3️⃣

NLP - Module 3 Gaurav Panjabi

Parsing In NLP <V.IMP complete topic>

- Parsing is the process of examining the grammatical structure and relationships inside a given sentence or text in natural language processing (NLP).
- It involves analyzing the text to determine the roles of specific words, such as nouns, verbs, and adjectives, as well as their interrelationships.
- This analysis produces a structured representation of the text, allowing NLP computers to understand how words in a phrase connect to one another.
- Parsers expose the structure of a sentence by constructing parse trees or dependency trees that illustrate the hierarchical and syntactic relationships between words.
- This stage is crucial for a variety of language understanding tasks, which allow machines to extract meaning, provide coherent answers, and execute tasks such as machine translation, sentiment analysis, and information extraction.

Parsing Techniques in NLP

1. Top-Down Parsing

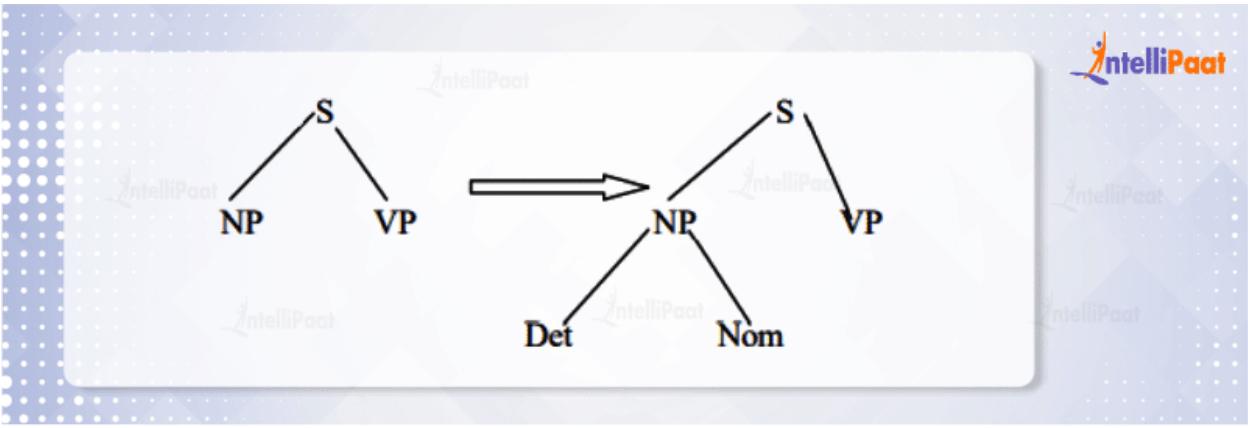
- A parse tree is a tree that defines how the grammar was utilized to construct the sentence. Using the top-down approach, the parser attempts to create a parse tree from the root node S down to the leaves.
- The procedure begins with the assumption that the input can be derived from the selected start symbol S.
- The next step is to find the tops of all the trees that can begin with S by looking at the grammatical rules with S on the left-hand side, which generates all the possible trees.

- Top-down parsing is a search with a specific objective in mind.
- It attempts to replicate the initial creation process by rederiving the sentence from the start symbol, and the production tree is recreated from the top down.
- Top-down, left-to-right, and backtracking are prominent search strategies that are used in this method.
- The search begins with the root node labeled S, i.e., the starting symbol, expands the internal nodes using the next productions with the left-hand side equal to the internal node, and continues until leaves are part of speech (terminals).
- If the leaf nodes, or parts of speech, do not match the input string, we must go back to the most recent node processed and apply it to another production.

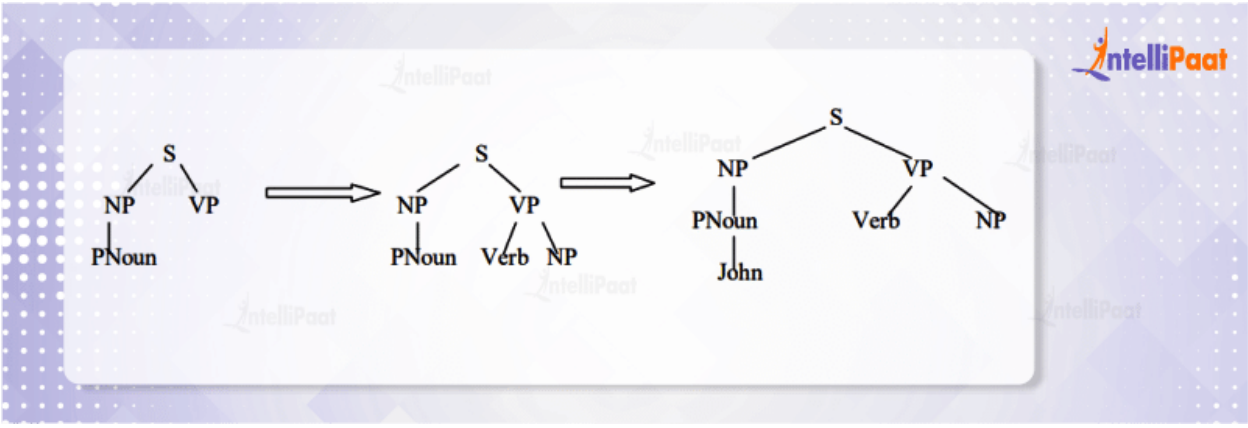
Let's consider the grammar rules:

Sentence = S = Noun Phrase (NP) + Verb Phrase (VP) + Preposition Phrase (PP)

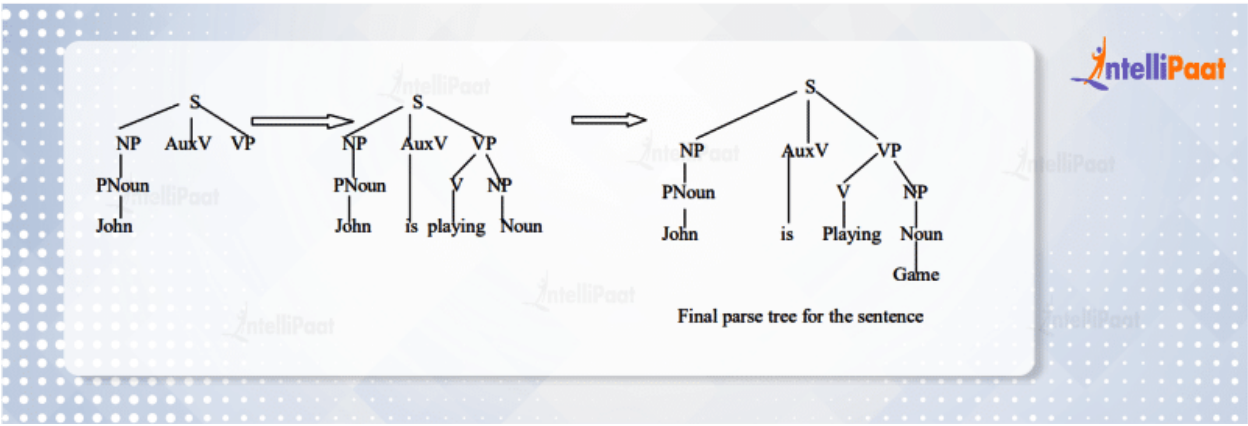
Take the sentence: "John is playing a game", and apply Top-down parsing



If part of the speech does not match the input string, backtrack to the node NP.



Part of the speech verb does not match the input string, backtrack to the node S, since PNoun is matched.



The top-down technique has the advantage of never wasting time investigating trees that cannot result in S, which indicates it never examines subtrees that cannot find a place in some rooted tree.

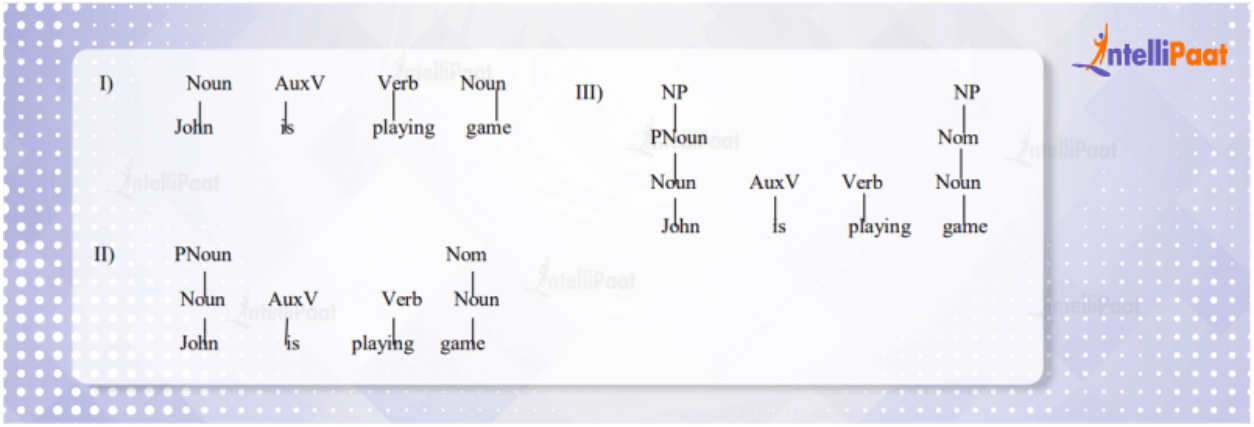
2. Bottom-Up Parsing

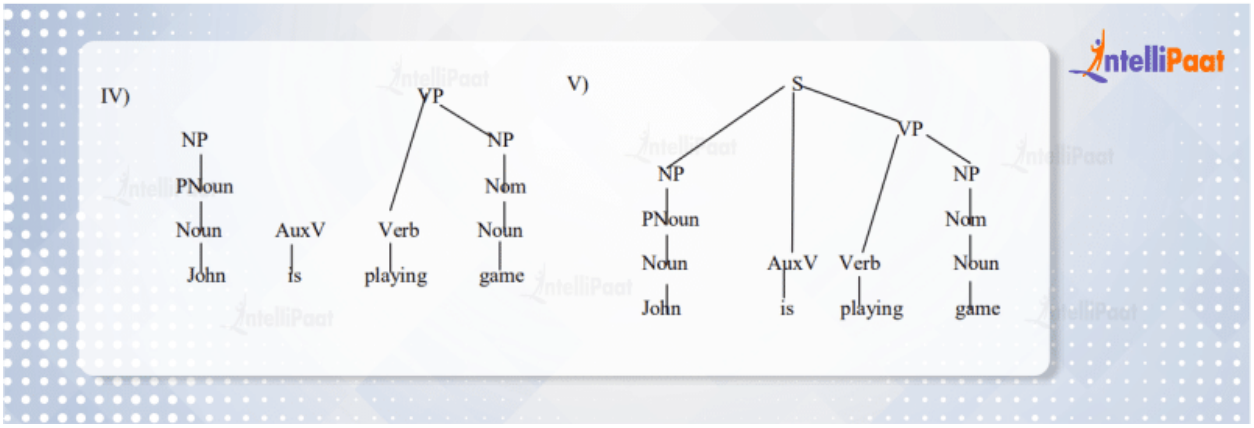
- Bottom-up parsing begins with the words of input and attempts to create trees from the words up, again by applying grammar rules one at a time.
- The parse is successful if it builds a tree rooted in the start symbol S that includes all of the input. Bottom-up parsing is a type of data-driven search. It attempts to reverse the manufacturing process and return the phrase to the start symbol S.
- It reverses the production to reduce the string of tokens to the beginning Symbol, and the string is recognized by generating the rightmost derivation in reverse.

- The goal of reaching the starting symbol S is accomplished through a series of reductions; when the right-hand side of some rule matches the substring of the input string, the substring is replaced with the left-hand side of the matched production, and the process is repeated until the starting symbol is reached.
- Bottom-up parsing can be thought of as a reduction process. Bottom-up parsing is the construction of a parse tree in postorder.

Considering the grammatical rules stated above and the input sentence “John is playing a game”,

The bottom-up parsing operates as follows:





Comparison of top-down and bottom-up parsing in NLP:

Feature	Top-Down Parsing	Bottom-Up Parsing
Direction	Starts with the starting symbol (top)	Starts with the input string (bottom)
Derivation	Uses leftmost derivation	Uses rightmost derivation

Approach	Recursive/Constructive	Shift-reduce
Implementation	Simple	Complex
Backtracking	May require backtracking	Generally avoids backtracking
Impact	It is less powerful then Bottom Up Parsing	It is more powerful then top down parsing
Efficiency	Can be inefficient for ambiguous grammars	More efficient for larger grammars, less backtracking
Examples	Recursive descent, LL parsers	Shift-reduce parsers (LR, SLR, LALR)
Common use	Simpler grammars, smaller inputs	Larger grammars, ambiguous grammars

POS tagging

1. Part-of-speech tagging is the process of assigning a part-of-speech or other lexical class marker to each word in a corpus.
2. Tags are also usually applied to punctuation markers; thus tagging for natural language is the same process as tokenization for computer languages, although tags for natural languages are much more ambiguous.
3. The input to a tagging algorithm is a string of words and a specified tagset.
4. The output is a single best tag for each word.
5. For example, here are some sample sentences from the Airline Travel Information Systems (ATIS) corpus of dialogues about air-travel reservations.
6. For each we have shown a potential tagged output using the Penn Treebank tagset

VB DT NN
Book that flight .
VBZ DT NN VB NN ?
Does that flight serve dinner ?

7. Even in these simple examples, automatically assigning a tag to each word is not trivial.
8. For example, book is ambiguous.
9. That is, it has more than one possible usage and part of speech.

10. It can be a verb (as in book that flight or to book the suspect) or a noun (as in hand me that book, or a book of matches):
11. Similarly, that can be a determiner (as in Does that flight serve dinner), or a complementizer (as in I thought that your flight was earlier).
12. The problem of POS-tagging is to resolve these ambiguities, choosing the proper tag for the context.
13. Most of the POS tagging falls under Rule Base POS tagging, Stochastic POS tagging and Transformation based tagging.

Approaches/Methods to perform Part of Speech (POS) tagging

1. **ule-Based POS Tagging:**
    - **Lexical Rules:** Assign POS tags based on the individual words. For example, words ending in "-ly" are often adverbs.
    - **Contextual Rules:** Consider the context of a word in a sentence to determine its POS tag. For instance, if a word follows "the," it is likely to be a noun.
    - **Morphological Rules:** Analyze the morphology of words, such as suffixes or prefixes, to infer their POS tags.
2. **Probabilistic Models:**
    - **N-gram Models:** Use statistical models like bigram or trigram models to predict the POS tag of a word based on the previous one or two words.
    - **Hidden Markov Models (HMM):** Model the sequence of POS tags as a hidden state sequence, and the observed words as emissions. The Viterbi algorithm is often used for decoding.
3. **Machine Learning-Based POS Tagging:**
    - **Supervised Learning:** Train a machine learning model (e.g., decision trees, support vector machines, or neural networks) on a labeled dataset where each word is tagged with its POS.
    - **CRF (Conditional Random Fields):** A type of probabilistic model that considers the dependencies between neighboring words and their POS tags.

- **Maximum Entropy Models:** Use models that maximize the entropy given the observed data to predict POS tags.

4. **Combination or Hybrid Approaches:**
    - Combine the strengths of rule-based and machine learning methods to create hybrid models. For example, use rule-based methods as a pre-processing step before applying machine learning models.
5. **Unsupervised Learning:**
    - Use unsupervised learning techniques, such as clustering algorithms or distributional similarity, to group words with similar syntactic behavior without relying on labeled training data.

Why is POS tagging hard? Discuss possible challenges while performing POS tagging.

1. **Word ambiguity:** Many words have multiple parts of speech depending on the context. For example, "play" can be a noun ("the play was a hit"), a verb ("the children played in the park"), or an adjective ("a playful kitten"). Determining the correct tag requires understanding the surrounding words and sentence structure.
2. **Out-of-vocabulary (OOV) words:** POS taggers are trained on large datasets of text, but new, rare, or domain-specific words not in the training data can trip them up. Assigning the most likely tag for these OOV words is often inaccurate.
3. **Morphological complexity:** Languages with rich inflectional morphology, like Latin or Polish, present extra challenges. Words can change their form significantly depending on grammatical function, making it harder to identify the base form and its corresponding part of speech.
4. **Contextual dependencies:** Accurate POS tagging often relies on understanding the broader context of a sentence, including semantic relationships between words and grammatical rules. This introduces challenges for taggers that rely solely on local information like word prefixes or suffixes.
5. **Data sparsity:** Training effective POS taggers requires large amounts of high-quality annotated data. However, for some languages or specialized domains, such data may be scarce or limited, impacting the model's accuracy and generalization capabilities.

6. **Tagset complexities:** Different tagsets used in POS tagging vary in granularity and complexity. Some tagsets distinguish fine-grained verb tenses or noun types, while others use coarser categories. Choosing the appropriate tagset depends on the specific application and target language, but can also impact the difficulty of the task.
7. **Model limitations:** Traditional rule-based taggers may struggle with ambiguity and lack flexibility. While deep learning approaches have shown promising results, they can be computationally expensive and require careful hyperparameter tuning to avoid overfitting or bias.

Explain rule-based POS tagging with example.

- Rule-based POS tagging involves using a set of linguistic rules to assign parts-of-speech (POS) tags to words in a given text.
- These rules are typically based on patterns, heuristics, or linguistic observations about how certain words or structures in a language tend to behave grammatically.
- Rule-based approaches are transparent, interpretable, and can be effective in certain contexts, but they may not capture the complexity and ambiguity found in natural language as well as more sophisticated machine learning models.

Here is a simple example of a rule-based POS tagging approach for English:

Example Rule: Adjective-Noun Combination

- **Rule:** If a word is preceded by an adjective, it is likely to be a noun.
- **Example:**
  - Sentence: "The **big** house is beautiful."
  - Rule-based POS Tagging:
    - "The" - Determiner
    - "big" - Adjective
    - "house" - Noun
    - "is" - Verb
    - "beautiful" - Adjective

In this example, the rule suggests that if a word follows an adjective, it is more likely to be a noun. As per the rule, "big" is tagged as an adjective, and "house" is tagged as a noun.

Example Rule: Verb Identification

- **Rule:** Words ending in "-ing" are likely to be verbs.
- **Example:**
  - Sentence: "She is **running** in the park."
  - Rule-based POS Tagging:
    - "She" - Pronoun
    - "is" - Verb
    - "running" - Verb
    - "in" - Preposition
    - "the" - Determiner
    - "park" - Noun

In this example, the rule suggests that words ending in "-ing" are likely to be verbs. The word "running" is tagged as a verb based on this rule.

Explain Maximum Entropy Model for POS Tagging.

- **Maximum Entropy Model (MaxEnt) for POS Tagging** is a statistical approach that uses a rich set of features and contextual information to assign the most likely POS tag to each word in a sentence.
- It's a supervised machine learning method that learns from annotated text data.
- The goal of the MaxEnt model in POS tagging is to find the probability distribution that maximizes the entropy given the observed training data.
- The model relies on a set of features that capture relevant information about the context of a word for POS tagging. These features can include the identity of the word itself, the surrounding words, their POS tags, and other contextual information.
- The model is trained on labeled data, where each word in a sentence is associated with its correct POS tag. During training, the model learns the weights associated

with each feature, adjusting them to maximize the likelihood of the observed data.

T**raining process:**

1. **Feature extraction:** The model extracts relevant features from a large corpus of annotated text.
2. **Parameter estimation:** The model uses an optimization algorithm, typically iterative scaling, to estimate the weights associated with each feature. These weights determine the contribution of each feature to the final probability distribution.

Tagging process:

1. **Feature extraction:** The model extracts features for each word in the new, unseen sentence.
2. **Probability calculation:** The model uses the trained weights to calculate the probability of each possible POS tag for each word.
3. **Tag assignment:** The model assigns the tag with the highest probability to each word, forming the final POS-tagged sentence.

Advantages of MaxEnt models for POS tagging:

- **Flexibility:** Can incorporate a wide range of features, making them adaptable to different languages and domains.
- **Accuracy:** Often achieve high accuracy, outperforming rule-based approaches in many cases.
- **Robustness:** Handle ambiguity and noise in the data relatively well.
- **Interpretability:** The weights assigned to features can provide insights into the model's decision-making process.

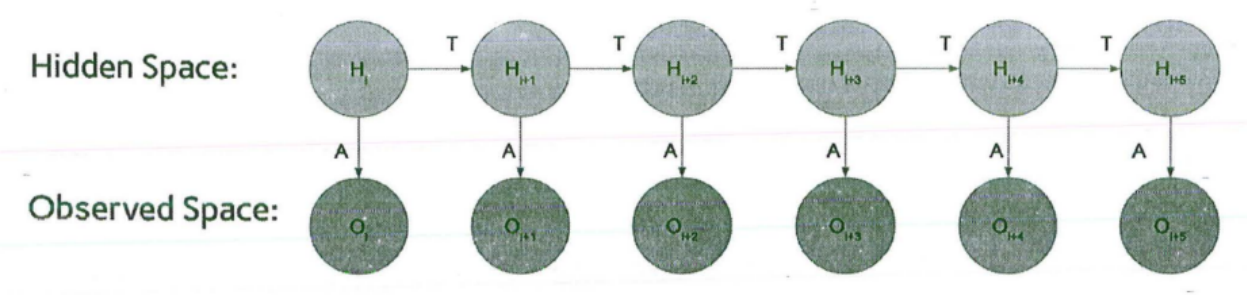
Hidden Markov Model

- Hidden Markov models (HMMs) are sequence models.
- **Key Concepts:**
    - **Hidden States:** These represent the underlying, unobservable structure of the data, such as the grammatical categories of words in a sentence.
    - **Observations:** These are the actual words or symbols we see in the text.

    - **Transition Probabilities:** These capture the likelihood of moving from one hidden state to another.
    - **Emission Probabilities:** These model the probability of generating a particular observation from a given hidden state.
- HMMs are "a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobservable (i.e. hidden) states".
- They are designed to model the joint distribution P(H , 0) , where H is the hidden state and O is the observed state.
- For example, in the context of POS tagging, the objective would be to build an HMM to model P(word I tag) and compute the label probabilities given observations using Bayes' Rule:

$$P(H|O) = \frac{P(O|H)P(H)}{P(O)}$$

- HMM graphs consist of a Hidden Space and Observed Space, where the hidden space consists of the labels and the observed space is the input.
- These spaces are connected via transition matrices {T, A} to represent the probability of transitioning from one state to another following their connections.
- Each connection represents a distribution over possible options; given our tags, this results in a large search space of the probability of all words given the tag.



- The main idea behind HMMs is that of making observations and traveling along connections based on a probability distribution.

- In the context of sequence tagging, there exists a changing observed state (the tag) which changes as our hidden state (tokens in the source text) also changes.

Applications of HMMs in NLP:

- **POS Tagging:** Unmasking words' secret identities (noun, verb, adjective).
- **NER:** Spotlighting the VIPs of text (names, places, brands).
- **Speech Recognition**
- **Machine Translation:** Bridging language barriers, one sentence at a time (hello, world!).
- **Text Chunking:** Carving sentences into meaningful phrases (group therapy for words).
- **Text Generation**
- **Sentiment Analysis:** Reading between the lines to uncover emotions (happy, sad, or meh?)

Advantages of HMMs in NLP:

- **Handle sequential data effectively:** Well-suited for modeling sequences of words or symbols, which is common in NLP tasks.
- **Robust to noise and ambiguity:** Can handle uncertainty and variations in language data.
- **Relatively efficient to train and decode:** Algorithms like the Forward-Backward algorithm and Viterbi algorithm allow efficient training and inference.

Limitations of HMMs:

- **Limited context:** Traditional HMMs only consider immediate neighboring states, potentially missing long-range dependencies in text.
- **Feature engineering:** Require careful feature engineering to capture relevant information for the task.
- **Expressivity:** Might not capture complex linguistic structures or semantic relationships.

What are the limitations of Hidden Markov Model

- **Limited Context:** Traditional HMMs only consider the immediate neighboring states when predicting the next state. This means they can struggle to capture long-range dependencies and complex relationships within the data sequence. This can be particularly problematic in tasks like natural language processing where distant words can influence the interpretation of current ones.
- **Limited Expressiveness:** HMMs are inherently limited in their ability to represent complex patterns and structures. They assume states and observations are independent, neglecting potential interactions between them. This can limit their effectiveness in tasks requiring a richer understanding of the data's underlying relationships.
- **Feature Engineering:** HMMs rely heavily on the features provided for training and analysis. Poorly chosen or insufficient features can significantly impact the model's ability to learn meaningful patterns and make accurate predictions. Choosing and engineering effective features can be a complex and critical step in using HMMs.
- **Stationarity Assumption:** Traditional HMMs assume the underlying transition and emission probabilities remain constant over time. This assumption may not hold true in real-world scenarios where dynamics and patterns can evolve over time. Models like Dynamic HMMs address this limitation, but they introduce additional complexity.
- **Data Requirements:** Training HMMs effectively often requires large amounts of labeled data. In domains with limited or scarce data, HMMs may perform poorly compared to other models that can learn from smaller datasets.
- **Computational Complexity:** While efficient algorithms exist for training and decoding HMMs, the complexity can increase significantly with larger state spaces and feature sets. This can make them less suitable for applications with massive datasets or highly complex tasks.
- **Comparison to other models:** In recent years, deep learning models like RNNs and Transformers have demonstrably surpassed HMMs in terms of accuracy and flexibility for many NLP tasks. While HMMs offer advantages in interpretability and efficiency, their limitations often necessitate exploring alternative models for optimal performance.

How HMM is used for POS tagging? Explain in detail. <IMP>

- Hidden Markov Models (HMMs) have been traditionally used for Part-of-Speech (POS) tagging in natural language processing. The basic idea is to model a sequence of words (observations) with corresponding hidden states representing the POS tags. Here's a detailed explanation of how HMMs are used for POS tagging:

Components of HMM for POS Tagging:

1. **Hidden States (POS Tags):**
    - Each hidden state corresponds to a POS tag. For example, in English, the set of hidden states might include tags like Noun (N), Verb (V), Adjective (ADJ), etc.
2. **Observations (Words):**
    - The observations are the words in a given sentence. Each word is associated with a specific POS tag in the training data.
3. **Transition Probabilities (A):**
    - The transition probabilities represent the likelihood of transitioning from one POS tag to another. These probabilities are estimated from the training data.
4. **Emission Probabilities (B):**
    - The emission probabilities represent the likelihood of observing a particular word given a POS tag. These probabilities are also estimated from the training data.
5. **Initial State Distribution (π):**
    - The initial state distribution represents the probability distribution of starting with a particular POS tag at the beginning of a sentence.

POS Tagging Process with HMM:

1. **Training:**
    - Given a labeled dataset of sentences with corresponding POS tags, the HMM is trained to estimate its parameters (transition probabilities, emission

probabilities, initial state distribution).

2. **Initialization:**
    - Initialize the HMM with the learned parameters.
3. **Inference (Viterbi Algorithm):**
    - Given an unseen sentence during testing, the Viterbi algorithm is employed to find the most likely sequence of hidden states (POS tags) for the observed sequence of words. The algorithm efficiently computes the most probable sequence by considering both the transition and emission probabilities.
    - The Viterbi algorithm maintains a dynamic programming table to efficiently compute the most likely sequence. It recursively fills in the table based on the probabilities of transitioning from one POS tag to another and the probabilities of observing a particular word given a POS tag.
    - At each step, the algorithm selects the most probable path to each state based on the cumulative probabilities.
4. **Assigning POS Tags:**
    - The Viterbi algorithm assigns a POS tag to each word in the sentence, producing a sequence of predicted POS tags.

Example:

Consider the sentence: "The cat is sitting on the mat."

- **States (POS Tags):** Noun (N), Verb (V), Article (ART), etc.
- **Observations (Words):** "The," "cat," "is," "sitting," "on," "the," "mat."

In this example, the Viterbi algorithm would determine the most likely sequence of POS tags for each word in the sentence based on the trained HMM.

Explain CFG with suitable example / Explain use of CFG in NLP with suitable example

In the context of Natural Language Processing (NLP), CFG stands for Context-Free Grammar. A Context-Free Grammar is a formalism used to describe the syntax or structure of a language. It consists of a set of production rules that define how various

components of a language can be combined to form valid sentences. Each rule in a CFG consists of a non-terminal symbol (representing a syntactic category) and a sequence of terminal and/or non-terminal symbols.

Let's consider a simple example of a CFG for a basic sentence structure in the English language. This CFG will describe how sentences can be formed with a subject, a verb, and an object:

1. Non-terminal symbols:
    - S (sentence)
    - NP (noun phrase)
    - VP (verb phrase)
    - Det (determiner)
    - N (noun)
    - V (verb)
2. Terminal symbols:
    - "the", "a" (determiners)
    - "cat", "dog" (nouns)
    - "chases", "catches" (verbs)
3. Production rules:
    - S → NP VP
    - NP → Det N
    - VP → V NP
    - Det → "the" | "a"
    - N → "cat" | "dog"
    - V → "chases" | "catches"

These rules specify how sentences, noun phrases, and verb phrases can be constructed. For example, the rule "S → NP VP" indicates that a sentence (S) can be formed by combining a noun phrase (NP) and a verb phrase (VP). The rules for NP, VP, Det, N, and V further break down the components of these phrases.

Now, let's use this CFG to generate a valid sentence:

1. Start with the sentence symbol S.
2. Use the rule "S → NP VP" to expand S into NP VP.
3. Use the rule "NP → Det N" to expand NP into "the cat".
4. Use the rule "VP → V NP" to expand VP into "chases NP".
5. Use the rule "Det → "the"" to expand Det into "the".
6. Use the rule "N → "cat"" to expand N into "cat".
7. Use the rule "V → "chases"" to expand V into "chases".

Putting it all together, we get the sentence "the cat chases the cat."

Discuss following potential problems in CFG such as 1) Agreement 2) Sub categorization 3) Movement.

1. **Agreement:**
    - CFGs struggle to express long-distance dependencies between elements in a sentence that need to agree, such as subject-verb agreement or noun-adjective agreement. For example, the CFG rule "S -> NP VP" doesn't specify how the verb tense in VP should match the person and number of the noun in NP.
    - Solutions to this problem can involve introducing additional rules or features in the CFG to capture these dependencies. For instance, you could have separate rules for different verb conjugations based on the properties of the subject NP. However, this can lead to an explosion in the number of rules and make the grammar unwieldy and less generalizable.

2. Subcategorization:

- Verbs in natural languages have specific requirements for the types of complements they can take. CFGs often lack the expressiveness to account for these subcategorization restrictions. For example, the verb "give" requires two arguments (a giver and a recipient), while "sleep" takes no arguments.
- One approach to address this is to use subcategorization frames associated with each verb in the lexicon. These frames specify the required and optional complement types, allowing the CFG to select appropriate syntactic rules based

on the verb. However, this requires manually annotating verbs with their subcategorization frames, which can be a time-consuming and error-prone process.

3. Movement:

- Natural languages allow for various types of movement phenomena, such as wh-movement (forming questions) and relative clause formation. These involve extracting elements from their original positions and placing them elsewhere in the sentence.
- CFGs, by their definition, are context-free and cannot directly handle such reordering operations. Several specialized techniques have been developed to express movement, such as transformations or movement rules within the CFG framework. However, these can add complexity and increase the computational cost of parsing sentences.

4. **Ambiguity:**
    - Natural language sentences can often have multiple valid syntactic parses, leading to ambiguity in interpretation. CFGs can generate multiple parse trees for such sentences, making it difficult to determine the intended meaning without additional context or semantic information.
    - Addressing ambiguity often involves incorporating statistical or probabilistic methods to choose the most likely parse based on word frequencies or semantic relationships, but this integration with CFGs can be challenging.
5. **Scalability:**
    - Parsing with large CFGs can become computationally expensive, especially for complex sentences or large corpora. This can limit their practical application in real-world NLP tasks.
    - Research efforts focus on developing more efficient parsing algorithms and techniques to reduce the computational cost of using CFGs.
