac{s}{r} \quad \text{where } s = \text{arc length}, r = \text{radius} \tag{357}$$

$$\text{Conversion:} \quad 1 \text{ radian} = \frac{180}{\pi} \approx 57.3^\circ \tag{358}$$

40.2.3 Angular Kinematic Equations

$$\omega(t) = \omega_0 + \alpha t \tag{359}$$

$$\theta(t) = \theta_0 + \omega_0 t + \frac{1}{2}\alpha t^2 \tag{360}$$

$$\omega^2 = \omega_0^2 + 2\alpha(\theta - \theta_0) \tag{361}$$

$$\theta(t) = \theta_0 + \frac{1}{2}(\omega_0 + \omega(t))t \tag{362}$$

40.3 Relating Linear and Angular Motion

For Pure Rolling Motion (no slipping):

$$s = r\theta \quad \text{(arc length)} \tag{363}$$

$$v = r\omega \quad \text{(tangential velocity)} \tag{364}$$

$$a_t = r\alpha \quad \text{(tangential acceleration)} \tag{365}$$

$$a_c = r\omega^2 = \frac{v^2}{r} \quad \text{(centripetal acceleration)} \tag{366}$$

Total Acceleration in Circular Motion:

$\vec{a}_{total} = \vec{a}_t + \vec{a}_c = r\alpha\hat{t} + r\omega^2\hat{n}$
Magnitude: $|\vec{a}_{total}| = r\sqrt{\alpha^2 + \omega^4}$

Where: $\hat{t}$ = tangential unit vector, $\hat{n}$ = normal unit vector

41 Newton's Laws of Motion

41.1 First Law (Law of Inertia)

Statement: An object remains at rest or moves with constant velocity unless acted upon by a net external force.

$\sum \vec{F} = 0 \Rightarrow \vec{v} = \text{constant}$

Key Concepts:

- **Inertia:** Resistance to change in motion state
- **Inertial Reference Frame:** Frame where law holds true

Robotics Applications:

- Cruise control in mobile robots
- Maintaining robot position without external forces
- Understanding sensor drift

41.2 Second Law (Fundamental Law of Dynamics)

Linear Motion: $\sum \vec{F} = m\vec{a} = m\frac{d\vec{v}}{dt} = \frac{d(m\vec{v})}{dt}$
Rotational Motion: $\sum \vec{\tau} = I\vec{\alpha} = I\frac{d\vec{\omega}}{dt}$

Where: m = mass, I = moment of inertia, $\vec{a}$ = linear acceleration, $\vec{\alpha}$ = angular acceleration

41.3 Third Law (Action-Reaction Law)

Statement: For every action, there is an equal and opposite reaction.

$$\vec{F}_{AB} = -\vec{F}_{BA}$$

Robotics Applications:

- Ground reaction forces in walking robots
- Thrust forces in flying robots
- Contact forces in manipulation tasks
- Recoil forces during fast movements

42 Moment of Inertia and Torque

42.1 Moment of Inertia

Definition: Rotational analog of mass; quantifies resistance to angular acceleration

42.1.1 Point Mass

$$I = mr^2$$

Where: m = mass, r = perpendicular distance from axis of rotation

42.1.2 System of Point Masses

$$I = \sum_{i=1}^{n} m_i r_i^2$$

42.1.3 Continuous Distribution

$$I = \int r^2 dm = \int r^2 \rho dV$$

Where: ρ = density, dV = volume element

42.1.4 Common Geometric Shapes

- **Solid cylinder:** $I = \frac{1}{2}MR^2$
- **Solid sphere:** $I = \frac{2}{5}MR^2$
- **Thin rod (center):** $I = \frac{1}{12}ML^2$
- **Thin rod (end):** $I = \frac{1}{3}ML^2$
- **Thin ring:** $I = MR^2$

42.1.5 Parallel Axis Theorem

$$I = I_{cm} + Md^2$$

Where: I = moment of inertia about the shifted axis, I_{cm} = moment of inertia about center of mass, M = total mass of the body, d = distance between axes

42.2 Torque

Definition: Measure of tendency of force to rotate object about axis or pivot point

42.2.1 Vector Definition

$\vec{\tau} = \vec{r} \times \vec{F}$
Magnitude: $|\vec{\tau}| = |\vec{r}||\vec{F}| \sin\phi = rF \sin\phi$

Where: $\vec{r}$ = position vector from axis to force application point, ϕ = angle between $\vec{r}$ and $\vec{F}$

42.2.2 Alternative Forms

- $\tau = rF_{\perp}$ (force perpendicular to lever arm)
- $\tau = r_{\perp}F$ (moment arm × force)

42.2.3 Sign Convention

- **Positive torque:** Counterclockwise rotation (right-hand rule)
- **Negative torque:** Clockwise rotation

42.2.4 Rotational Newton's Second Law

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

This is the rotational analog of $\sum \vec{F} = m\vec{a}$

43 Equilibrium and Free Body Diagrams

43.1 Mechanical Equilibrium

Definition: State of balance where opposing forces cancel out

43.1.1 Types of Equilibrium

Static Equilibrium:

$$\sum \vec{F} = 0 \quad \text{and} \quad \sum \vec{\tau} = 0$$

Object at rest with no acceleration
Dynamic Equilibrium:

$\sum \vec{F} = 0 \quad \text{and} \quad \sum \vec{\tau} = 0$

Object moving with constant velocity

43.1.2 Stability of Equilibrium

1. Stable Equilibrium:

- Restoring forces bring system back after disturbance
- Example: Ball in valley

2. Unstable Equilibrium:

- System accelerates away after disturbance
- Example: Pencil balanced on tip

3. Neutral Equilibrium:

- System stays in new position after disturbance
- Example: Ball on flat surface

43.2 Free Body Diagrams (FBD)

Definition: Simplified sketch showing all external forces and torques acting on isolated object

43.2.1 Steps to Draw FBDs

1. **Isolate object:** Draw boundary around object
2. **Identify contact surfaces:** Mark all contact points
3. **Define coordinate system:** Choose convenient orientation
4. **Add contact forces:**
   - Normal force: $\vec{F}_N$ (perpendicular to surface)
   - Friction force: $\vec{F}_f$ (parallel to surface)
   - Tension: $\vec{T}$ (along rope/cable)
5. **Add non-contact forces:**
   - Gravity: $\vec{F}_g = m\vec{g}$ (vertically downward)
   - Magnetic/electric forces if applicable
6. **Resolve angled forces:** Use trigonometry
7. **Multiple bodies:** Draw separate FBD for each

43.2.2 Support Reactions

1. Roller Support:

- Allows horizontal motion
- Provides single vertical reaction force

2. Hinge (Pin) Support:

- Allows rotation, prevents translation
- Provides two reaction forces: R_x and R_y

3. Fixed Support:

- Prevents all motion
- Provides three reactions: R_x, R_y, and moment M

44 Center of Mass and Stability

44.1 Center of Mass (COM)

Definition: Unique location representing average position of system's mass

44.1.1 Calculation Methods

Discrete System:

$\vec{r}_{COM} = \frac{\sum m_i \vec{r}_i}{\sum m_i}$

Continuous Distribution:

$\vec{r}_{COM} = \frac{1}{M_{total}} \int \vec{r} dm$

44.1.2 Center of Mass vs Center of Gravity

- **COM:** Geometric/mass-based, exists anywhere
- **COG:** Based on gravitational force, requires gravity field
- **On Earth:** COM $\approx$ COG

44.2 Support Polygon and Stability

Support Polygon: 2D area formed by projecting all ground contact points onto horizontal plane

44.2.1 Stability Condition

Stable: COM projection inside support polygon
Unstable: COM projection outside support polygon

44.2.2 Examples by Number of Legs

- **2-legged:** Line segment (highly unstable)
- **3-legged:** Triangle (minimal stable)
- **4-legged:** Quadrilateral (statically stable walking)
- **6-legged:** Polygon with high stability (tripod gait)

44.2.3 Static vs Dynamic Stability

Static Stability:

- Maintains balance without motion/control
- Purely geometric and gravity-based
- Simple but limits mobility

Dynamic Stability:

- Actively maintained during movement
- COM may temporarily leave support polygon
- Requires real-time sensing and control
- Enables agile motion

44.3 Zero Moment Point (ZMP)

Definition: Point where sum of horizontal moments due to ground reaction forces equals zero

44.3.1 ZMP Conditions

$$\sum M_x = 0 \quad \text{and} \quad \sum M_y = 0$$

Where: M_x = moment of all forces about the x-axis, M_y = moment of all forces about the y-axis

44.3.2 ZMP Equations

$$x_{ZMP} = x_{CM} - \frac{z_{CM}}{Mg}\ddot{x}_{CM} + \frac{M_y}{MgZ_{CM}} \tag{367}$$

$$y_{ZMP} = y_{CM} - \frac{z_{CM}}{Mg}\ddot{y}_{CM} - \frac{M_x}{MgZ_{CM}} \tag{368}$$

Where:

- (x_{CM}, y_{CM}, z_{CM}) = Center of mass positions
- $\ddot{x}_{CM}, \ddot{y}_{CM}$ = Center of mass accelerations
- M_x, M_y = External moments about center of mass
- M = Robot mass, g = gravitational acceleration

44.3.3 ZMP vs COM Differences

Condition	COM	ZMP
Static	COM projection ZMP	COM projection ZMP
Dynamic	COM may leave SP	ZMP must stay inside SP
Role	Mass distribution	Dynamic balance indicator
Control	Not directly used	Key for stability control

45 Robot Anatomy and Mechanical Components

45.1 Robot Anatomy - Skeleton of Manipulator

Robot Structure: Chain of rigid links connected by joints, similar to human skeleton

45.1.1 Basic Components

- **Base:** Fixed to floor or mounted on mobile platform
- **Links:** Rigid bodies connecting joints
- **Joints:** Provide motion between parts
- **Arm:** Enables mobility and reachability
- **Wrist:** Allows orientation control
- **End-effector:** Device at final link (gripper, welder, etc.)

45.1.2 Joint-Link Numbering

- **Link 0:** Base (input to Joint 1)
- **Joint i:** Connects Link (i-1) to Link i
- **Consistent numbering:** Enables systematic analysis

45.2 Types of Joints in Industrial Robots

45.2.1 Translational Joints

1. **Linear Joint (Type L):**
   - Sliding motion with parallel link axes
   - Used for linear extension/retraction
2. **Orthogonal Joint (Type O):**
   - Sliding motion with perpendicular axes
   - Used for perpendicular positioning

45.2.2 Rotational Joints

1. **Rotational Joint (Type R):**
   - Most common in industrial arms
   - Rotation axis perpendicular to both link axes
   - Like door hinge
2. **Twisting Joint (Type T):**
   - Rotation axis parallel to input/output link axes
   - Used for wrist rotation
3. **Revolving Joint (Type V):**
   - Input axis parallel to rotation axis
   - Output axis perpendicular
4. **Prismatic Joints (P):**
   - Linear motion along single axis
   - Like telescope extension
   - Precise positioning applications
5. **Universal Joints (U):**
   - Rotation about two perpendicular axes
   - Flexible coupling between shafts
6. **Spherical Joints (S):**
   - Rotation around three perpendicular axes
   - Similar to human shoulder
   - Full orientation control
7. **Cylindrical Joints (C):**
   - Combines rotation + linear sliding
   - Same axis for both motions

45.3 Common Robot Configurations

45.3.1 Cartesian (XYZ) Configuration

- **Joint structure:** Three prismatic joints (PPP)
- **Movement:** Straight lines along X, Y, Z axes
- **Workspace:** Rectangular volume
- **Applications:** Pick-and-place, CNC, 3D printing
- **Advantages:** Simple kinematics, precise motion

45.3.2 Cylindrical Configuration

- **Joint structure:** T-joint (rotation) + L-joint (vertical) + O-joint (radial)
- **Movement:** Vertical cylindrical volume
- **Applications:** Vertical stacking, loading/unloading
- **Advantages:** Good height and radial reach

45.3.3 Polar (Spherical) Configuration

- **Joint structure:** Rotating base + elevating link + extending arm
- **Movement:** Partial spherical workspace
- **Applications:** Floor manipulation, sweeping motions
- **Characteristics:** Large workspace volume

45.3.4 Jointed Arm (Articulated) Configuration

- **Joint structure:** Multiple rotary joints (RRR)
- **Components:** Waist, shoulder, elbow joints
- **Movement:** Human-like arm motion
- **Applications:** Welding, painting, assembly
- **Advantages:** High flexibility, complex poses

45.3.5 SCARA Configuration

- **Full name:** Selective Compliance Assembly Robot Arm
- **Joint structure:** Two rotary joints (vertical axes) + one prismatic (vertical)
- **Compliance:** Horizontal compliance, vertical rigidity
- **Applications:** High-speed assembly, vertical insertion
- **Advantages:** Fast, precise horizontal motion

45.3.6 Wrist Configuration

- **Purpose:** Define end-effector orientation
- **DOF:** Typically 2 or 3
- **Motions:**
    - Roll (T-joint): Rotation about arm axis
    - Pitch (R-joint): Up-down tilt
    - Yaw (R-joint): Left-right rotation

45.4 Degrees of Freedom (DOF)

Definition: Minimum number of independent variables needed to specify rigid body's position and orientation

45.4.1 Types of DOF

- **Translational DOF:** Linear motion (x, y, z directions)
- **Rotational DOF:** Angular motion (roll, pitch, yaw)
- **Total DOF:** Sum of all joint DOF in system

45.4.2 Gruebler-Kutzbach (GK) Criterion

For Spatial Mechanisms:

$$\boxed{DOF = 6(N - 1 - J) + \sum_{j=1}^{J} f_j}$$

For Planar Mechanisms:

$$\boxed{DOF = 3(N - 1 - J) + \sum_{j=1}^{J} f_j}$$

Where:

- N = number of links (including base)
- J = number of joints
- f_j = DOF of j-th joint

Example - 5-Link Planar Robot:

- $N = 5$ (4 links + 1 base)
- $J = 4$ joints
- Each joint has 1 DOF
- $DOF = 3(5 - 1 - 4) + 4 = 4$

46 Linkages and Mechanisms

46.1 What is a Linkage?

Definition: Mechanism formed by connecting two or more levers to control component movement, transmit force, or change motion direction

46.1.1 Functions of Linkages

- Change direction of force
- Convert motion types (rotary linear)
- Modify or amplify force
- Create specific motion patterns

46.1.2 Types of Motion

- **Rotating:** Continuous circular motion
- **Oscillating:** Back-and-forth motion
- **Reciprocating:** Linear back-and-forth

46.2 Common Linkage Types

46.2.1 Reverse-Motion Linkage

- **Function:** Input and output move in opposite directions
- **Design:** Fixed pivot with moving input/output links
- **Characteristics:** Can rotate through 360°
- **Applications:** Control systems requiring opposite motion

46.2.2 Push-Pull Linkage

- **Function:** Input and output move in same direction
- **Design:** Four-bar linkage configuration
- **Characteristics:** Maintains direction through 360°
- **Applications:** Synchronized linear motion systems

46.2.3 Parallel-Motion Linkage

- **Function:** Maintains parallel motion at fixed distance
- **Design:** Parallelogram structure with equidistant pivots
- **Characteristics:** Preserves geometry through rotation
- **Applications:** Pantographs, overhead cable systems

46.2.4 Bell-Crank Linkage

- **Function:** Changes motion direction by 90°
- **Design:** Two cranks bent at 90°, pinned together
- **Applications:** Bicycle brakes, manual linkages
- **Advantage:** Efficient direction conversion

46.2.5 Crank and Slider Linkage

- **Function:** Converts rotary motion to reciprocating linear motion
- **Components:** Crank (rotates) + connecting rod + slider
- **Applications:** Engines, pistons, compressors, presses
- **Advantage:** Reliable rotary-to-linear conversion

46.2.6 Four-Bar Linkage

- **Structure:** 4 links + 4 rotational joints
- **Versatility:** Basic and adaptable design
- **Applications:** Pumpjacks, folding mechanisms, vehicle suspension
- **Design factor:** Link lengths influence movement range

46.2.7 Treadle Linkage

- **Function:** Rotary input turns crank through connected rods
- **Components:** One fixed pivot + two moving pivots
- **Applications:** Windscreen wipers, sewing machines
- **Advantage:** Synchronized paired outputs

47 Gears and Transmission Systems

47.1 What are Gears?

Definition: Toothed mechanical components used to transmit torque and rotational speed between machine parts

47.1.1 Functions of Gears

- Change rotational speed
- Modify torque output
- Change direction of rotation
- Transmit power between parallel or intersecting shafts

47.2 Types of Gears

47.2.1 Spur Gears

- **Design:** Teeth parallel to axis of rotation
- **Application:** Transfer power between parallel shafts
- **Advantages:** Simple manufacture, high efficiency
- **Use:** Large gear reductions in robotics

47.2.2 Helical Gears

- **Design:** Teeth cut at angle (helix) to rotation axis
- **Advantages:** Smoother, quieter operation than spur gears
- **Load capacity:** Handle higher loads
- **Applications:** Automotive, industrial systems

47.2.3 Bevel Gears

- **Design:** Conical gears for intersecting shafts
- **Arrangement:** Typically at right angles (90°)
- **Function:** Change axis direction
- **Applications:** Mixers, crushers, differential drives

47.2.4 Worm Gears

- **Components:** Worm (screw-like shaft) + worm wheel
- **Advantages:** Large reduction ratios, compact form
- **Feature:** Self-locking capability
- **Applications:** Robot joints, grippers, steering mechanisms

47.3 Gear Ratio and Torque-Speed Relationship

47.3.1 Gear Ratio Definition

$$\boxed{\text{Gear Ratio (GR)} = \frac{\text{Teeth on Driven Gear } (T_2)}{\text{Teeth on Driving Gear } (T_1)} = \frac{T_2}{T_1}}$$

47.3.2 Gear Ratio Interpretation

Example: If $T_2 = 56$ teeth, $T_1 = 8$ teeth

$GR = \frac{56}{8} = 7 : 1$

Meaning: Driving gear rotates 7 times for driven gear to complete 1 rotation

47.3.3 Torque-Speed Trade-off

$$\tau_{driven} = GR \times \tau_{driver} \tag{369}$$

$$\omega_{driven} = \frac{\omega_{driver}}{GR} \tag{370}$$

$$P = \tau \cdot \omega = \text{constant (ideal case)} \tag{371}$$

Where: τ = torque, ω = angular velocity, GR = gear ratio, P = power transmitted (remains constant in the ideal case)
Relationships:

- **High gear ratio:** High torque, low speed
- **Low gear ratio:** High speed, low torque
- **Design choice:** Based on task requirements

Examples:

- **Lifting robot arm:** High torque, slow motion (high GR)
- **Mobile robot:** High speed, lower torque (low GR)

48 Newton-Euler Dynamics

48.1 Newton-Euler Formulation: Core Concepts

Purpose: Describes motion of rigid body by combining translational and rotational dynamics

48.1.1 Fundamental Equations

Translational Motion (Newton's 2nd Law):

$\vec{F} = m\vec{a}_C$

Where: $\vec{F}$ = net external force, m = mass, $\vec{a}_C$ = acceleration of center of mass
Rotational Motion (Euler's Equation):

$\vec{\tau} = I_C\vec{\alpha} + \vec{\omega} \times (I_C\vec{\omega})$

Where: $\vec{\tau}$ = net external torque, I_C = inertia tensor about COM, $\vec{\alpha}$ = angular acceleration, $\vec{\omega}$ = angular velocity

48.1.2 Newton-Euler in Terms of Momentum

Linear Momentum:

$\sum \vec{F} = \dot{\vec{G}} = m\frac{d\vec{v}}{dt}$, where $\vec{G} = m\vec{v}$

Angular Momentum:

$\sum \vec{M} = \dot{\vec{H}} = \frac{d(I\vec{\omega})}{dt}$, where $\vec{H} = I_c\vec{\omega}$

For planar systems: gyroscopic term $\vec{\omega} \times (I_C\vec{\omega})$ vanishes

48.2 Whole-Body Dynamics and Momentum Formulation

Concept: Newton-Euler equations correspond to the six unactuated coordinates in robot equations of motion

General Form of Equations of Motion:

$F(q(t), \dot{q}(t), \ddot{q}(t), u(t), t) = 0$

Where:

- t = time variable
- q = vector of generalized coordinates (joint angles)
- $\dot{q}$ = velocity vector (first time derivative)
- $\ddot{q}$ = acceleration vector (second time derivative)
- u = vector of control inputs (actuator commands)

Newton's Equation for Linear Motion:

$\sum_{\text{link } i} m_i\ddot{p}_i = \sum_{\text{link } i} m_i g + \sum_{\text{contact } i} f_i$

Where:

- $\ddot{p}_i$ = acceleration of center of mass of link i
- m_i = mass of link i
- g = gravitational acceleration vector
- f_i = resultant contact force on link i

Center of Mass for Robot System:

$$\boxed{mp_G = \sum_{\text{link } i} m_i p_i}$$

Where:

- $m = \sum_i m_i =$ total robot mass
- $p_G =$ position of overall center of mass
- $p_i =$ position of center of mass of link i

Linear Momentum Form:

$$\boxed{\dot{P} = m\ddot{p}_G = mg + \sum_{\text{contact } i} f_i}$$

Where: $P := m\dot{p}_G =$ linear momentum of robot
Angular Momentum Formulation:

$$\boxed{\sum_{\text{link } i} \vec{H}_i = \sum_{\text{link } i} I_{ci}\vec{\omega}_i + \sum_{\text{link } i} \vec{r}_i \times m_i \vec{v}_i}$$

Where:

- $\vec{H}_i =$ angular momentum of link i
- $I_{ci} =$ inertia tensor of link i about its center of mass
- $\vec{\omega}_i =$ angular velocity of link i
- $\vec{r}_i =$ position vector from reference point to center of mass of link i
- $\vec{v}_i =$ linear velocity of center of mass of link i

Euler's Equation for Angular Motion:

$$\boxed{\dot{\vec{H}} = \sum_{\text{contact } i} \vec{\tau}_i + \sum_{\text{link } i} \vec{r}_i \times m_i \vec{g}}$$

Combined Newton-Euler Equations: The complete system dynamics for a robot can be expressed as:

$$m\ddot{p}_G = mg + \sum_{\text{contact } i} f_i \tag{372}$$

$$\dot{H}_G = \sum_{\text{contact } i} \vec{\tau}_i + \sum_{\text{link } i} \vec{r}_i \times m_i \vec{g} \tag{373}$$

These equations represent the 6 unactuated degrees of freedom (3 translational + 3 rotational) that must be satisfied regardless of the joint actuation.

48.3 Recursive Newton-Euler Algorithm

Main Objective: Compute joint torques $\vec{\tau}$ required to achieve given joint positions, velocities, and accelerations

48.3.1 Algorithm Structure

Robot as Chain: Serial manipulator = chain of rigid links connected by joints
Two-Phase Approach:

1. **Forward Pass (Blue):** Propagate kinematics from base to end-effector
2. **Backward Pass (Red):** Propagate forces/moments from end-effector to base

48.3.2 Required Inputs

- Joint positions: θ_i
- Joint velocities: $\dot{\theta}_i$
- Joint accelerations: $\ddot{\theta}_i$
- Robot kinematics: link lengths, joint types
- Mass properties: mass, center of mass, inertia tensor

48.3.3 Notation and Symbols

Frame and Index Notation:

- $i, i+1$: Link indices
- ${}^{i}(\cdot)$: Quantity expressed in frame i
- ${}^{i+1}R_i$: Rotation matrix from frame i to $i+1$
- ${}^{i}R_{i+1}$: Rotation matrix from frame $i+1$ to i

Joint and Link Variables:

- θ_{i+1}: Joint angle
- $\dot{\theta}_{i+1}$: Joint angular velocity
- $\ddot{\theta}_{i+1}$: Joint angular acceleration
- ${}^{i}P_{i+1}$: Vector to next frame
- ${}^{i}P_{ci}$: Vector to center of mass

Dynamic Quantities:

- m_i: Mass of link i
- I_{ci}: Inertia tensor at center of mass
- ${}^{i}\omega_i$: Angular velocity

- ${}^{i}\alpha_i$: Angular acceleration
- ${}^{i}a_i$: Linear acceleration
- ${}^{i}a_{ci}$: Linear acceleration at center of mass

48.3.4 Forward Recursion: Velocities and Accelerations

Angular Velocity Propagation:

$$\boxed{{}^{i+1}\omega_{i+1} ={}^{i+1} R_i \cdot^i \omega_i + \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_{i+1} \end{bmatrix}}$$

Angular Acceleration Propagation:

$$\boxed{{}^{i+1}\alpha_{i+1} ={}^{i+1} R_i[{}^{i}\alpha_i +^i \omega_i \times \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_{i+1} \end{bmatrix}] + \begin{bmatrix} 0 \\ 0 \\ \ddot{\theta}_{i+1} \end{bmatrix}}$$

Linear Acceleration at Frame Origin:

$$\boxed{{}^{i+1}a_{i+1} ={}^{i+1} R_i[{}^{i}a_i +^i \alpha_i \times^i P_{i+1} +^i \omega_i \times ({}^{i}\omega_i \times^i P_{i+1})]}$$

Linear Acceleration at Center of Mass:

$$\boxed{{}^{i}a_{ci} =^i a_i +^i \alpha_i \times^i P_{ci} +^i \omega_i \times ({}^{i}\omega_i \times^i P_{ci})}$$

48.3.5 Inertial Forces and Moments

Inertial Force:

$$\boxed{{}^{i}F_i = m_i \cdot^i a_{ci}}$$

Inertial Moment:

$$\boxed{{}^{i}N_i = I_{ci} \cdot^i \alpha_i +^i \omega_i \times (I_{ci} \cdot^i \omega_i)}$$

48.3.6 Backward Recursion: Forces and Torques

Force Recursion:

$$\boxed{{}^{i}f_i =^i R_{i+1} \cdot^{i+1} f_{i+1} +^i F_i}$$

Moment Recursion:

$$\boxed{{}^{i}n_i =^i R_{i+1} \cdot^{i+1} n_{i+1} +^i N_i +^i P_{ci} \times^i F_i +^i P_{i+1} \times ({}^{i}R_{i+1} \cdot^{i+1} f_{i+1})}$$

Joint Torque Extraction:

$$\tau_i = [0 \quad 0 \quad 1] \cdot^i n_i$$

48.4 Complete Algorithm Summary

Initialization (Base):

$$^0\omega_0 = 0, \quad ^0\alpha_0 = 0, \quad ^0a_0 = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

Forward Recursion (per link): Compute: $\omega_{i+1}, \alpha_{i+1}, a_{i+1}, a_{ci}, F_i, N_i$
Backward Recursion (per link): Compute: f_i, n_i, τ_i
Result: Efficient inverse dynamics solution for serial manipulators

49 Energy, Work, and Power

49.1 Energy - The Capacity to Do Work

Definition: Energy is the measure of ability to do work and may be stored or transferred in various forms

Properties:

- Scalar quantity
- SI Unit: Joule (J) = kgm²/s²
- Can exist in various forms (mechanical, chemical, electrical, etc.)

49.2 Mechanical Energy Forms

49.2.1 Kinetic Energy

Translational: $KE_{trans} = \frac{1}{2}mv^2$
Rotational: $KE_{rot} = \frac{1}{2}I\omega^2$
Total for Rigid Body: $KE_{total} = \frac{1}{2}mv_{cm}^2 + \frac{1}{2}I_{cm}\omega^2$

Rolling Objects:

$$KE = \frac{1}{2}mv^2 + \frac{1}{2}I\omega^2 = \frac{1}{2}mv^2(1 + \frac{I}{mr^2})$$

49.2.2 Potential Energy

Gravitational:

$$PE_g = mgh$$

Where: h = height above reference level
Elastic (Spring):

$$PE_{elastic} = \frac{1}{2}kx^2$$

Where: k = spring constant, x = displacement from equilibrium

49.3 Work

Definition: Energy transferred via force acting over displacement

49.3.1 Work Formulations

Constant Force:

$$W = \vec{F} \cdot \vec{d} = Fd\cos\phi$$

Where: ϕ = angle between force and displacement
Variable Force:

$$W = \int_{x_1}^{x_2} F(x)dx \text{ (1D)}$$
$$W = \int_C \vec{F} \cdot d\vec{r} \text{ (3D path)}$$

Rotational Work:

$$W = \int \vec{\tau} \cdot d\vec{\theta} = \int \tau d\theta$$

49.3.2 Work-Energy Theorem

$$W_{net} = \Delta KE = KE_f - KE_i$$

49.4 Conservative and Non-Conservative Forces

Conservative Forces:

- Work independent of path (gravity, spring force)
- Energy can be fully recovered
- $W_{closed\ path} = 0$

Non-Conservative Forces:

- Work depends on path (friction, air resistance)
- Energy dissipated as heat

49.5 Conservation of Energy

Total Mechanical Energy:

$E = KE + PE =$ constant (conservative systems)

With Non-Conservative Forces:

$E_i + W_{nc} = E_f$

Where: W_{nc} = work by non-conservative forces

49.6 Power

Definition: Rate of doing work or energy transfer

49.6.1 Power Formulations

$$P_{avg} = \frac{W}{\Delta t} \tag{374}$$

$$P_{inst} = \frac{dW}{dt} = \vec{F} \cdot \vec{v} \tag{375}$$

$$P_{rot} = \vec{\tau} \cdot \vec{\omega} = \tau\omega \tag{376}$$

Units: Watt (W) = J/s = kgm^2/s^3

49.6.2 Power in Robotics

- Motor power requirements: $P_{motor} = \frac{\tau\omega}{\eta}$
- Battery life estimation: $t_{battery} = \frac{E_{battery}}{P_{average}}$
- Actuator selection criteria
- Thermal management considerations

50 Friction, Compliance, and Contact

50.1 Types of Friction

50.1.1 Static Friction

$f_s \leq \mu_s N$ (prevents motion)
$f_{s,max} = \mu_s N$ (maximum static friction)

50.1.2 Kinetic (Dynamic) Friction

$f_k = \mu_k N$ (during sliding)

50.1.3 Friction Characteristics

- **Coefficient of friction:** μ depends on material properties and surface conditions
- **Normal force:** N = force perpendicular to contact surface
- **Static friction:** Generally higher than kinetic friction ($\mu_s > \mu_k$)
- **Direction:** Always opposes relative motion or potential motion
- **Independence:** Friction force independent of contact area (for rigid bodies)
- **Temperature dependence:** Coefficients can vary with temperature

Friction Coefficient Examples:

- **Steel on steel (dry):** $\mu_s = 0.7$, $\mu_k = 0.4$
- **Rubber on concrete:** $\mu_s = 1.0$, $\mu_k = 0.8$
- **Teflon on steel:** $\mu_s = 0.04$, $\mu_k = 0.04$
- **Wood on wood:** $\mu_s = 0.5$, $\mu_k = 0.3$
- **Ice on ice:** $\mu_s = 0.1$, $\mu_k = 0.03$

50.1.4 Angle of Friction

Angle of Friction (ϕ): The angle between the normal force and the resultant of normal and friction forces

$$\tan\phi = \mu$$

Where:

- ϕ = angle of friction [radians or degrees]
- μ = coefficient of friction
- For static friction: $\phi_s = \arctan(\mu_s)$
- For kinetic friction: $\phi_k = \arctan(\mu_k)$

50.1.5 Angle of Repose

Angle of Repose (θ_r): Maximum angle at which granular material remains stable on an inclined surface

$$\theta_r = \arctan(\mu_s)$$

Properties:

- Equals the angle of static friction for the material
- Critical for robot navigation on slopes and loose surfaces
- Important in material handling and storage applications
- Varies with particle size, shape, and moisture content

50.1.6 Cone of Friction

Cone of Friction: 3D representation of all possible friction force directions at a contact point

$$F_{friction} \leq \mu N$$

Geometric Properties:

- Apex angle = 2ϕ (twice the angle of friction)
- Axis aligned with normal force direction
- Friction force vector must lie within or on the cone surface
- Essential for grasp stability analysis in robotics
- Used in contact force optimization algorithms

50.2 Rolling Friction

$$f_{rolling} = \mu_{rolling} N$$

Where:

- $f_{rolling}$ = Rolling friction force
- $\mu_{rolling}$ = Coefficient of rolling resistance
- N = Normal reaction force

Properties:

- Much smaller than sliding friction ($\mu_{rolling} << \mu_k$)
- Caused by deformation at contact point
- Important for wheel design in mobile robots
- Depends on wheel material, surface material, and wheel radius

- Rolling resistance coefficient: typically 0.001 to 0.1

Rolling Friction Applications:

- **Wheel selection:** Hard wheels on smooth surfaces for low resistance
- **Tire design:** Optimize tread pattern for traction vs efficiency
- **Bearing design:** Minimize rolling resistance in joints
- **Energy calculations:** Account for rolling losses in mobile robots

Special Case of Rolling Without Slipping in Case of Powered Robot on an Incline

For a powered robot going **up a ramp at constant speed** (no angular/linear acceleration, no rolling resistance), the traction requirement is:

$$\boxed{f_{req} = mg\sin\theta}$$

Where:

- f_{req} = Required traction force to move robot uphill
- m = Mass of the robot
- g = Acceleration due to gravity
- θ = Inclination angle of the ramp

The maximum available traction force from friction is:

$$\boxed{f_{max} = \mu_s N}$$

Where:

- f_{max} = Maximum available static friction force
- μ_s = Coefficient of static friction between wheel and surface
- N = Normal reaction force on the wheels

If all drive wheels share the normal load, $N \approx mg\cos\theta$, so the **no-slip condition** is:

$$\boxed{\mu_s \geq \tan\theta}$$

Interpretation: This condition means that the coefficient of static friction must be at least equal to the tangent of the slope angle for the robot to climb without slipping.

50.3 Advanced Friction Models

Stribeck Model:

$$\mathrm{f}() = \mathrm{f}_c + (f_s - f_c)e^{-(\dot{x}/v_s)^n} + f_v\dot{x}$$

Where:

- f_c = Coulomb friction level
- f_s = static friction level
- v_s = Stribeck velocity
- f_v = viscous friction coefficient
- n = empirical parameter (usually $n = 1$ or 2)

Dahl Model (for Stick-Slip):

$$\mathrm{df}\frac{}{dx=\sigma(1-\frac{f}{F_c}\mathrm{sign}(\dot{x}))}$$

Where σ = stiffness parameter, F_c = Coulomb friction level

50.4 Compliance and Contact Modeling

Compliance: Measure of how much an object deforms under applied force

50.4.1 Spring-Damper Model

$$F_{contact} = kx + c\dot{x}$$

Where:

- k = spring constant (stiffness) [N/m]
- c = damping coefficient [Ns/m]
- x = deformation/penetration depth [m]
- $\dot{x}$ = rate of deformation [m/s]

Stiffness Values (Typical):

- **Steel:** $10^8 - 10^{11}$ N/m
- **Aluminum:** $10^7 - 10^{10}$ N/m
- **Rubber:** $10^4 - 10^7$ N/m
- **Foam:** $10^2 - 10^5$ N/m
- **Human tissue:** $10^1 - 10^4$ N/m

50.4.2 Hertzian Contact Model

$$F = k_{hertz}\delta^{3/2}$$

Where: δ = contact deformation, k_{hertz} = Hertzian stiffness
For sphere-plane contact:

$$\mathrm{k}_{hertz} = \frac{4}{3}E^*\sqrt{R}$$

Where E^* = effective elastic modulus, R = sphere radius

50.4.3 Hunt-Crossley Model (with damping):

$$F = k\delta^n + c\delta^n\dot{\delta}$$

More realistic for impact modeling with energy dissipation

50.5 Applications in Robotics

50.5.1 Grasping and Manipulation

- Friction enables secure grasping without excessive force
- Compliance allows adaptation to object shape variations
- Contact modeling predicts grasp stability and forces
- Slip detection through force/torque sensors
- Optimal grip force: $F_{grip} > \frac{F_{task}}{\mu}$ (minimum to prevent slip)
  - F_{grip} = grip force applied by robot [N]
  - F_{task} = task force (external force on object) [N]
  - μ = coefficient of friction between gripper and object [dimensionless]
- Cone of friction determines feasible grasp configurations
- Force closure analysis uses friction cone constraints

50.5.2 Locomotion

- Friction provides traction for walking/rolling
- Compliance affects ground contact dynamics and stability
- Important for terrain adaptation and energy efficiency
- Slip prevention in mobile robots and walking machines
- Optimal wheel design for different surfaces
- Angle of repose limits climbable slope angles
- Friction cone analysis for foot placement planning

50.5.3 Safety Considerations

- Compliant contact reduces impact forces during collisions
- Friction coefficients affect slip detection and prevention
- Contact modeling enables safe human-robot interaction
- Force limiting to prevent damage to objects or humans
- Emergency stopping based on unexpected contact forces
- Angle of friction considerations for safe manipulation forces

Design Guidelines:

- **High friction:** Use for grippers, feet, drive wheels
- **Low friction:** Use for sliding joints, low-wear surfaces
- **High compliance:** Use for safe interaction, shock absorption
- **Low compliance:** Use for precise positioning, rigid structures
- **Variable compliance:** Adjustable stiffness for different tasks
- **Friction cone design:** Optimize contact geometry for desired force capabilities