UNIT-II

Unit II: Word Level Analysis – N-grams, Evaluation, Smoothing, Interpolation & Backoff

---

1. Introduction to Language Modeling

- **Language Modeling** is the process of determining the probability of a sequence of words.
- It is widely used in applications like:
    - **Speech Recognition**
    - **Spam Filtering**
    - **Machine Translation**
    - **Predictive Text Input**

---

2. Types of Language Models

a. Statistical Language Models:

- Use **probabilistic models** to predict the next word in a sequence.
- Based on the frequency of word sequences in a training corpus.
- Example: **N-gram Models**

b. Neural Language Models:

- Use **neural networks** and **word embeddings**.
- Handle long-range dependencies better than statistical models.
- Used in deep learning applications like GPT, BERT.

---

3. N-gram Models

Definition:

An **N-gram** is a contiguous sequence of n items (words, letters) from a given text/speech sample.

Types:

- **Unigram**: single word (n=1)
- **Bigram**: sequence of two words (n=2)
- **Trigram**: sequence of three words (n=3)

N-gram Language Model:

- Predicts the next word using the previous n-1 words.
- Formula:

P(wn|w1,w2,...,wn−1)P(w_n | w_1, w_2, ..., w_{n-1})

4. 💥 Chain Rule of Probability – Detailed Explanation

✅ What is the Chain Rule of Probability?

The Chain Rule helps in calculating the joint probability of a sequence of events by breaking it into a series of conditional probabilities.

📌 Formula (General Form):

Let's say you want to find the probability of a sequence of words:

P(w1,w2,w3,...,wn)P(w_1, w_2, w_3, ..., w_n)

Using the Chain Rule of Probability, we can expand it as:

P(w1,w2,...,wn)=P(w1)·P(w2|w1)·P(w3|w1,w2)·····P(wn|w1,w2,...,wn−1)P(w_1, w_2, ..., w_n) = P(w_1) \cdot P(w_2 | w_1) \cdot P(w_3 | w_1, w_2) \cdot \dots \cdot P(w_n | w_1, w_2, ..., w_{n-1})

Or more compactly:

P(w1n)=∏i=1nP(wi|w1i−1)P(w_1^n) = \prod_{i=1}^{n} P(w_i | w_1^{i-1})

Where:

- **w1nw_1^n means words from 1 to n**
- **P(wi|w1i−1)P(w_i | w_1^{i-1}) is the probability of the word wiw_i given all the previous words.**

💡 Why Chain Rule in NLP?

In language modeling, we aim to compute the probability of a sentence or phrase:

Example: "This article is on NLP"

P("This article is on NLP")=P("This")·P("article"|"This")·P("is"|"This article")·...P(\text{"This article is on NLP"}) = P(\text{"This"}) \cdot P(\text{"article"} | \text{"This"}) \cdot P(\text{"is"} | \text{"This article"}) \cdot ...

This becomes computationally complex as the number of previous words increases.

🔁 Solution: Markov Assumption

We simplify using the Markov Assumption which states:

The probability of a word depends only on a limited number of previous words, not the entire history.

🎗️ **For Different N-grams:**

1. Unigram Model (No context)

P(w1,w2,...,wn)≈∏i=1nP(wi)P(w_1, w_2, ..., w_n) \approx \prod_{i=1}^{n} P(w_i)

Each word is independent of the others.

2. Bigram Model (One-word context)

P(w1,w2,...,wn)≈∏i=1nP(wi|wi−1)P(w_1, w_2, ..., w_n) \approx \prod_{i=1}^{n} P(w_i | w_{i-1})

Each word depends only on the previous word.

3. Trigram Model (Two-word context)

P(w1,w2,...,wn)≈∏i=1nP(wi|wi−2,wi−1)P(w_1, w_2, ..., w_n) \approx \prod_{i=1}^{n} P(w_i | w_{i-2}, w_{i-1})

Each word depends on the previous two words.

📌 **Example using Bigram:**

Sentence: "Natural Language Processing"

Let's say we want to calculate:

P("Natural Language Processing")≈P("Natural"|<start>)·P("Language"|"Natural")·P("Processing"|"Language")P(\text{"Natural Language Processing"}) \approx P(\text{"Natural"} | <start>) \cdot P(\text{"Language"} | \text{"Natural"}) \cdot P(\text{"Processing"} | \text{"Language"})

From the table in the image:

- **P(Natural|<start>)=0.12P(\text{Natural} | <start>) = 0.12**
- **P(Language|Natural)=0.5P(\text{Language} | \text{Natural}) = 0.5**
- **P(Processing|Language)=0.3P(\text{Processing} | \text{Language}) = 0.3**

So,

P("Natural Language Processing")=0.12×0.5×0.3=0.018P(\text{"Natural Language Processing"}) = 0.12 \times 0.5 \times 0.3 = 0.018

✅ **Key Takeaways for Exams:**

- **The Chain Rule allows decomposition of a joint probability into conditional probabilities.**
- **In NLP, we apply it to predict the probability of a sentence.**
- **Since full history is impractical, Markov Assumption is used to approximate with N-gram models.**
- **Bigram, Trigram, etc., limit the history to 1, 2... previous words.**
- **This forms the core logic behind traditional statistical language models.**

---

5. Python Example using NLTK:

```
import string, random
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('reuters')
from nltk.corpus import reuters
from nltk import FreqDist

# Input the sentences
sents = reuters.sents()

# Remove punctuation and stopwords
stop_words = set(nltk.corpus.stopwords.words('english'))
removal_list = list(stop_words) + list(string.punctuation) + ['lt', 'rt']
```

---

6. Evaluation of N-gram Models

a. Entropy (H):

- Measures the amount of information.
- Formula:

H(p)=∑xp(x)(−log2(p(x)))H(p) = \sum_{x} p(x) (-\log_2(p(x)))

b. Cross-Entropy:

- Measures the accuracy of a model on test data.
- Formula:

H(p)=1n∑i=1n(−log2(p(wi|wi−1)))H(p) = \frac{1}{n} \sum_{i=1}^{n} (-\log_2(p(w_i | w_{i-1})))

c. Perplexity:

- Inverse probability of the test set normalized by word count.
- Measures how well the model predicts.
- Formula:

2Cross-Entropy2^{\text{Cross-Entropy}}

---

7. Example: Word Prediction Using N-gram

Sentence: **"Natural Language Processing"**

Given Probabilities:

Initial Word Probabilities:

Word	P(word |)
The	0.4
Processing	0.3
Natural	0.12
Language	0.18

After "Natural":

Word	P(word | "Natural")
The	0.05
Processing	0.3
Natural	0.15
Language	0.5

---

8. Interpolation and Backoff

- **Backoff**: If a higher-order N-gram has zero frequency, back off to lower-order N-gram (e.g., from trigram to bigram).
- **Interpolation**: Combines different N-gram probabilities with weights (e.g., combine unigram + bigram + trigram).

✅ **Key Points to Remember**

- N-gram models are simple but effective for many NLP tasks.
- Bigram and trigram models improve over unigram by adding context.
- Entropy, cross-entropy, and perplexity are important evaluation metrics.
- Interpolation and backoff handle zero-probability issues in sparse data.

Based on the provided images, here's a **detailed and student-friendly explanation** of the key concepts and formulas related to **Language Modeling, Smoothing, and Vocabulary Tasks** — useful for a **10-mark exam answer**.

UNIT-II

🔍 Language Modeling and Smoothing Techniques – Detailed Notes

Language models estimate the probability of sequences of words. However, many possible sequences may never appear in the training data, leading to **zero probabilities**. To handle this, we use **smoothing techniques**.

---

1. Unigram Prior Smoothing

- **Definition**: In this method, we replace the probability of a word P(Wi)P(W_i) with a smoothed value 1V\frac{1}{V} (where VV is the vocabulary size).
- **Purpose**: Helps avoid assigning zero probability to unseen words in Unigram models.
- **Note**: While it helps in interpolation, it's not enough for complex language models.

---

2. Backoff

- **Definition**: When there isn't enough data to use a higher-order language model (e.g., trigram), we **"back off"** to a simpler model (e.g., bigram or unigram).
- **Use Case**: Prevents zero probabilities when data is sparse.

---

3. Interpolation

- **Definition**: Combines different n-gram models (e.g., unigram, bigram, trigram) instead of choosing just one.
- **Types**:
    1. **Simple Interpolation**:

P(Wi)=L1⋅P(Wi)+L2⋅P(Wi|Wi−1)+L3⋅P(Wi|Wi−2,Wi−1)P(W_i) = L_1 \cdot P(W_i) + L_2 \cdot P(W_i | W_{i-1}) + L_3 \cdot P(W_i | W_{i-2}, W_{i-1})

Subject to: L1+L2+L3=1L_1 + L_2 + L_3 = 1

    2. **Conditional Lambda Interpolation**:
        - Lambdas L1,L2,L3L_1, L_2, L_3 vary based on context.
        - Use a development dataset to determine optimal lambda values.

---

4. Open vs Closed Vocabulary Tasks

- **Open Vocabulary**: We do **not** know all the words in advance (realistic scenario).
- **Closed Vocabulary**: All words are known in advance (limited use cases).
- **Handling OOV (Out-of-Vocabulary)**:
    - Use a special token: <UNK>

- Low probability words are replaced with <UNK> during training and testing.
- The model then computes probability for unseen words based on <UNK> token.

---

5. Smoothing for Large N-Grams (Web-scale N-Grams)

- **Technique**: **Stupid Backoff**
- Formula:

P(Wi|Wi−k…Wi−1)={if count is low: k·P(Wi|Wi−k+1…Wi−1)else: use probability directlyP(W_i | W_{i-k} \ldots W_{i-1}) = \begin{cases} \text{if count is low: } k \cdot P(W_i | W_{i-k+1} \ldots W_{i-1}) \\ \text{else: use probability directly} \end{cases}

- **Note**:
  - No probabilities, only scores.
  - Works well for very large datasets.
  - **Add-K Smoothing** is not effective here.

---

6. Good Turing Smoothing

- **Idea**: Adjust probabilities of unseen events based on frequency of frequencies.
- Definitions:
  - NcN_c: Number of items seen exactly cc times.
  - N1N_1: Items seen once.
  - NN: Total observations.
- Formula for unseen:

P(new things)=N1NP(\text{new things}) = \frac{N_1}{N}

- Generalization:

c∗=(c+1)·Nc+1NcandP=c∗Nc^* = \frac{(c+1) \cdot N_{c+1}}{N_c} \quad \text{and} \quad P = \frac{c^*}{N}

- **Use Case**: Works well when some items are seen only once or not at all.

---

7. Kneser-Ney Smoothing

- **Improved smoothing** technique over Good Turing.
- Focuses on **novel continuation** instead of just occurrence.
- Formula:

PKN(wi|wi−1)=max(count(wi−1wi)−d,0)count(wi−1)+λ(wi−1)⋅Pcont(wi)P_{KN}(w_i | w_{i-1}) = \frac{\max(\text{count}(w_{i-1} w_i) - d, 0)}{\text{count}(w_{i-1})} + \lambda(w_{i-1}) \cdot P_{cont}(w_i)

- **Components**:
    - Pcont(w)P_{cont}(w): Continuation probability = (Number of unique contexts ww appears in) / (Total bigrams)
    - λ(wi−1)\lambda(w_{i-1}): Normalization constant
- **Explanation**:
    - Frequent words (e.g., “Francisco” only follows “San”) get **low continuation probability**.
    - Better at handling **lower-order n-gram smoothing**.

---

8. Word Classes (Part of Speech - PoS)

- **Definition**: Words grouped into categories like noun, verb, adjective, etc.
- **Purpose**:
    - Helps determine role of word in a sentence.
    - Useful in predicting most probable following word.
- **Examples of Corpora**:
    - Penn Treebank (45 tags)
    - Brown Corpus (87 tags)
    - FT7 Tagset (146 tags)
- **Example**:
    - "he" (personal pronoun) vs. "his" (possessive pronoun)
    - Predicting following words becomes easier with correct PoS.

✅ **Summary for Exams**

Topic	Key Point
Unigram Prior Smoothing	Replaces 1V\frac{1}{V} with P(Wi)P(W_i) for unseen words
Backoff	Move to lower n-gram when data is insufficient
Interpolation	Combine uni/bi/tri-grams; use lambdas
Open vs Closed Vocabulary	Handle OOV with <UNK> token

UNIT-II

Topic	Key Point
Stupid Backoff	Simple backoff technique for web-scale data
Good Turing Smoothing	Adjust frequency using frequency of frequency
Kneser-Ney Smoothing	Improves upon Good Turing by using continuation probability
Word Classes	Use PoS tags for better word prediction

Part of Speech (POS) Tagging and Word Classes

1. Introduction to Word Classes (Open Classes)

Word classes (also known as parts of speech) are categories of words with similar grammatical behavior. The four **largest open classes** in most languages are:

- **Nouns**
- **Verbs**
- **Adjectives**
- **Adverbs**

These classes are called **open classes** because new words are frequently added to them.

---

2. Explanation of Word Classes

A. Nouns

- Represent people, places, objects, ideas, or feelings.
- Can be **concrete** (e.g., ship, table) or **abstract** (e.g., freedom, relationship).
- **Proper nouns** (e.g., Marco, Italy) vs. **Common nouns** (e.g., book, lecture).
- **Count nouns** (e.g., apples – have plural form) vs. **Mass nouns** (e.g., snow – singular only).

B. Verbs

- Indicate **actions** or **states** (e.g., write, go, be).
- Inflect for **tense** and **aspect**:
    - eat, eats, eating, ate, eaten
- **Auxiliary verbs**: Help form tenses (e.g., be, have, do).

C. Adjectives

- Describe **qualities or properties**.

- Examples: colors (white), age (old), size (big), quality (good).
- Modify nouns (e.g., a *beautiful* girl).

D. Adverbs

- Modify **verbs**, **adjectives**, or other **adverbs**.
- Types of adverbs:
    - **Direction/Location**: here, up, there
    - **Degree**: very, extremely
    - **Manner**: quickly, smartly
    - **Time**: yesterday, later, Monday
- Adverbs are a **heterogeneous class**, meaning some resemble nouns (e.g., “Monday” in "We meet on Mondays").

---

3. POS Tagging (Part-of-Speech Tagging)

Definition:

POS tagging is the process of assigning a **grammatical tag** (e.g., noun, verb) to each word in a sentence.

Key Points:

- Tags are chosen **based on context**.
- Some words are ambiguous and can have **multiple tags** (e.g., "can" as a verb or noun).
- Example:
    - "Did you box your belongings?"
        - “Did” → VBD (verb, past tense)
        - “box” → could be noun or verb (tag based on context)

---

4. POS Tagging Algorithms

A. Rule-Based Taggers

- Use **handcrafted grammar rules**.
- Example rule: "A word is likely a noun if preceded by a determiner."
- Use **lexicons** and **morphological rules** (e.g., suffix -ed = past tense verb).

Example Table from ENGTWOL:

Text	PoS	Features
Pavlov	N	NOM SG PROPER
had	V	PAST VFIN SVO
shown	PCP2	SVOO SVO SV
that	ADV/PRON/DET/CS	DEM/SG etc.

B. Probabilistic Taggers (Stochastic)

- Based on **probability/statistics**.
- Estimate the **likelihood of a tag** in a given context.

Types:

1. **Word Frequency Approach**:
    - Assigns the most frequent tag from training data.
    - Limitation: May produce invalid tag sequences.
2. **Tag Sequence Probabilities** (n-gram model):
    - Considers **probabilities of tag sequences**.
    - Example: If previous tags were “DT JJ”, next is likely “NN”.

Formula (n-gram model):

P(t□ | t□₋₁, t□₋₂, ..., t□₋□)
Where:

- P = probability
- t = tag
- n = current position
- k = number of previous tags considered

C. Other Approaches

- **Machine learning-based classifiers**.
- Use features from surrounding words.
- Learn from **examples**, not fixed rules.

---

5. Rules in Rule-Based Tagging (Example from ENGTWOL)

ADVERBIAL-THAT RULE:

If “that” is followed by an adjective, adverb, or quantifier

AND sentence boundary is near

AND previous word is a verb like “consider”

Then eliminate non-ADV tags

Else eliminate ADV tag

ENGTWOL contains **1100+ rules** and also includes **probabilistic constraints**.

---

6. Properties of Stochastic POS Tagging

- Based on **tag probabilities**.
- Requires **training corpus**.
- Cannot handle unknown words (outside the training set).
- Uses **testing corpus** for evaluation.

✅ **Conclusion:**

POS tagging is a crucial process in NLP to identify the grammatical role of words. There are different techniques for tagging:

- Rule-based (uses grammar rules)
- Probabilistic (uses data statistics)
- Machine learning-based (uses classifiers)

Understanding the **word classes** (nouns, verbs, adjectives, adverbs) and how they are tagged helps in building many language-processing applications like grammar checkers, translation tools, and chatbots.

📌 **Useful for Exams:**

- Define open classes.
- Describe each class with examples.
- Define POS tagging and its need.
- Explain tagging algorithms: rule-based, stochastic.
- Mention stochastic formulas (n-gram).
- Include properties and examples from ENGTWOL.

UNIT-II

Transformation-Based Tagging (Brill Tagging)

(10-Mark Exam Answer)

Definition:

Transformation-Based Tagging (TBL), also known as **Brill Tagging**, is a **rule-based approach** to **Part-of-Speech (POS) tagging** that combines the advantages of **rule-based** and **machine learning** approaches.

It applies a sequence of transformation rules to improve the accuracy of initial POS tags assigned to words in a text.

What is Transformation-Based Learning (TBL)?

TBL is a **machine learning technique** that starts with an initial solution (e.g., assigning the most frequent POS tag) and **improves it iteratively** using a series of **transformation rules** learned from data.

- It is **rule-driven**, but the rules are **learned automatically**, not manually written.
- It works well in **classification tasks** like POS tagging.

🔁 **How TBL Works (Steps in TBL)**

1. **Start with a solution**
    - Assign an initial POS tag to each word in the text.
    - Usually, the most frequent tag from the training corpus is assigned.
2. **Choose the most beneficial transformation**
    - The system finds a rule that **corrects the most number of mistakes**.
    - Example rule: *If the word is "can" and it follows a noun, change the tag from noun to verb.*
3. **Apply the rule to the problem**
    - Apply the selected transformation to the data.
    - This improves the tagging accuracy.
4. **Repeat the process**
    - Continue finding and applying new rules until **no more improvements** can be made.

✅ **Advantages of Transformation-Based Learning (TBL)**

- **Simple and Understandable Rules:**
  A small number of rules are easy to learn and apply.
- **Easy Debugging:**
  Rules are **transparent and interpretable**, making it easy to find and fix mistakes.
- **Reduces Complexity:**
  Combines human-generated and machine-learned rules for better accuracy.
- **Faster than Stochastic Models:**
  TBL is often **faster** than models like the **Hidden Markov Model (HMM)** in terms of tagging, after training.

❌ **Disadvantages of Transformation-Based Learning (TBL)**

- **No Tag Probabilities:**
  TBL does **not provide probability scores** for each tag, which is useful in some applications.
- **Training Time is Long:**
  Especially on **large datasets (corpora)**, training takes a long time due to repeated rule evaluations.

📌 **Key Points for Exams:**

- TBL is also called **Brill Tagging**.
- It is **rule-based but uses machine learning** to learn the rules.
- Combines features of **rule-based and stochastic** taggers.
- Works by **starting with a basic solution and applying transformation rules** to improve tagging.
- Does **not use probabilities**, unlike statistical taggers.
- Is **interpretable, efficient (after training), and accurate** in many cases.

Example Rule (Just for Understanding):

If word = “can” and previous tag = noun, then change tag from noun to verb.

This kind of transformation rule is **learned automatically** during training by evaluating its benefit in correcting errors.

✍ **Conclusion:**

Transformation-Based Tagging is an important technique in Natural Language Processing (NLP). It offers a balance between **accuracy**, **efficiency**, and **interpretability**, making it a strong candidate for

tasks like POS tagging. Though it may lack probability modeling and have longer training times, its ease of debugging and clarity make it a widely studied method in linguistics and AI.

📘 **Hidden Markov Model (HMM) for POS Tagging – 10 Marks Answer**

🔷 **Definition: What is a Hidden Markov Model (HMM)?**

A **Hidden Markov Model (HMM)** is a statistical model that is used when the system being modeled is a **Markov process with hidden states**.

- It is called "hidden" because the actual process (states) is not visible to us.
- What we can see are **observations**, which are produced by these hidden states.

🔷 **Example (Explained Simply)**

Imagine you are tossing coins, but:

- You don't know which coin is being used at each toss.
- You only observe the outcomes: **Heads (H)** or **Tails (T)**.

Suppose there are two different biased coins:

- Coin 1: gives Heads with probability P_1
- Coin 2: gives Heads with probability P_2

These coin selections are hidden (we don't know which one is being used), but we can observe the outcomes (H or T). This kind of setup is modeled using HMM.

🔷 **Key Components of an HMM**

An HMM is defined by the following elements:

1. **N** – Number of hidden states
    - Example: Number of coins = 2 (states = coin 1 and coin 2)
2. **M** – Number of observable symbols
    - Example: H (Heads), T (Tails), so M = 2
3. **A** – State Transition Probability Matrix
    - Represents the probability of moving from one state to another.
    - Example Matrix:

A=[a11a12a21a22]A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}

- aija_{ij} = probability of transitioning from state i to state j

- Rows must sum to 1:

a11+a12=1,a21+a22=1a_{11} + a_{12} = 1, \quad a_{21} + a_{22} = 1

4. **P** – Observation probability
    - P_1 = probability of Heads from coin 1
    - P_2 = probability of Heads from coin 2
5. **π (Pi)** – Initial state distribution
    - Probability of starting in each state

🔷 Use of HMM in POS Tagging

POS Tagging is the process of assigning part-of-speech tags (like noun, verb, adjective) to each word in a sentence.

We can model this as an HMM:

- **States** = POS tags (Noun, Verb, etc.)
- **Observations** = Words in the sentence

In this model:

- Tags (POS) are hidden.
- Words are observable.

🔷 Goal in POS Tagging

We aim to **find the most likely sequence of tags (ff)** for a given sequence of words (W).

Mathematically, we want to **maximize** this probability:

P(ff|W)P(ff | W)

Where:

- ff=f1,f2,…,fTff = f_1, f_2, \ldots, f_T: Sequence of POS tags
- W=w1,w2,…,wTW = w_1, w_2, \ldots, w_T: Sequence of words

🔷 Applying Bayes' Rule

To solve this, we use **Bayes' Theorem**:

P(ff|W)=P(W|ff)·P(ff)P(W)P(ff | W) = \frac{P(W | ff) \cdot P(ff)}{P(W)}

- Since P(W)P(W) is constant, we can ignore it for maximization.
- So, the problem reduces to maximizing:

P(W|ff)·P(ff)P(W | ff) \cdot P(ff)

🔹 Assumptions to Simplify the Model

1. n-gram Assumption

The probability of a tag depends only on a few previous tags (not all previous tags). This is known as the **n-gram model**:

- **Bigram model** (depends on previous tag only):

P(fi|fi−1)P(f_i | f_{i-1})

- **Trigram model** (depends on two previous tags):

P(fi|fi−2,fi−1)P(f_i | f_{i-2}, f_{i-1})

2. Initial Tag Probability

At the start of the sentence, we assume an initial probability:

P(f1)=Initial probability of the first tagP(f_1) = \text{Initial probability of the first tag}

🔹 Final Objective Function

We combine everything to find the best sequence of tags (ff):

max[P(w1,w2,...,wT|f1,f2,...,fT)·P(f1,f2,...,fT)]\max \left[P(w_1, w_2, ..., w_T | f_1, f_2, ..., f_T) \cdot P(f_1, f_2, ..., f_T) \right]

✅ Key Points to Remember

- HMM is a model with **hidden states** (like POS tags) and **observable outputs** (like words).
- It is widely used in **natural language processing (NLP)**, especially **POS tagging**.
- Uses **state transition probabilities**, **observation probabilities**, and **initial probabilities**.
- Simplifies complex probability calculations using **Bayes' Theorem** and **n-gram models**.

✍️ Conclusion

HMM provides a mathematical and probabilistic approach to solving POS tagging problems where the actual sequence of tags is hidden, and only the words are observed. With proper training data and assumptions like bigram or trigram models, HMMs can accurately predict the most probable sequence of tags for a given sentence.

UNIT-II

◆ Hidden Markov Model (HMM) and POS Tagging

✓ Definition of HMM

A **Hidden Markov Model (HMM)** is a statistical model where the system is modeled as a **Markov process** with **hidden states**. These states cannot be observed directly but influence observable events.

In simple words:

- We **observe outputs** (like coin flips, words, etc.),
- But the **process generating those outputs is hidden** (like the biased coin being used or the part of speech).

✓ Key Elements of HMM

1. **N**: Number of hidden states (e.g., coins in the coin toss example).
2. **M**: Number of possible observations (e.g., heads/tails or words).
3. **A**: State transition probability matrix

A=[a11a12a21a22]A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}

   - aija_{ij}: Probability of moving from state i to state j.
   - Each row must sum to 1.

4. **P (B)**: Observation symbol probability distribution for each state (like probability of heads/tails for each coin).
5. **I (π)**: Initial probability distribution (starting probabilities for each state).

✓ HMM for POS Tagging

POS (Part-of-Speech) Tagging means assigning tags like noun, verb, adjective to each word in a sentence.

- In HMM-based POS tagging:
  - The **words are observations**.
  - The **tags are hidden states**.
  - The goal is to find the most likely tag sequence (ff) for a given word sequence (W).

✓ Mathematical Objective

Find the tag sequence ffff that **maximizes the probability**:

P(ff|W)P(ff | W)

Using **Bayes' Rule**:

P(ff|W)=P(W|ff)·P(ff)P(W)P(ff | W) = \frac{P(W | ff) \cdot P(ff)}{P(W)}

Since P(W)P(W) is constant for all tag sequences, we can ignore it:

P(ff|W)∝P(W|ff)·P(ff)(1)P(ff | W) \propto P(W | ff) \cdot P(ff) \tag{1}

✅ **First Assumption (Tag Sequence Probability)**

We simplify P(ff)P(ff) using an **n-gram model**:

- **Bigram Model**:

P(ff)=∏i=1TP(ffi|ffi−1)P(ff) = \prod_{i=1}^{T} P(ff_i | ff_{i-1})

- **Initial Tag Probability**:

P(ff1)=Initial Probability of tag ff1P(ff_1) = \text{Initial Probability of tag ff}_1

✅ **Second Assumption (Word Emission Probability)**

We assume the word depends only on the current tag:

P(W|ff)=∏i=1TP(Wi|ffi)P(W | ff) = \prod_{i=1}^{T} P(W_i | ff_i)

So now equation (1) becomes:

Goal: Maximize ∏i=1TP(Wi|ffi)·P(ffi|ffi−1)(2)\text{Goal: Maximize } \prod_{i=1}^{T} P(W_i | ff_i) \cdot P(ff_i | ff_{i-1}) \tag{2}

✅ **Estimating Probabilities from Data**

Using training data:

- **Transition probability**:

P(ffverb|ffnoun)=#times verb follows noun#times noun appearsP(ff_{\text{verb}} | ff_{\text{noun}}) = \frac{\# \text{times verb follows noun}}{\# \text{times noun appears}}

- **Emission probability**:

P(W|ff)=#times word W appears with tag ff#times tag ff appearsP(W | ff) = \frac{\# \text{times word W appears with tag ff}}{\# \text{times tag ff appears}}

🔷 **Maximum Entropy (MaxEnt) Models**

✅ **Definition**

UNIT-II

MaxEnt is a machine learning model used in NLP classification problems. It tries to **combine all known information** (features) to make predictions.

It chooses the **most uniform (high entropy)** model that still fits the training data.

✅ Maximum Likelihood Principle

We try to **find weights** ww that maximize the probability of the correct label for each training instance:

w^=argmaxw∏i=1nf(yi|xi;w)\hat{w} = \arg\max_w \prod_{i=1}^{n} f(y_i | x_i; w)

✅ Log-Linear Model

- The model predicts labels using:

f(x,y)=∑jwjfj(x,y)f(x, y) = \sum_j w_j f_j(x, y)

- Where wjw_j is the weight of feature fjf_j, which indicates how important that feature is in predicting label yy.

✅ Training using Gradient Descent

Steps:

1. Initialize weights randomly.
2. Loop through the dataset.
3. Update weights using the gradient of the likelihood.
4. Repeat until convergence.

◆ Maximum Entropy Markov Model (MEMM)

✅ What is MEMM?

MEMM is a sequence model like HMM but combines:

- The **Markov property** (past state influences current)
- The **MaxEnt model** (probabilistic classifier)

✅ Formula in MEMM

P(y1,y2,...,yn|x1,...,xn)=∏i=1nP(yi|yi−1,xi)P(y_1, y_2, ..., y_n | x_1, ..., x_n) = \prod_{i=1}^{n} P(y_i | y_{i-1}, x_i)

Each state transition is modeled using MaxEnt.

✅ **Shortcoming – Label Bias Problem**

MEMM suffers from **label bias**, meaning:

- States with fewer transitions become biased because probabilities are normalized **within states**, ignoring observations.

✅ **MEMM Probability Formula**

P(yi|yi−1,x)=1Z(yi−1,x)exp(∑i=1nwifi(yi,yi−1,x))P(y_i | y_{i-1}, x) = \frac{1}{Z(y_{i-1}, x)} \exp\left(\sum_{i=1}^{n} w_i f_i(y_i, y_{i-1}, x)\right)

Where:

- ZZ is a normalization factor.
- fif_i are feature functions.
- wiw_i are weights.

🔷 **Applications of MaxEnt**

- **Sentiment Analysis**
- **Election Preferences**
- **Medical Diagnosis**

These models help in tasks like POS tagging, text classification, and structured prediction.

✅ **Summary for Exam (10 Marks Answer)**

🔹 **HMM:**

- Statistical model with hidden states.
- Used for POS tagging.
- Works with transition and emission probabilities.
- Uses **Bayes Rule** to estimate most likely tag sequence.

🔹 **MaxEnt:**

- Classification model using features.
- Learns weights for features.
- Uses **maximum likelihood** and **log-linear models**.

🔹 **MEMM:**

- Combines MaxEnt and Markov Chains.

- Models sequences with dependencies.
- Has label bias issue.
