



Prepared by Group 3

Named Entity Recognition

Aarohi Mishra (39)

Pearl Ochani (41)

Poorva Pathak (45)





What is NER

Named Entity Recognition (NER) is a subtask of Natural Language Processing (NLP) that identifies and classifies named entities (like people, places, organizations, dates, etc.) from text.

Example:

Sentence: "Virat Kohli plays for India in the ICC World Cup 2023."

NER output →

- Virat Kohli → PERSON
- India → LOCATION
- ICC World Cup 2023 → EVENT



Approaches to NER

Era	Approach	Key Idea
1990s	Rule-based	Pattern matching using linguistic rules and dictionaries
2000s	Statistical (ML-based)	Probabilistic models: HMM, MEMM, CRF
2010s	Neural / Deep Learning	BiLSTM, CNN, Transformers



Challenges in NER for Indian Languages

No Capitalization

English uses uppercase to detect names; Indian scripts (like Devanagari) lack this cue.

Rich Morphology

Words change forms by gender, number, and inflection (e.g., “दिल्ली” vs “दिल्ली में”).

Compound Words

“रामनगर”, “महात्मागांधीनगर” – entities merge into longer tokens.

Free Word Order

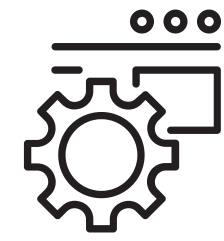
The same sentence can appear with different word orders.

Ambiguous Names

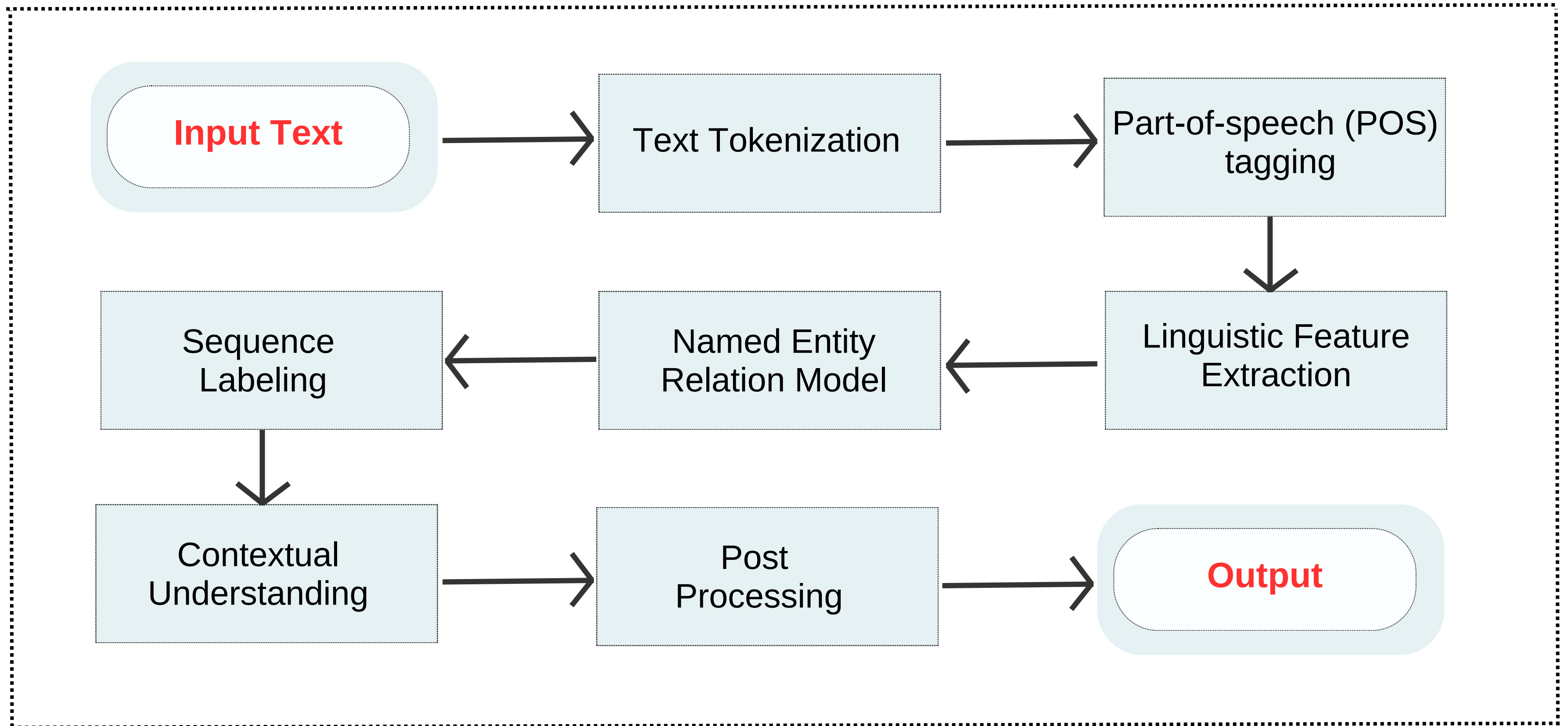
Common nouns can be names (“सूरज”, “राज”) or regular words.

Code-Mixed Text

Frequent mix of English and Hindi in same sentence.



Working of NER



Hidden Markov Model (HMM)

Hidden Markov Models provide a **probabilistic** framework for sequential labeling problems like POS Tagging and NER.

Why HMM for NER?

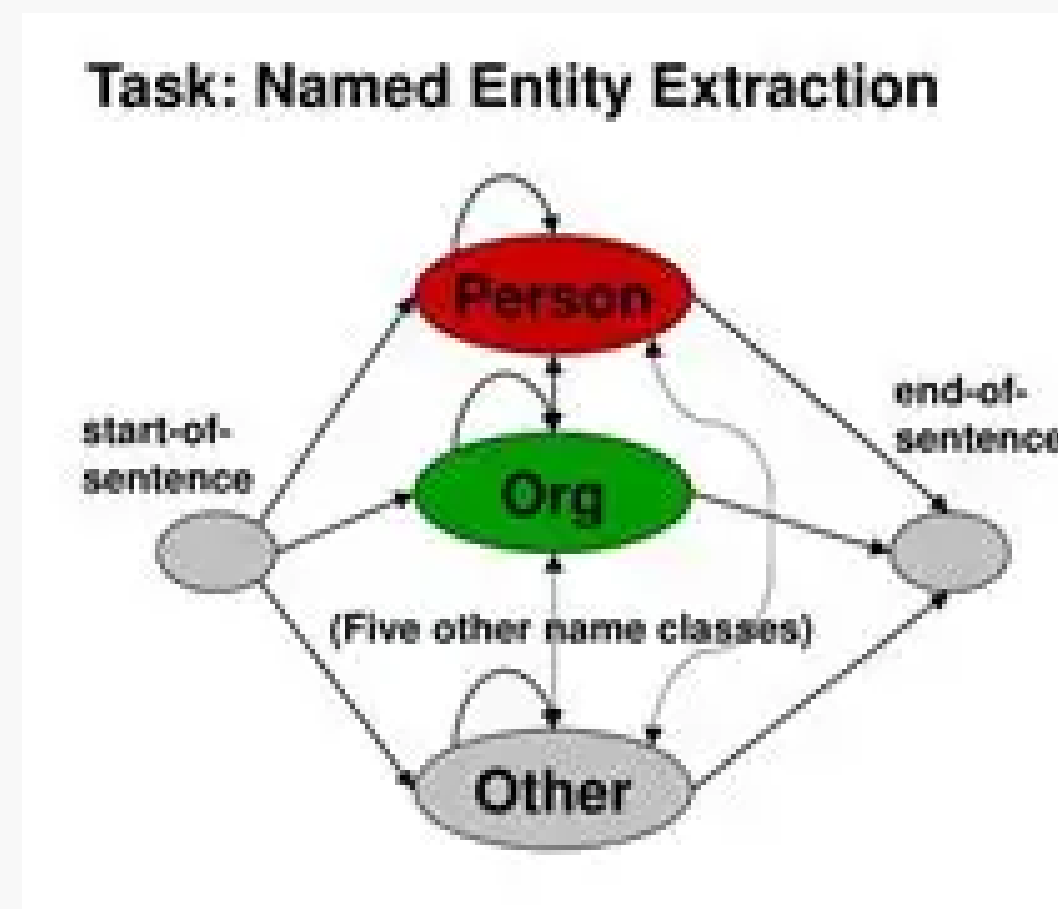
Given a sentence $\mathbf{W}=[w_1, w_2, \dots, w_n]$, assign each word a tag $\mathbf{T}=[t_1, t_2, \dots, t_n]$ like **PERSON**, **LOCATION**, **ORGANIZATION**, or **O** (other).

HMM is a probabilistic model for sequences that learns:

- how likely one entity type (tag) follows another, and
- how likely each word is given an entity type.

So in NER:

- Hidden states = entity tags (e.g., PERSON, LOCATION, ORG, O)
- Observations = words in the sentence



Components of HMM

Transition Probabilities (A)

The probability of transitioning from one entity label to another.

$$\mathbf{A} = \{a_{i,j} = P(\text{tag}_j \mid \text{tag}_i)\}$$

This models how likely one entity tag follows another in the sentence.

Emission Probabilities (B)

The probability of a word being generated given a particular entity tag.

$$\mathbf{B} = \{b_{i,k} = P(\text{word}_k \mid \text{tag}_i)\}$$

Determines how likely a word appears for each named entity class.

Initial State Distribution (π)

The probability of starting the sequence with a particular entity tag.

$$\pi = \{\pi_i = P(\text{tag}_1 = i)\}$$

Determines which entity tag is most likely to begin a sentence or text sequence.

Combined Insight:

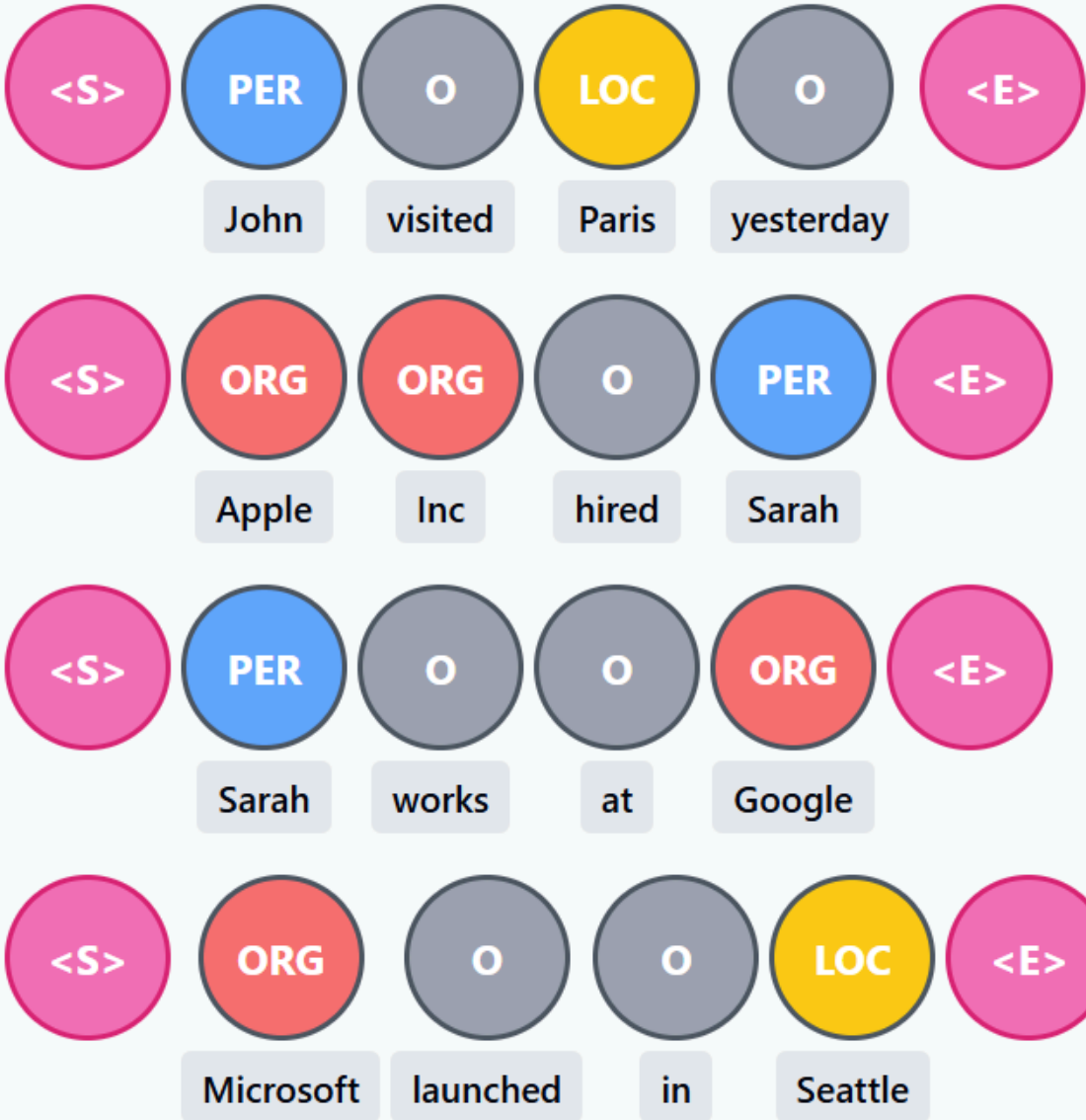
- π initializes the likelihood of starting tags.
- \mathbf{A} models sequential dependencies between entity labels.
- \mathbf{B} models lexical evidence for each label.

Probability Calculations in HMM

Tag Legend:

PER = Person (व्यक्ति) LOC = Location (स्थान) ORG = Organization (संगठन) O = Other (अन्य)

Training Sentences (English)



Transition Counts (from training data)

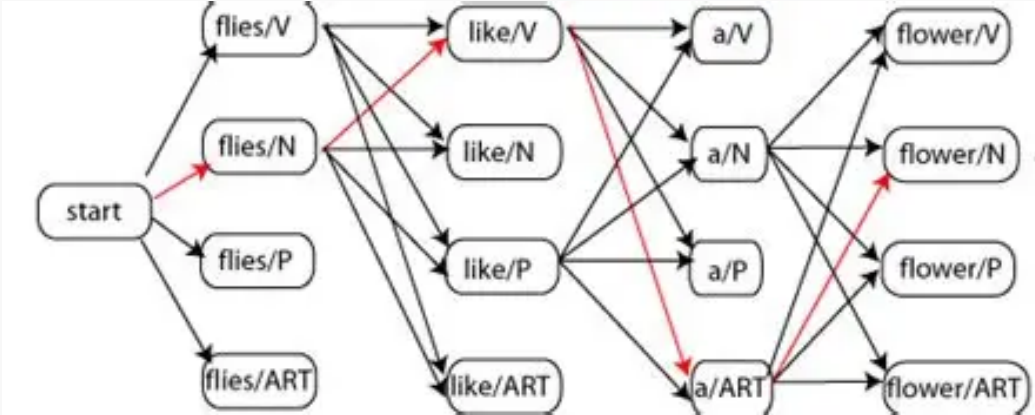
	PER	LOC	ORG	O	<E>
<S>	?	?	?	?	0
PER	1	2	1	5	3
LOC	1	0	1	4	2
ORG	2	1	0	3	2

Transition Probabilities

	PER	LOC	ORG	O	<E>
<S>	3/12	2/12	2/12	5/12	0
PER	1/12	2/12	1/12	5/12	3/12
LOC	1/8	0	1/8	4/8	2/8
ORG	2/8	1/8	0	3/8	2/8

Working of HMM for NER after Probability Calculations:

1. Train on labeled data to learn transition and emission probabilities
2. For new sentence, **Viterbi** algorithm finds the most likely tag sequence
3. Dynamic programming ensures we find the optimal path efficiently



Viterbi Algorithm in HMM

The Viterbi algorithm finds the most likely sequence of POS tags using **dynamic programming** and choosing the path with highest probability.

Algorithm Steps

1. **Initialization:** Calculate initial probabilities for first word
2. **Forward Pass:** For each word, find best path to each possible tag
3. **Backtracking:** Trace back the optimal path

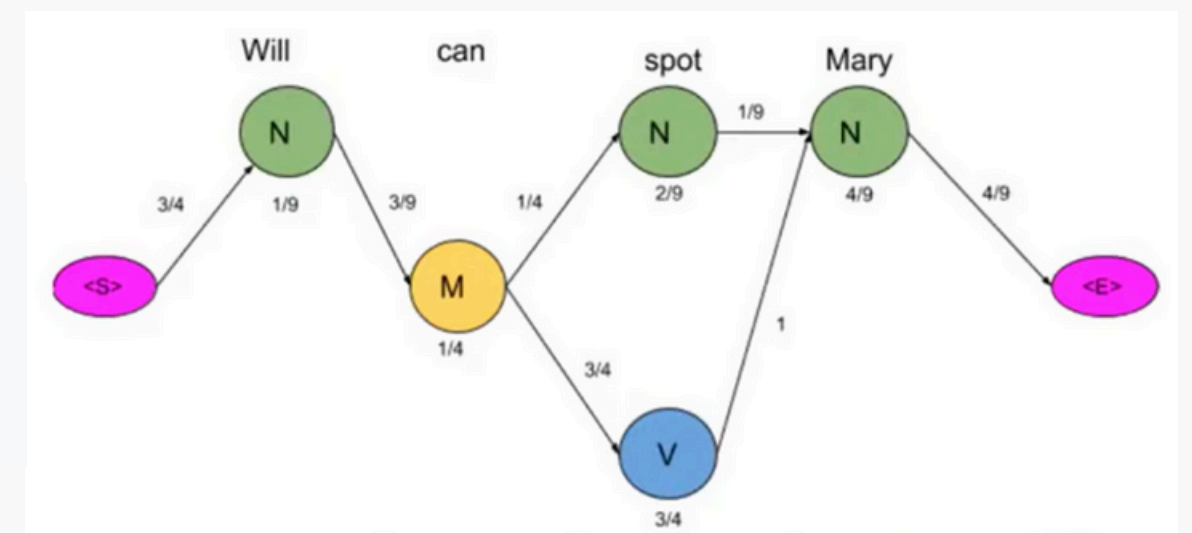
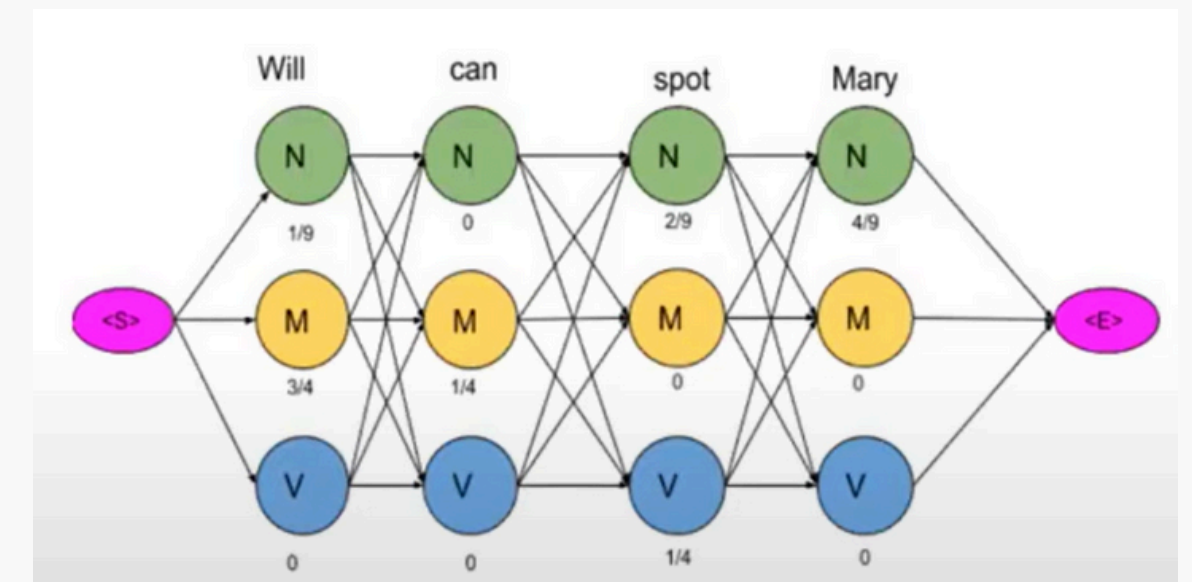
Mathematical Formulation

$\delta_t(i)$ = probability of best path ending in state i at time t

$$\delta_1(i) = \pi(i) \times P(\text{word}_1 \mid \text{tag}_i)$$

$$\delta_t(j) = \max[\delta_{t-1}(i) \times P(\text{tag}_j \mid \text{tag}_i)] \times P(\text{word}_t \mid \text{tag}_j)$$

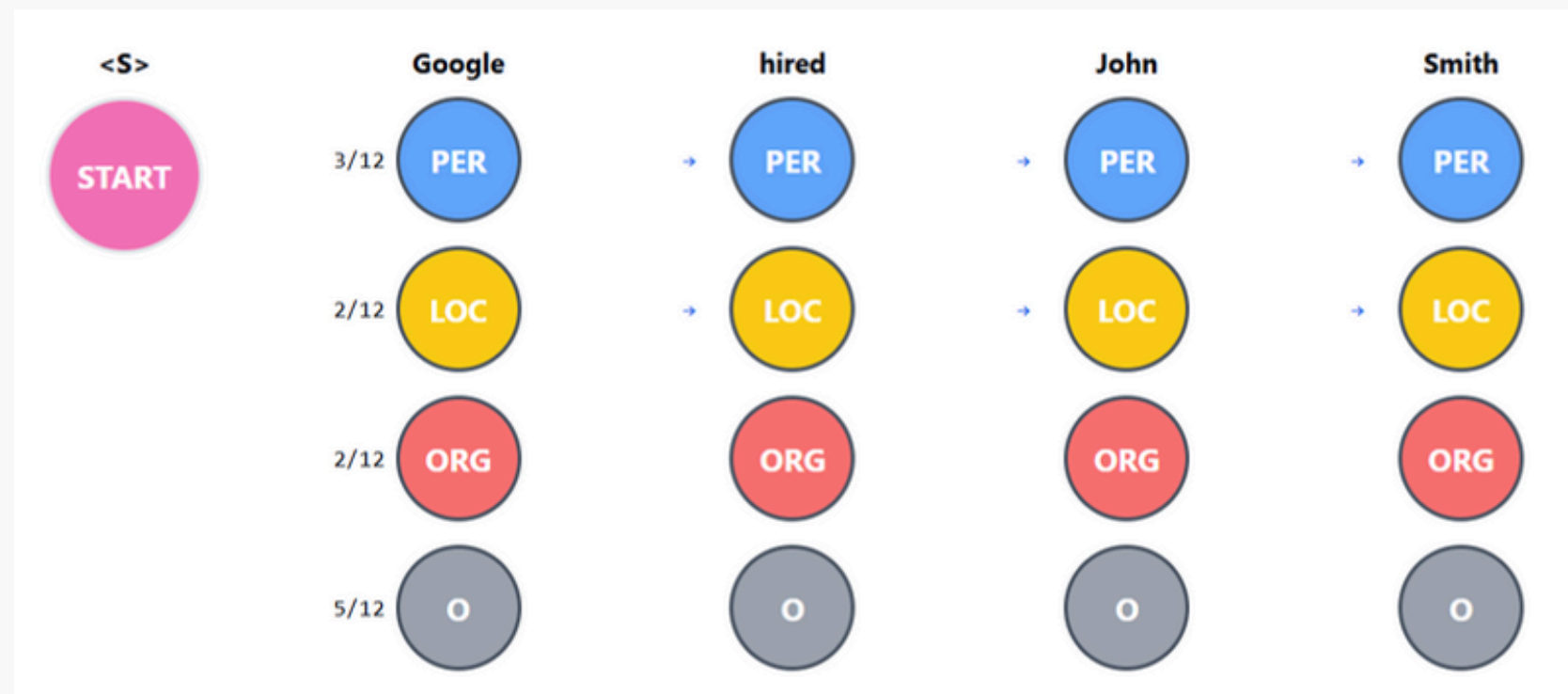
Assumption: Current tag depends only on the previous tag, not the entire history.



Final Result (After Viterbi)

Test with new sentence: "Google hired John Smith"

Viterbi Graph (simplified):



Final Result (After Viterbi)

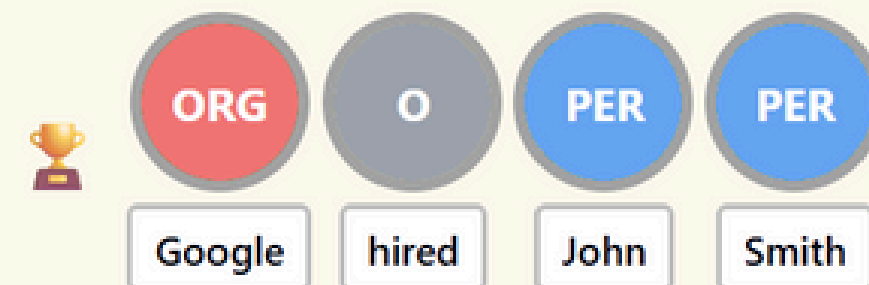
Comparison of Top 2 Paths:

Path 1 (HIGHEST):

$$\langle S \rangle \rightarrow \text{ORG} \rightarrow \text{O} \rightarrow \text{PER} \rightarrow \text{PER} \rightarrow \langle E \rangle = \frac{2}{12} \times \frac{5}{9} \times \frac{3}{8} \times \frac{5}{12} \times \frac{1}{12} \times \frac{3}{12} = 0.001205$$

Path 2:

$$\langle S \rangle \rightarrow \text{PER} \rightarrow \text{O} \rightarrow \text{LOC} \rightarrow \text{O} \rightarrow \langle E \rangle = \frac{3}{12} \times \frac{3}{10} \times \frac{5}{12} \times \frac{2}{8} \times \frac{4}{8} \times \frac{5}{12} = 0.000521$$



✓ HMM found the correct tags! The highest probability path is correct.

Finding suitable Path in Viterbi Graphs

Algorithm Steps:

1) Initialization (j=1)

$$V[s][1] = \pi[s] \times b_s(w_1)$$

2) Recursion (j = 2 \dots n)

for each state s,

$$V[s][j] = \max_{s'} (V[s'][j-1] \times a_{s',s}) \times b_s(w_j)$$

3) Termination

$$P^* = \max_s V[s][n]$$

$$q_n^* = \arg \max_s V[s][n]$$

4) Backtracking

for j = n-1 to 1:

$$q_j^* = \text{Path}[q_{j+1}^*][j+1]$$

Complexity: $O(N \times |\text{Tags}|^2)$

Pseudocode for Viterbi Coding:

Algorithm Viterbi(Words, States, A, B, π):

Initialize:

for each state s:

$$V[1][s] = \pi[s] * B[s][\text{word}_1]$$

$$\text{backptr}[1][s] = 0$$

for i = 2 to N:

for each state s:

$$(\text{prob}, \text{prev_state}) = \max_{s'} \text{over } s' \text{ of } [V[i-1][s'] * A[s']$$
$$[s] * B[s][\text{word}_i]]$$

$$V[i][s] = \text{prob}$$

$$\text{backptr}[i][s] = \text{prev_state}$$

$$P^* = \max_{s} \text{over } s \text{ of } V[N][s]$$

return best path by backtracking

Summary

Current Trend in NLP:

- Early Rule-Based Systems relied on handcrafted linguistic patterns — effective but rigid and language-dependent.
- Statistical Models like Hidden Markov Models (HMMs) introduced probabilistic reasoning and sequence learning, marking the foundation of modern NER.
- HMMs modeled the relationship between observed words and hidden entity labels, capturing dependencies through transition and emission probabilities, decoded using Viterbi algorithm.

Limitations and Transition Forward:

- HMMs assume Markov independence and emission conditionality, which restrict their ability to capture long-range dependencies.
- They struggle with data sparsity, contextual ambiguity, and rich morphology of Indian languages.
- These challenges led to Conditional Random Fields (CRF), Neural architectures, and later Transformer-based models.

“HMM may be old, but it taught machines how to read, one token at a time.”



Thank you

