CONTENTS

1. Natural Language Processing

1.1 Introduction

Natural Language Processing (NLP) is a subfield of Artificial Intelligence that enables computers to understand, interpret, and generate human language.
It combines Linguistics, Computer Science, and Machine Learning to bridge the gap between human communication and machine understanding.

Definition:

NLP is the art of teaching machines to understand human language — both text and speech.

Examples:

- Google Translate
- Siri or Alexa understanding voice commands
- ChatGPT answering natural questions

1.2 Why NLP is Important

- Enables human-like communication with computers.
- Powers real-world apps like translation, summarization, chatbots, and emotion detection.
- Helps process massive unstructured text data efficiently.
- Crucial for AI-driven decision-making systems.

1.3 Core Tasks in NLP

Task	Description	Example
Lexical Analysis	Breaking text into words (tokenization), stemming, and lemmatization	"Playing" → "play"
Syntax Analysis	Analyzing grammatical structure and parts of speech (POS tagging)	"He runs fast" → [Pronoun][Verb][Adverb]
Semantics	Understanding meaning; includes NER (Named Entity Recognition), sentiment, word sense	"Apple" → Company or fruit
Pragmatics	Understanding context and intent	"Can you open the door?" → Request, not question

Task	Description	Example
Discourse Integration	Ensures meaning is consistent across sentences	Resolving “He” → refers to “John” earlier

---

1.4 NLP Workflow

1. Data Collection – Gather text/speech data (tweets, reviews).
2. Text Preprocessing – Clean and normalize text (remove noise).
3. Feature Extraction – Convert text to numerical form (TF-IDF, word embeddings).
4. Model Training – Apply algorithms (Naive Bayes, RNNs, Transformers).
5. Evaluation – Use metrics like accuracy, BLEU score, or F1-score.

1.5 NLP Techniques & Models

- Rule-Based NLP: Relies on grammar/syntax rules.
- Statistical NLP: Uses probabilistic models like HMM (Hidden Markov Model) and CRF (Conditional Random Fields).
- Neural NLP: Uses deep learning models like RNNs and Transformers (BERT, GPT).
- Transformers Revolution: BERT and GPT transformed NLP with attention-based models.

1.6 NLP Tools and Libraries

- NLTK: For tokenization and tagging.
- spaCy: Industrial-grade NLP library.
- Gensim: Topic modeling and word embeddings.
- Transformers (Hugging Face): Pretrained models like BERT, GPT-2.
- Cloud APIs: Google Cloud NLP, AWS Comprehend for production NLP.

1.7 Challenges in NLP

- Ambiguity (same word, multiple meanings).
- Detecting sarcasm or slang.

- Handling multilingual or code-mixed text.
- Understanding long contextual references.

1.8 Real-World Applications

- Voice Assistants (Siri, Alexa)
- Text Summarization (news, reports)
- Machine Translation (Google Translate)
- Sentiment Analysis (social media)
- Search Engines (Google, Bing)

2. Sentiment Analysis

2.1 Introduction

Sentiment Analysis, also known as Opinion Mining, is a subfield of Natural Language Processing (NLP) that focuses on identifying and classifying emotions or opinions expressed in text.

Definition:

Sentiment Analysis is the process of determining whether a piece of text expresses a positive, negative, or neutral sentiment.

It helps machines understand human feelings, attitudes, and opinions — especially from social media, reviews, or feedback.

Example:

- "I love this movie!" → Positive
- "This phone is not worth the money." → Negative
- "The book was okay." → Neutral

2.2 Importance of Sentiment Analysis

- Helps companies analyze customer feedback automatically.
- Used in brand monitoring and market research.
- Detects public mood on social media (politics, products).
- Assists in decision-making for business and policy.
- Powers recommendation systems and chatbots to respond empathetically.

2.3 Types of Sentiment Analysis

Type	Description	Example
Fine-grained	Measures polarity on a scale	Very Positive, Positive, Neutral, Negative, Very Negative
Emotion Detection	Identifies emotions like joy, anger, sadness	"I'm thrilled!" → Joy
Aspect-based	Detects opinion about specific product aspects	"The camera is great, but battery is poor."
Intent Analysis	Finds the user's intent or purpose	"Cancel my order" → Negative intent
Multilingual Sentiment Analysis	Analyzes text in multiple languages	"Muy bueno" → Positive (Spanish)

2.4 Steps in Sentiment Analysis

1. Data Collection
   - Gather data from reviews, tweets, comments, etc.
2. Data Preprocessing
   - Clean data: remove noise, punctuation, stopwords.
   - Tokenize text and normalize (lowercase conversion).
3. Feature Extraction
   - Convert text into numerical features.
   - Methods: Bag of Words (BoW), TF-IDF, Word Embeddings (Word2Vec, BERT).
4. Model Training
   - Use algorithms to classify sentiments.
   - Common models: Naive Bayes, Logistic Regression, Support Vector Machine, LSTM, BERT.
5. Prediction and Evaluation
   - Evaluate using metrics: Accuracy, Precision, Recall, F1-score.

2.5 Techniques Used

Category	Techniques	Description
Lexicon-Based	Uses predefined dictionaries of positive/negative words	"good" = +1, "bad" = -1
Machine Learning-Based	Uses labeled datasets and algorithms	Model learns patterns in text

Category	Techniques	Description
Deep Learning-Based	Uses neural networks (RNN, CNN, Transformer)	Learns complex context and tone

2.6 Example

Text:

"The camera quality is excellent, but the battery drains fast."

Process:

- Split into aspects → *camera* and *battery*.
- Sentiment for camera → Positive.
- Sentiment for battery → Negative.

  Output: Mixed sentiment (Positive + Negative aspects).

2.7 Tools and Libraries

- TextBlob: Simple sentiment analysis in Python.
- VADER: Specialized for social media sentiment.
- NLTK / spaCy: General NLP frameworks.
- Transformers (BERT, RoBERTa): Deep learning-based models.
- Cloud APIs: Google NLP API, AWS Comprehend, Azure Text Analytics.

2.8 Applications

- Customer Feedback Analysis: Identify satisfaction level.
- Brand Monitoring: Understand public perception.
- Political Sentiment: Track election opinions on Twitter.
- Product Improvement: Analyze user complaints.
- Financial Forecasting: Measure investor sentiment.

2.9 Challenges

- Sarcasm: "Great! My phone just died again."
- Negation Handling: "Not bad" → actually positive.
- Domain Dependence: Word meaning changes by context.
- Multilingual Text: Mixed-language comments ("Nice movie yaar").
- Contextual Emotions: Same word may mean differently based on context.

2.10 Experiential Learning

Lexicon Based Method

```
sentences = [
    "I love this movie!",
    "I hate this food.",
    "The product is okay, not great.",
    "What a fantastic day!"
]
```

Textblob Method

```
from textblob import TextBlob
for s in sentences:
    blob = TextBlob(s)
    score = blob.sentiment.polarity

    if score > 0:
        sentiment = "Positive"
    elif score < 0:
        sentiment = "Negative"
    else:
        sentiment = "Neutral"
    print(s,"\t\t\t",sentiment,"\t\t\t",score)
```

Vader Method

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyzer = SentimentIntensityAnalyzer()

for s in sentences:
    score = analyzer.polarity_scores(s)['compound']

    if score > 0:
        sentiment = "Positive"
    elif score < 0:
        sentiment = "Negative"
    else:
        sentiment = "Neutral"
    print(s,"\t\t\t",sentiment,"\t\t\t",score)
```

Machine Learning Model

```
import pandas as pd
df=pd.read_csv("Reviews.tsv",sep="\t")
df.head()
df.dropna(inplace=True)
df.shape
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df['text'],
                                    df['sentiment'],
                                    test_size=0.3,
                                    random_state=1)
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB()
model.fit(X_train_vec, y_train)
from sklearn.metrics import accuracy_score
y_pred = model.predict(X_test_vec)
print("Accuracy:", accuracy_score(y_test, y_pred))

new_texts = ["I love this phone", "This is a bad movie"]
new_vec = vectorizer.transform(new_texts)
print(model.predict(new_vec))
```

3. Language Models

3.1 Introduction

A Language Model (LM) is a fundamental component of Natural Language Processing (NLP) that learns the probability distribution of words in a language.

Definition:

A Language Model assigns a probability to a sequence of words — predicting the likelihood of the next word in a sentence.

Example:

If we start a sentence:

"The cat sat on the ___"
the model may predict "mat" as most probable, because it often follows that pattern in English.
Language Models are used in text prediction, machine translation, chatbots, and speech recognition.

3.2 Need for Language Models

- To make computers understand structure and meaning in human language.
- To predict or generate coherent text sequences.
- To improve spelling correction, speech recognition, and machine translation.
- To help chatbots and virtual assistants understand intent.

3.3 Types of Language Models

Type	Description	Example
Statistical Language Models	Based on probability and frequency of words. Includes Unigram, Bigram, Trigram models.	"The boy" (Bigram), "The boy plays" (Trigram)
Neural Language Models	Uses neural networks to capture word dependencies beyond fixed windows.	RNN, LSTM, GRU
Transformer-based Models	Uses attention mechanism to model long-range context efficiently.	BERT, GPT, T5

3.4 Statistical Language Models

Language models calculate the probability of word sequences.
For a sentence W = w_1, w_2, w_3, ..., w_n,

$$\boldsymbol{P(W) = P(w_1) \times P(w_2 \mid w_1) \times P(w_3 \mid w_1, w_2) \times \ldots \times P(w_n \mid w_1, w_2, \ldots, w_{n-1})}$$

But this is complex — so we make the Markov assumption (a word depends only on a few previous words).

(a) Unigram Model

Assumes each word is independent:

$$P(W) = P(w_1) \times P(w_2) \times P(w_3) \ldots$$

Example: "AI is amazing" → P(AI) × P(is) × P(amazing)

(b) Bigram Model

Considers the previous word:

$$P(W) = P(w_1) \times P(w_2 \mid w_1) \times P(w_3 \mid w_2) \ldots$$

Example: “New York City” – the bigram “New York” is much more probable than “York City.”

(c) Trigram Model

Considers two previous words:

$$P(w_3 \mid w_1, w_2)$$

Example: “The cat sat” vs “Cat sat on.”

3.5 Neural Language Models

Overcome the limitations of statistical models by using word embeddings (numerical representations of words) and neural networks.

Model	Description
RNN (Recurrent Neural Network)	Handles sequences by maintaining hidden states.
LSTM (Long Short-Term Memory)	Deals with long-term dependencies and context.
GRU (Gated Recurrent Unit)	Simplified version of LSTM, faster training.

Example: Predicting the next word in “I am feeling very ___” → likely “happy.”

3.6 Transformer-Based Language Models

Transformers use the self-attention mechanism, which allows the model to focus on important words in a sentence, regardless of their distance.

Model	Key Feature
BERT (Bidirectional Encoder Representations from Transformers)	Understands context from both left and right sides.
GPT (Generative Pre-trained Transformer)	Generates text by predicting the next word sequentially.
T5 (Text-to-Text Transfer Transformer)	Converts all NLP tasks into a text-to-text format.

3.7 Evaluation of Language Models

Common metrics to measure LM performance:

Metric	Description
Perplexity	Lower perplexity = better model. Measures how well a model predicts sample text.
Accuracy / BLEU	Used in translation and text generation evaluation.
Log-likelihood	Probability of a given sentence under the model.

3.8 Applications of Language Models

- Autocorrect and Text Prediction
  (Mobile keyboard suggestions)
- Speech Recognition
  (Predicts next word in spoken phrases)
- Machine Translation
  (Predicts best target-language sentence)
- Chatbots and Virtual Assistants
  (Maintains conversation flow)
- Document Summarization and Paraphrasing

3.9 Challenges

- Ambiguity in words or phrases.
- Handling low-frequency or unseen words.
- Large computational cost in neural models.
- Ethical concerns — bias, misinformation, or data privacy.

4. Chatbots

4.1 Introduction

A Chatbot is an AI-powered software designed to simulate human conversation through text or voice interactions.

They are one of the most practical applications of Natural Language Processing (NLP) and Language Models.

Definition:

A chatbot is an AI system that can interact with users in natural language and respond intelligently to their questions or commands.

Examples:

- Customer support bots on websites.

- Virtual assistants like Alexa, Siri, and Google Assistant.
- ChatGPT, a generative AI-based conversational system.

4.2 Purpose of Chatbots

- To automate customer interactions and reduce manual workload.
- To provide 24×7 support for businesses.
- To assist users in information retrieval (e.g., banking, bookings).
- To make human–computer communication natural and engaging.

4.3 Types of Chatbots

Type	Description	Example
Rule-Based Chatbots	Use predefined scripts or flowcharts. Respond based on keywords or patterns.	IVR systems, FAQ bots
Retrieval-Based Chatbots	Choose the best-matching predefined response using NLP similarity.	Customer service assistants
Generative Chatbots	Generate new responses dynamically using deep learning models.	ChatGPT, Google Bard

4.4 Components of a Chatbot System

1. User Interface (UI):
   The medium of interaction (text box, voice input, chat window).
2. Natural Language Understanding (NLU):
   Extracts user intent and entities from text.
   Example:
   - User: “Book a flight to Delhi tomorrow.”
   - Intent: Booking
   - Entities: Destination = Delhi, Date = Tomorrow
3. Dialogue Manager:
   Maintains the state of conversation and decides the next action.
4. Response Generation:
   Produces an appropriate reply — either retrieved or generated.
5. Database or API Layer:
   Fetches real-time information if required (e.g., weather, flight status).

4.5 Workflow of a Chatbot

1. Input: User sends a message.
2. Processing:
    - The chatbot interprets text using NLP.
    - Identifies intent and entities.
3. Response Generation:
    - Retrieves or generates a suitable answer.
4. Output: Sends the response to the user.
5. Learning (for AI chatbots):
    - Model improves over time with more data.

4.6 Technologies Used

- NLP libraries: spaCy, NLTK, Rasa NLU.
- Deep learning frameworks: TensorFlow, PyTorch.
- Pretrained models: BERT, GPT, T5.
- APIs: Dialogflow (Google), IBM Watson Assistant, Microsoft Bot Framework.

4.7 Example: Simple Travel Booking Chatbot

Conversation Example:

User: “Book a train ticket to Mumbai for tomorrow.”

Bot: “Sure! What time would you like to travel?”

User: “Morning.”

Bot: “Train ticket to Mumbai tomorrow morning has been booked!”

How it works:

- Intent: Ticket booking
- Entity: Location = Mumbai, Time = Tomorrow Morning
- Response generated based on learned pattern or template.

4.8 Advantages of Chatbots

Advantages

- Available 24×7.
- Fast and consistent replies.
- Cost-effective (reduce human workload).
- Personalization through user data.
- Scalable — handle thousands of users simultaneously.

4.9 Limitations of Chatbots

Limitations

- Limited understanding of complex emotions or sarcasm.
- May fail if input deviates from trained data.
- Difficult to handle long contextual conversations.
- Overdependence on language model quality.

4.10 Applications of Chatbots

- Customer Service: Answering queries on websites.
- Healthcare: Symptom checking, appointment booking.
- Education: Virtual tutors, FAQ bots.
- Banking & E-commerce: Account information, order tracking.
- Entertainment: Personalized recommendations and interactive games.

4.11 Experiential Learning

```
from transformers import BlenderbotTokenizer
from transformers import BlenderbotForConditionalGeneration

model_name = "facebook/blenderbot-400M-distill"
tokenizer = BlenderbotTokenizer.from_pretrained(model_name)
model = BlenderbotForConditionalGeneration.from_pretrained(model_name)
print("Chatbot is ready! Type 'quit' to stop.\n")
while True:
    user_input = input("You: ")
    if user_input.lower() == "quit":
        break

    inputs = tokenizer([user_input], return_tensors="pt")
    reply_ids = model.generate(**inputs)
    reply = tokenizer.batch_decode(reply_ids,
                                   skip_special_tokens=True)[0]
    print("Chatbot:", reply)
```

5.Computer Vision & Image Classification

5.1 Introduction

Computer Vision (CV) is a field of Artificial Intelligence (AI) that enables computers to see, interpret, and understand images or videos, just like humans do.

Definition:

Computer Vision is the technology that allows machines to extract, process, and analyze visual data (images or videos) to make intelligent decisions.

Goal:

To make computers "see" and understand visual inputs — identifying objects, people, scenes, or actions automatically.

Example:

- Face detection on smartphones.
- Google Photos auto-tagging.
- Self-driving cars identifying road signs.

5.2 Human Vision vs Computer Vision

Aspect	Human Vision	Computer Vision
Input	Eyes	Cameras/Sensors
Processing	Brain interprets objects	Algorithms process pixels
Learning	Experience	Machine Learning/Deep Learning
Decision	Based on visual understanding	Based on trained models

5.3 Applications of Computer Vision

- Facial Recognition: Security systems, attendance apps.
- Healthcare: Detecting diseases from X-rays or MRI scans.
- Agriculture: Crop monitoring and yield prediction.
- Retail: Product recognition and inventory automation.
- Autonomous Vehicles: Object detection and lane tracking.
- Robotics: Navigation and environment mapping.

5.4 Steps in Computer Vision System

1. Image Acquisition
    - Capturing an image from a camera or dataset.
    - Example: Using CCTV or webcam for live input.
2. Image Preprocessing
    - Improving quality for better analysis:
        - Resizing, Noise Reduction, Contrast Adjustment.
    - Converts images into consistent formats.
3. Feature Extraction

- Identifying key characteristics (edges, colors, textures).
- Traditional methods: SIFT, HOG, SURF.
- Deep Learning methods: CNN automatically extracts features.

4. Object Detection / Classification
   - Model identifies or labels the image.
   - Example: “This is a dog” or “This is a cat.”
5. Decision Making
   - Model output is used for real-world tasks like navigation, medical diagnosis, or alerts.

5.5 Image Representation

- Images are made of pixels arranged in a 2D grid.
- Each pixel stores color intensity values (RGB: Red, Green, Blue).
- Computer Vision converts these pixels into numerical matrices that algorithms can process.

5.6 Image Classification

Definition:

Image classification is the process of assigning a label to an image based on its visual content.

It involves training a model to recognize patterns within images.

Example:

Training data contains:

- 1000 images of cats
- 1000 images of dogs

When shown a new image, the model predicts whether it’s a cat or dog.

5.7 Machine Learning vs Deep Learning Approaches

Approach	Description	Techniques
Traditional ML	Uses handcrafted features like shape, texture	SVM, KNN, Random Forest
Deep Learning	Automatically extracts hierarchical features	CNN (Convolutional Neural Network)

5.8 Convolutional Neural Networks (CNNs)

CNNs are the foundation of modern computer vision.

They process images using convolutional layers to detect patterns like edges, corners, shapes, and objects.

Basic Structure:

1. Input Layer: Receives image pixels.
2. Convolution Layer: Applies filters to extract features.
3. Pooling Layer: Reduces image size (keeps essential info).
4. Fully Connected Layer: Combines extracted features to predict output.
5. Output Layer: Classifies the image into categories (e.g., Dog, Cat).

5.9 Example: Fruit Classification

Objective: Identify whether the image is of an Apple, Banana, or Orange.

Steps:

1. Load the dataset of labeled fruit images.
2. Preprocess: resize images to same dimensions.
3. Train CNN model (with convolution + pooling layers).
4. Test model on new fruit image.
5. Output: Predicted label (e.g., “Orange”).

5.10 Advantages of Computer Vision

- Automates visual inspection tasks.
- Improves accuracy and speed of recognition.
- Enables real-time object tracking.
- Reduces human intervention in repetitive visual work.

5.11 Limitations of Computer Vision

- Requires large labeled datasets for training.
- Sensitive to lighting, angle, and image noise.
- High computational cost for complex deep learning models.
- May fail with unseen or ambiguous images.

5.12 Experiential Learning

Image Classification Using CNN (Deep Learning Method)

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_gen = ImageDataGenerator(rescale=1./255)
test_gen = ImageDataGenerator(rescale=1./255)
```

```
train_data = train_gen.flow_from_directory('train', target_size=(64, 64), batch_size=32, class_mode='binary')
test_data = test_gen.flow_from_directory('test', target_size=(64, 64), batch_size=32, class_mode='binary')
print("Class indices:", train_data.class_indices)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),  # Convolution
    MaxPooling2D(2, 2),                                  # Pooling
    Flatten(),                                  # Flattening
    Dense(128, activation='relu'),                       # Hidden layer
    Dense(1, activation='sigmoid')                       # Output layer (binary)
])
model.compile(loss='binary_crossentropy', metrics=['accuracy'])

model.fit(train_data, epochs=10, validation_data=test_data)
loss, acc = model.evaluate(test_data)
print("Test Accuracy:",acc)
pred = model.predict(test_data)
print(pred)
predicted = (pred > 0.5).astype(int)
predicted
for i in range(5):
    if predicted[i] == 0:
        print("Cat")
    else:
        print("Dog")
```

Image Classification using Pretrained Model

```
#!pip install ultralytics
from ultralytics import YOLO
model = YOLO("yolov8n.pt")
results = model.predict("image.jpeg", show=True)
```

6. Game Playing

6.1 Introduction

Game Playing is an important research area in Artificial Intelligence. Games provide a controlled environment to test AI strategies, reasoning, and decision-making.

- Games do not require huge domain knowledge.
- The key elements are:
  - Rules
  - Legal moves
  - Win/loss conditions
- AI systems aim to play optimally and predict future situations.

1.2 Types of Games

A. Deterministic vs Stochastic

Type	Meaning	Example
Deterministic	No randomness; outcome depends only on players' actions	Chess
Stochastic	Outcome involves chance or randomness	Backgammon

B. Zero-Sum vs Non-Zero-Sum

Type	Meaning	Example
Zero-Sum	One player's gain = another player's loss	Chess
Non-Zero-Sum	Players may cooperate or both gain/lose	Prisoner's Dilemma

C. Perfect vs Imperfect Information

Type	Meaning	Example
Perfect Information	All players see the entire game state	Chess
Imperfect Information	Some information is hidden	Poker

6.3 Why traditional search (BFS/DFS) is not suitable

- Every position in a game may have multiple possible moves.
- BFS/DFS takes too long because the game tree grows exponentially.

Therefore, AI needs smarter procedures:

1. Generate procedure → generate only *good* moves
2. Test procedure → evaluate best move first

6.4 Game Playing Techniques in AI

1. Search Algorithms
   - Minimax algorithm
   - Alpha-beta pruning (reduces unnecessary branches)
2. Machine Learning Approaches
   - Deep Reinforcement Learning
     - Agents learn by trial and error
     - Example: DeepMind's AlphaGo

6.5 Minimax Algorithm

- Applied in two-player adversarial games (Chess, Checkers, Tic-Tac-Toe).
- Two roles:
  - Maximizer → Tries to win (maximize score)
  - Minimizer → Tries to make the maximizer lose (minimize score)

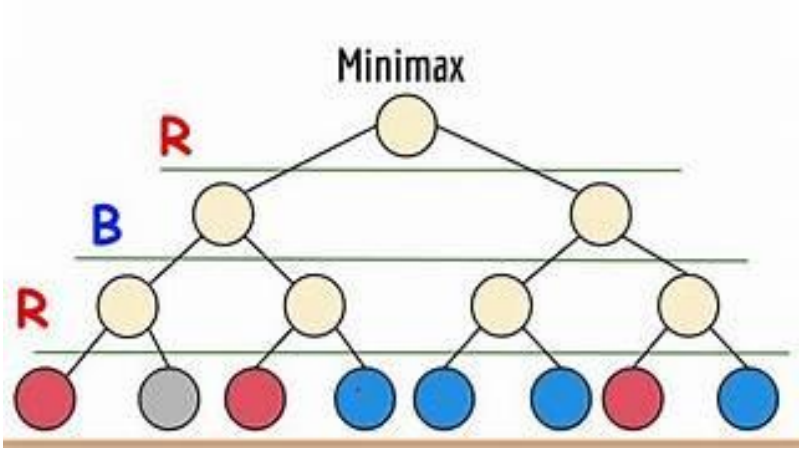
How Minimax Works (as per slides)

1. Build the complete game tree to terminal nodes:
   - Terminal states include win/loss/draw.
   - Assign utility values:

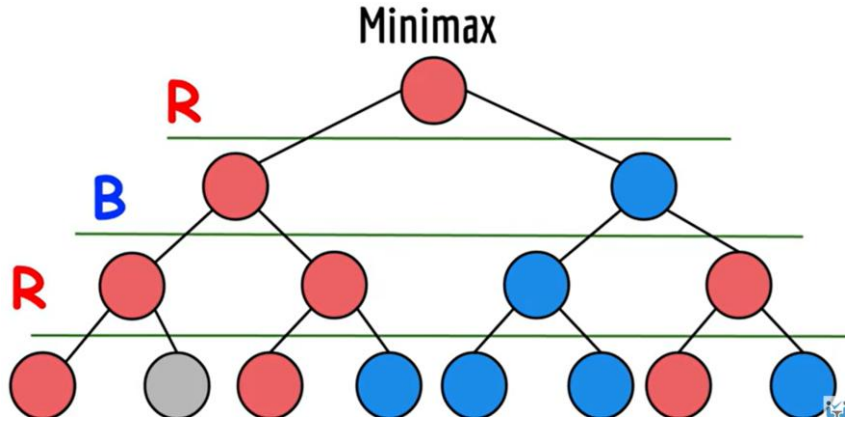
     +1 → win for maximizer

     –1 → win for minimizer

     0 → draw



2. Backpropagate (backup) values:
   - At Max node → choose *maximum* value among children
   - At Min node → choose *minimum* value among children



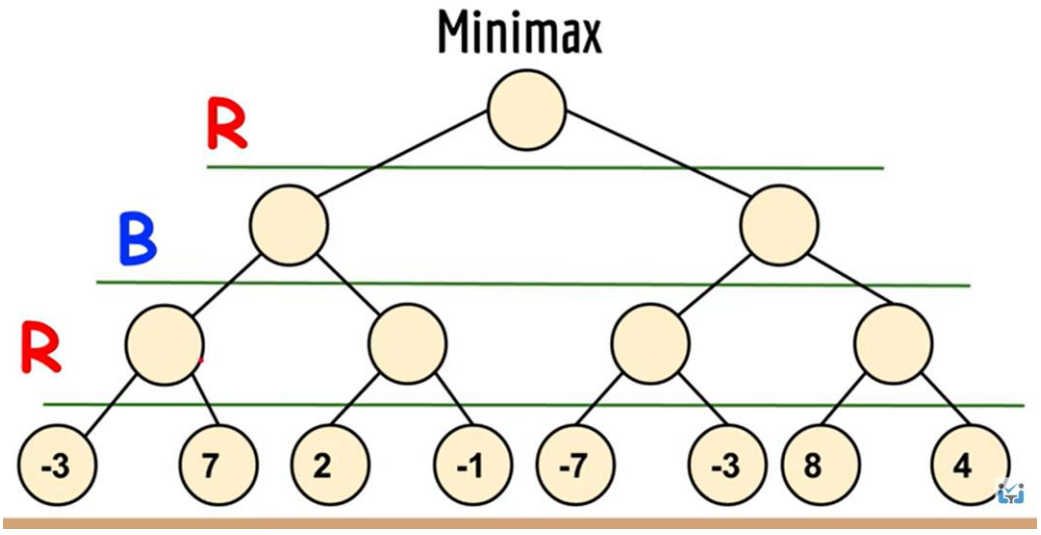
6.6 Heuristic Evaluation (Non-terminal States)

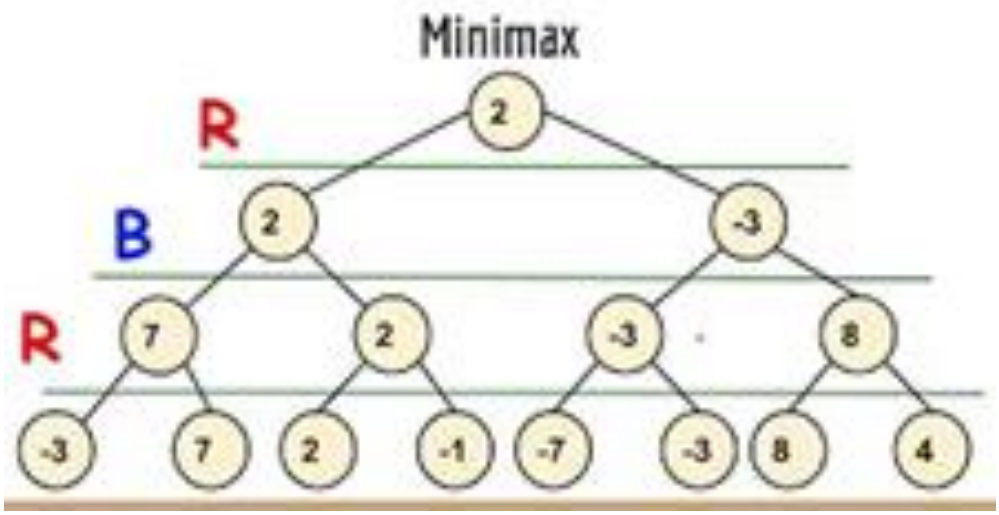
When full game tree is too deep:

- Assign heuristic values to positions:

  Positive → good for maximizer (R)

  Negative → good for minimizer (B)
- Min chooses smallest
- Max chooses largest





6.7 Experiential Learning

Tic-Tac-Toe

Initialize empty board

```
import math
board = [' ' for _ in range(9)]
```

Function to print board

```
def print_board(board):
    print()
    for i in range(3):
        print(board[3*i], '|', board[3*i+1], '|', board[3*i+2])
        if i < 2:
            print('--+---+--')
    print()
```

Check for a winner

```
def check_winner(board):
    win_conditions = [
        (0,1,2), (3,4,5), (6,7,8),  # rows
        (0,3,6), (1,4,7), (2,5,8),  # columns
        (0,4,8), (2,4,6)            # diagonals
    ]
```

```
    for a,b,c in win_conditions:
        if board[a] == board[b] == board[c] and board[a] != ' ':
            return board[a]
    return None
```

Check if board is full

```
def is_full(board):
    return ' ' not in board
```

MiniMax algorithm

```
def minimax(board, is_maximizing):
    winner = check_winner(board)
    if winner == 'O':  # Computer
        return 1
    elif winner == 'X':  # Human
        return -1
    elif is_full(board):
        return 0

    if is_maximizing:
        best_score = -math.inf
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'O'
                score = minimax(board, False)
                board[i] = ' '
                best_score = max(score, best_score)
        return best_score
    else:
```

```
        best_score = math.inf
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'X'
                score = minimax(board, True)
                board[i] = ' '
                best_score = min(score, best_score)
        return best_score
```

Function to get computer move

```
def best_move(board):
    best_score = -math.inf
    move = None
    for i in range(9):
        if board[i] == ' ':
            board[i] = 'O'
            score = minimax(board, False)
            board[i] = ' '
            if score > best_score:
                best_score = score
                move = i
    return move
```

Play Game

```
def play_game():
    print("Welcome to Tic Tac Toe! You are X, Computer is O.")
    print_board(board)

    while True:
        # Human move
        human_move = int(input("Enter your move (1-9): ")) - 1
```

```
        if board[human_move] != ' ':
            print("Cell already taken. Try again.")
            continue
        board[human_move] = 'X'

        print_board(board)
        if check_winner(board) == 'X':
            print("You win!")
            break
        if is_full(board):
            print("It's a draw!")
            break

        # Computer move
        print("Computer is thinking...")
        move = best_move(board)
        board[move] = 'O'
        print_board(board)

        if check_winner(board) == 'O':
            print("Computer wins!")
            break
        if is_full(board):
            print("It's a draw!")
            break
play_game()
```

7. Real-World Use-Cases of AI

7.1 Introduction

AI is widely used in:

- Healthcare
- E-commerce
- Education
- Business & Finance
- Manufacturing
- Transportation
- Media & Entertainment
- Agriculture
- Security & Defense

Each domain uses AI for automation, decision-making, prediction, and improving user experience.

7.2 AI in Healthcare

Applications:

- Disease prediction & diagnosis
- Medical image analysis (X-ray, MRI)
- Drug discovery
- Virtual assistants & chatbots
- Personalized treatment recommendations

Example from slide:

IBM Watson Health helps doctors in diagnosis and treatment planning.

7.3 AI in E-Commerce

Applications:

- Product recommendation systems (Amazon, Flipkart)
- Customer behavior analysis
- Dynamic pricing
- Chatbots for product queries

Example:

Amazon's recommendation engine suggests best products.

7.4 AI in Education

Applications:

- Personalized learning
- Automated grading
- Virtual AI tutors
- Predicting student performance

Example:

Duolingo uses AI for personalized language learning paths.

7.5 AI in Business & Finance

Applications:

- Fraud detection
- Credit scoring
- Customer service bots
- Predictive analytics
- Algorithmic trading

Example:

American Express uses AI for fraud detection.

7.6 AI in Manufacturing

Applications:

- Predictive maintenance
- Quality control
- Supply chain optimization
- Robotics & automation

Example:

Siemens uses AI for factory predictive maintenance.

7.7 AI in Transportation

Applications:

- Self-driving cars
- Traffic prediction
- Drones and delivery robots
- Smart traffic systems

Example:

Tesla Autopilot uses deep learning for navigation.

7.8 AI in Media & Entertainment

Applications:

- Movie/Video recommendation (Netflix, YouTube)
- Automatic subtitles
- Deepfake detection
- AI-driven game characters

Example:

Netflix recommendation engine.

7.9 AI in Agriculture

Applications:

- Crop health monitoring
- Weather prediction
- Automated irrigation
- Pest detection
- Smart tractors

Example:

John Deere uses AI for precision agriculture.

7.10 AI in Security & Défense

Applications:

- Video surveillance
- Cybersecurity systems
- Facial recognition
- Autonomous drones

Example:

AI-based facial recognition at borders.