Autocorrect & Hidden Markov Models

POS Tagging & Sequence Labeling

Learning Objectives

- **Understand Autocorrect Systems**

  Learn how spelling correction works using edit distance and language models

- **Minimum Edit Distance Algorithm**

  Implement dynamic programming solution for string similarity measurement

- **Part-of-Speech Tagging**

  Understand grammatical categories and their importance in NLP applications

- **Hidden Markov Models**

  HMM-based POS tagger with transition and emission probabilities

- **Viterbi Algorithm**

  Find optimal tag sequences using dynamic programming decoding

Today's Agenda

Autocorrect Systems
Spelling correction fundamentals

Minimum Edit Distance
Levenshtein distance & DP

Part-of-Speech Tagging
Grammatical categories & tagsets

Hidden Markov Models
Probabilistic sequence modeling

Viterbi Algorithm
Optimal sequence decoding

Lab: Autocorrect System
Hands-on implementation

REVIEW

Quick Recap: Session 04

Machine Translation

Transformation matrices for word vectors

$X \cdot R \approx Y$ mapping between languages

Frobenius norm for loss optimization

K-Nearest Neighbors for translation

Document Search

TF-IDF vector representations

Cosine similarity for matching

Approximate nearest neighbors

Efficiency vs accuracy trade-offs

Locality Sensitive Hashing

Hash functions for similar vectors

Random hyperplanes partitioning

Multi-table approach for accuracy

$O(1)$ lookup for large datasets

PART 1

Autocorrect Systems

Spelling Error Types & Detection

Edit Distance Fundamentals

Minimum Edit Distance Algorithm

Candidate Generation & Scoring

PART 1: AUTOCORRECT

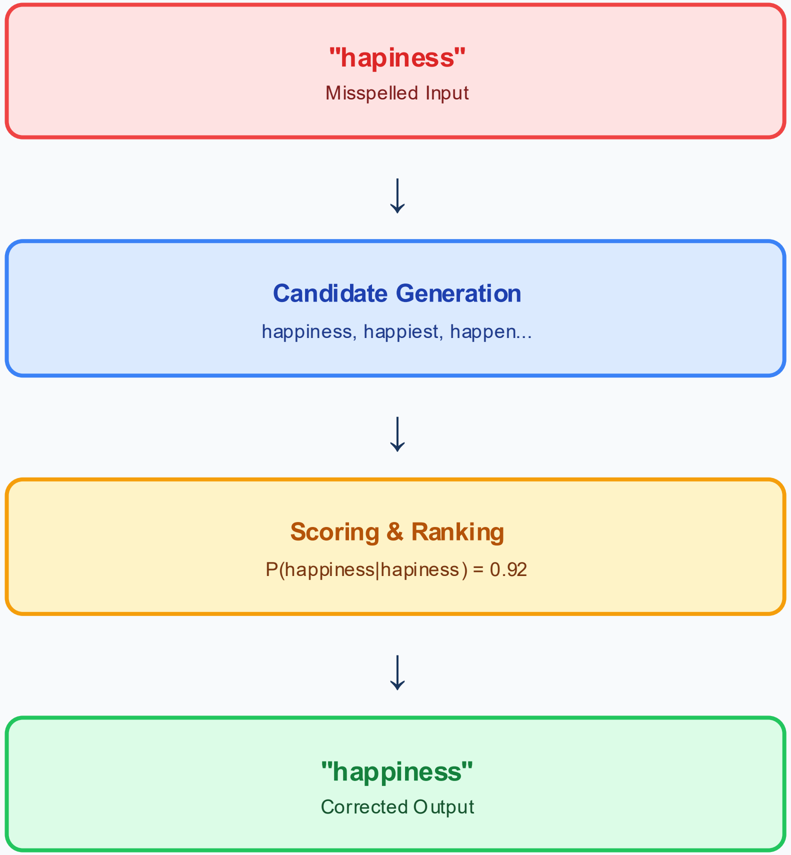
What is Autocorrect?

Definition

Autocorrect is a feature that automatically detects and corrects misspelled words, typos, and grammatical errors as you type.

Core Components

- **Error Detection:** Identify misspelled words
- **Candidate Generation:** Find possible corrections
- **Candidate Scoring:** Rank by probability
- **Correction:** Apply the best candidate



Types of Spelling Errors

Non-word Errors

Result in invalid words

Insertion: "thhe" → "the"

Deletion: "acros" → "across"

Substitution: "graffe" → "giraffe"

Transposition: "teh" → "the"

Detection: Easy - word not in dictionary

Real-word Errors

Valid but wrong words

Typos: "there" → "three"

Cognitive: "their" → "there"

Homophones: "piece" → "peace"

Grammar: "affect" → "effect"

Detection: Hard - requires context analysis

80%

of errors are within edit distance 1

25%

of spelling errors are real words

97%

covered by edit distance ≤ 2

Edit Distance (Levenshtein Distance)

Definition

The **minimum edit distance** between two strings is the minimum number of editing operations (insert, delete, substitute) needed to transform one string into another.

Edit Operations

I	Add a character: "at" → "cat"
D	Remove a character: "heat" → "hat"
S	Replace a character: "hat" → "hot"

Named after: Vladimir Levenshtein (1965)

Example: "intention" → "execution"

I N T E N T I O N

E X E C U T I O N

Edit Distance = 5

3 substitutions + 1 insertion + 1 deletion

Minimum Edit Distance Algorithm

Dynamic Programming Approach

Build a matrix D[i,j] where D[i,j] represents the minimum edit distance between the first i characters of source string and first j characters of target string.

Recurrence Relation

```
D[i,j] = min {
    D[i-1,j] + 1 // delete
    D[i,j-1] + 1 // insert
    D[i-1,j-1] + cost // sub
}
```

where cost = 0 if source[i] = target[j], else cost = 1 (or 2 for Levenshtein)

Base Cases:
D[i,0] = i (delete all i characters)
D[0,j] = j (insert all j characters)

Complexity Analysis

O(mn)	O(mn)
Time	Space

m = |source|, n = |target|

Distance Variants

Levenshtein: sub cost = 1

LCS Distance: sub cost = 2

Damerau: includes transposition

Weighted: keyboard-aware costs

Edit Distance Matrix Example

D matrix for "CAT" → "CUT"

		C	U	T
	0	1	2	3
C	1	0	1	2
A	2	1	1	2
T	3	2	2	1

Result: D[3,3] = 1 (one substitution: A → U)

Cell Calculation

For D[2,2] (comparing "CA" with "CU"):

```
min{D[1,2]+1, D[2,1]+1, D[1,1]+1}
= min{1+1, 1+1, 0+1} = 1
```

Color Legend

🟦	Base cases (initialization)
🟩	Match (cost = 0)
🟨	Substitution path
🟧	Final answer

Backtrace: Finding the Alignment

Backtrace: "INTENTION" → "EXECUTION"

	ε	E	X	E	C	U	T
ε	**0**	1	2	3	4	5	6
I	1	**1**	2	2	3	4	5
N	2	2	**2**	3	3	4	5
T	3	3	3	**3**	4	4	4
E	4	3	4	**3**	**4**	5	5
N	5	4	4	4	4	**5**	5

Backtrace path shown in green

Note: depend on selecting adjacent cells but total cost is 5

Backtrace Rules

↖ **Diagonal:** Match or Substitute

⟵ **Left:** Insert into target

↑ **Up:** Delete from source

Resulting Alignment

```
I N T E N T I O N
| | | | | | | | |
E X E C U T I O N
```

Operations:

5 substitutions

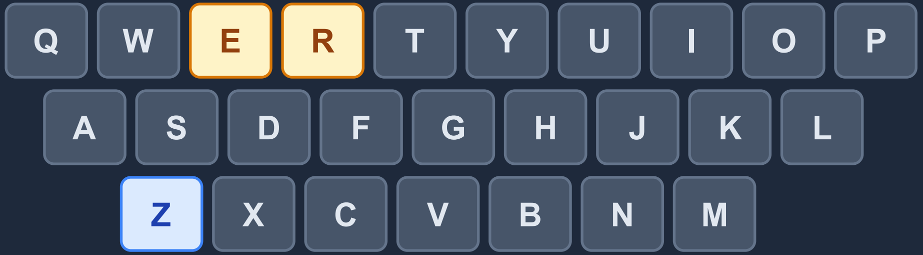
Total: **5**

Weighted Edit Distance

Keyboard-Aware Costs

Adjacent keys on keyboard are more likely to be confused than distant keys. Weighted edit distance assigns lower costs to more probable errors.

QWERTY Keyboard Layout



E↔R substitution: cost 0.5 (adjacent) | E↔Z substitution: cost 2.0 (distant)

Cost Matrix Examples

Operation	Pair	Cost
sub(adjacent)	e → r	0.5
sub(same row)	e → t	1.0
sub(distant)	e → z	2.0
delete	any	1.0
insert	any	1.0

Confusion Matrices

del[x,y]: P(xy typed as x)

ins[x,y]: P(x typed as xy)

sub[x,y]: P(x typed as y)

Candidate Generation

Edit Distance 1 Operations

For misspelled word "helo":

Deletion: elo, hlo, heo, hel

Insertion: ahelo, bhelo, ... zhelo, haelo, ...

Substitution: aelo, belo, ... zelo, halo, ...

Transposition: ehlo, hleo, heol

Number of Candidates

Operation	Formula	n=4
Deletion	n	**4**
Insertion	26 × (n+1)	**130**
Substitution	26 × n	**104**
Transposition	n − 1	**3**
Total	54n + 25	**~241**

Filtering Pipeline

"helo"
Misspelled word

Generate all edits
~250 candidates

Filter by vocabulary
Keep only valid words

hello, help, held, hero...
~5-10 valid candidates

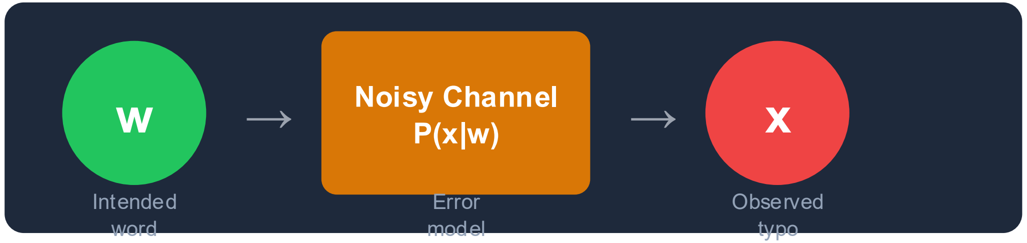
Edit Distance 2

Apply edits to edit-1 candidates: ~40K → ~50 valid

Noisy Channel Model

The Core Idea

The writer intended to type word w, but the "noisy channel" (keyboard + human errors) produced the observed misspelling x. Find the most likely original word.



Bayes' Rule for Spelling Correction

$$\hat{w} = \text{argmax } P(w|x) = \text{argmax } P(x|w) \cdot P(w)$$

P(x|w) - Channel Model
Probability of error x given w

P(w) - Language Model
Prior probability of word w

Example: "acress"

Candidate	P(x|w)	P(w)	Score
actress	0.037	0.0002	7.4e-6
cress	0.012	0.0001	1.2e-6
caress	0.008	0.00005	4.0e-7
access	0.017	0.0003	5.1e-6
across	0.019	0.0004	7.6e-6

Language Model P(w)

Unigram: word frequency
Bigram: $P(w|w_{-1})$
N-gram: context-aware

Channel Model P(x|w)

Based on confusion matrices from spelling error corpora

PART 2

Part-of-Speech Tagging

What is POS Tagging?

Tag Sets: Penn Treebank & Universal

Ambiguity & Challenges

Applications in NLP

What is Part-of-Speech Tagging?

Definition

Part-of-Speech (POS) tagging is the process of assigning a grammatical category (noun, verb, adjective, etc.) to each word in a sentence based on its definition and context.

Example Tagging

DET ADJ NOUN VERB ADP DET ADJ

NOUN

Also known as: POS tagging, grammatical tagging, word-class tagging

Why is POS Tagging Important?

- 📊 **Parsing:** Syntactic analysis
- 🔍 **NER:** Named entity recognition
- 🌐 **MT:** Machine translation
- 💬 **QA:** Question answering

Sequence Labeling Task

Input: sequence of words
Output: sequence of tags

One tag per word

Challenge: Same word can have different tags depending on context!
"I **book** a flight" (VERB)
"Read this **book**" (NOUN)

Penn Treebank Tag Set

Core Tags (45 total)

Tag	Description	Example
NN	Noun, singular	*cat, dog*
NNS	Noun, plural	*cats, dogs*
NNP	Proper noun, singular	*John, London*
NNPS	Proper noun, plural	*Americans*
VB	Verb, base form	*run, eat*
VBD	Verb, past tense	*ran, ate*
VBG	Verb, gerund	*running*
VBN	Verb, past participle	*eaten*
VBP	Verb, non-3rd sg pres	*run, eat*
VBZ	Verb, 3rd sg present	*runs, eats*
JJ	Adjective	*big, fast*
JJR	Adjective, comparative	*bigger*
JJS	Adjective, superlative	*biggest*
RB	Adverb	*quickly*
RBR	Adverb, comparative	*faster*
RBS	Adverb, superlative	*fastest*

Function Words & Others

Tag	Description	Example
DT	Determiner	*the, a, an*
PRP	Personal pronoun	*I, you, he*
PRP$	Possessive pronoun	*my, your*
WDT	Wh-determiner	*which, that*
WP	Wh-pronoun	*who, what*
WP$	Possessive wh-	*whose*
IN	Preposition/subord conj	*in, of, that*
CC	Coordinating conj	*and, but, or*
TO	"to"	*to*
MD	Modal	*can, will*
CD	Cardinal number	*one, 2, 100*
EX	Existential there	*there is*
FW	Foreign word	*bonjour*
UH	Interjection	*oh, wow*
.	Sentence-final punct	*. ! ?*

Note: Penn Treebank is the most widely used tag set for English NLP

Universal Dependencies Tag Set

Why Universal Tags?

Penn Treebank has **45 tags** – very details but only apply to Eng.
Universal Dependencies (UD) provide tags.

17 Universal POS Tags

Tag	Description	Tag	Description
ADJ	Adjective	NOUN	Noun
ADP	Adposition	NUM	Numeral
ADV	Adverb	PART	Particle
AUX	Auxiliary	PRON	Pronoun
CCONJ	Coord. Conjunction	PROPN	Proper Noun
DET	Determiner	PUNCT	Punctuation
INTJ	Interjection	SCONJ	Subord. Conjunction
VERB	Verb	SYM	Symbol
		X	Other

So sánh Tag Sets

Penn Treebank

`NN, NNS, NNP, NNPS`

4 noun tags

Universal

`NOUN, PROPN`

2 noun tags

🌍 Cross-lingual Example

English: The cat sleeps `DET NOUN VERB`

Vietnamese: Con mèo ngủ `NOUN NOUN VERB`

UD Project: 200+ treebanks cho 100+ ngôn ngữ

https://universaldependencies.org

POS Ambiguity - Challenges

Ambiguity statistics:

40% word types have multi tags

55% word tokens ambiguous

Example

"book"

NOUN	*"I read a book"*
VERB	*"Please book a flight"*

"back"

NOUN	*"My back hurts"*	VERB	*"Back the car up"*
ADV	*"Go back home"*	ADJ	*"The back door"*

Disambiguation Approaches

1. Context-based
words before/after help to determine tags

2. Statistical
P(tag|word, previous_tags)

3. Neural Networks
Bi-LSTM + CRF models

Most Frequent Tag Baseline

If assign popular tag to each word:

Accuracy: **~90%**

~10% disambiguate!

Challenge: OOV - Out of Vocabulary account for 5-10% tokens in the new text

POS Tagging Approaches

Rule-based

Use dictionary + rule base provided by experts

Ví dụ quy tắc:

```
IF word ends "-ing"
 AND prev_tag = "be"
 THEN tag = VBG
```

✅ Explainable, precise

❌ Labor-intensive, inflexible

Statistical (HMM)

Learn from labeled data (supervised learning)

Hidden Markov Model:

$P(tag|prev\_tag) \times P(word|tag)$

✅ Data-driven, ~97% acc

❌ Needs labeled data

Neural Networks

Deep learning with word embeddings and sequence models

Modern approach:

BiLSTM + CRF

BERT + Fine-tuning

Transformer models

✅ State-of-the-art, ~98% acc

❌ Compute-intensive

03

Hidden Markov Models

HMM for Sequence Labeling

States & Transitions

Emission Probabilities

Decoding Algorithm

PART 3: HIDDEN MARKOV MODELS

What is a Hidden Markov Model?

Definition

Hidden Markov Model (HMM) is a probabilistic model in which the system is modeled as a Markov processing with hidden states.

Assumptions

1 Markov Assumption

Probability of current state depending on previous state:

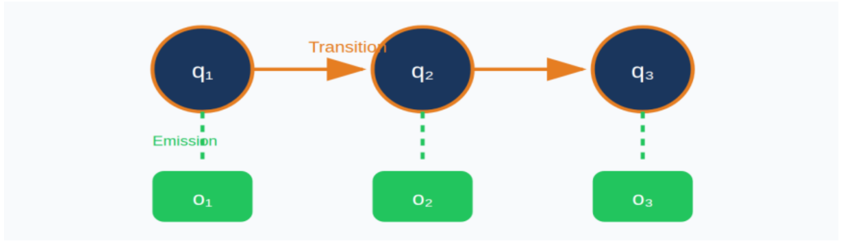
$$P(q_i \mid q_1 \ldots q_{i-1}) = P(q_i \mid q_{i-1})$$

2 Output Independence

Observation only depending on the its state:

$$P(o_i \mid q_1 \ldots q_n, o_1 \ldots o_n) = P(o_i \mid q_i)$$

HMM Structure



HMM	POS Tagging	Examples
Hidden States	POS Tags	NOUN, VERB, DET, ADJ...
Observations	Words in sentence	the, cat, runs, fast...
Transitions	Prob of tag → tag	P(NOUN|DET) = 0.8
Emissions	Prob of tag → word	P('cat'|NOUN) = 0.02

"Hidden" means we can not observe – only see outputs! We just see the words

HMM Components ($\lambda = A, B, \pi$)

Q - States

Collection of N states has:

$Q = \{q_1, q_2, \ldots, q_n\}$

POS: {NOUN, VERB, ADJ, DET, ...}

V - Observations

Vocabulary - symbol observations:

$V = \{v_1, v_2, \ldots, v_m\}$

POS: {the, cat, run, quickly, ...}

A - Transitions

Transition matrix of states:

$a_{ij} = P(q_j \mid q_i)$

Ví dụ: P(NOUN|DET) = 0.8

B - Emissions

Emission matrix of observation from states:

$b_i(v_k) = P(v_k \mid q_i)$

Ví dụ: P("cat"|NOUN) = 0.02

π - Initial Probs

Initial probability:

$\pi_i = P(q_1 = q_i)$

Ví dụ: P(start=DET) = 0.4

Summary

HMM is defined by:

$$\lambda = (A, B, \pi)$$

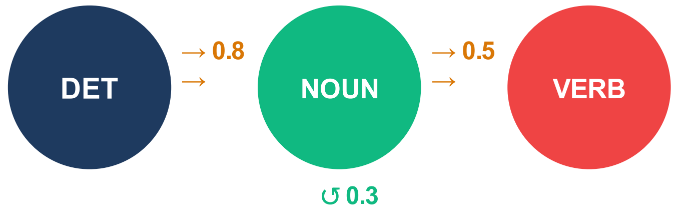
N×N + N×M + N parameters

Transition Probabilities (A Matrix)

Definition

$$a_{ij} = P(t_j \mid t_i) = C(t_i, t_j) / C(t_i)$$

State Transition Diagram



Transition Matrix Example

A	<s>	DET	NOUN	VERB	ADJ	</s>
<s>	0	0.40	0.25	0.20	0.10	0.05
DET	0	0.02	0.80	0.01	0.15	0.02
NOUN	0	0.05	0.10	0.50	0.05	0.30
VERB	0	0.25	0.35	0.10	0.15	0.15
ADJ	0	0.05	0.70	0.05	0.10	0.10

Constraints: sum of each row = 1: $\Sigma_j a_{ij} = 1$

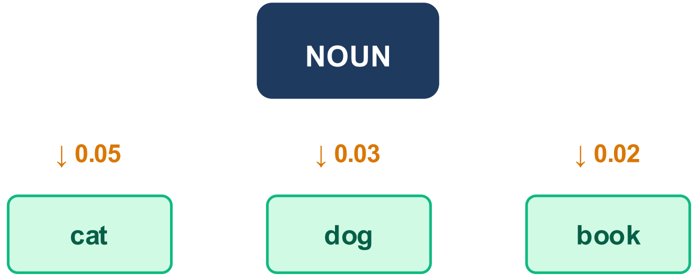
Insight: P(NOUN|DET) = 0.80 high because articles usually is come along with nouns ("the cat", "a book")

Emission Probabilities (Matrix B)

Definition

$$b_i(w) = P(w \mid t_i) = C(t_i, w) / C(t_i)$$

Emission Visualization



Emission Matrix Example

B	the	a	cat	dog	runs	big
DET	0.65	0.30	0	0	0	0
NOUN	0	0	0.05	0.03	0	0
VERB	0	0	0	0	0.08	0
ADJ	0	0	0	0	0	0.12

Sparsity Problem
$|V|$ = 50,000 words and $|T|$ = 45 tags
→ Matrix B has 2.25M entries, almost = 0. needs smoothing!

Constraints: with each tag t_i: $\Sigma w\, b_i(w) = 1$ (sum of probabilities emit from all words = 1)

Initial Probabilities (Vector π)

Definition

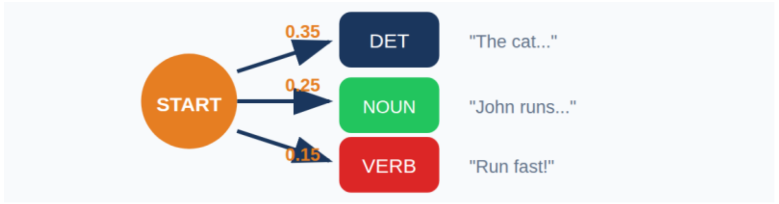
$$\pi = P(t = t) = C(t \text{ is a tag at beginning of sent}) / N$$

Example of Vector π

π(DET)	π(NOUN)	π(VERB)	π(PRON)	π(other)
0.35	0.25	0.15	0.20	0.05

Constraints: $\Sigma_i \pi_i = 1$

Start Distribution



HMM Complete Definition

λ = (A, B, π)

A: Transition matrix (N × N)
B: Emission matrix (N × |V|)
π: Initial distribution (N × 1)

N = no of tags, |V| = vocabulary size

Training HMM from Corpus

Supervised Training (with labeled corpus)

Tagged Corpus Example:

```
The/DET cat/NOUN sat/VERB on/ADP the/DET mat/NOUN ./PUNCT
A/DET dog/NOUN runs/VERB fast/ADV ./PUNCT
```

Count-Based Estimation

STEP 1: Transition Counts
$a_{ij} = C(t_i \rightarrow t_j) / C(t_i)$

STEP 2: Emission Counts
$b_i(w) = C(t_i, w) / C(t_i)$

STEP 3: Initial Counts
$\pi_i = C(t_i \text{ starts sentence}) / N$

Zero Probability Problem
if a bigram tag is not in training → P = 0 → all the sequence = 0!

Solution: Add-k Smoothing

$$a_{ij} = (C(t_i, t_j) + k) / (C(t_i) + k \cdot N)$$

k = 0.001 or 1 (Laplace smoothing)

Corpus: Penn Treebank, Brown, Universal Dependencies

Viterbi Algorithm

PART 4: VITERBI ALGORITHM

The Decoding Problem

Goal

$$\hat{t} = \text{argmax}_{t_1 \ldots t_n} P(t_1 \ldots t_n \mid w_1 \ldots w_n)$$

Find the best tags sequence for a given words

Example

Input:

"The bear can run"

Output:

DET NOUN VERB VERB

Brute Force Problem

with N tags and T words → N^T **possible sequences!**

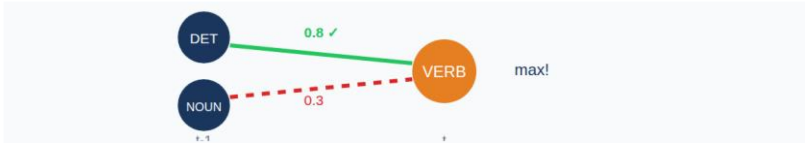
N=45, T=20 → $45^{20} \approx 10^{33}$ → impossible mission (:P)

Solution: Dynamic Programming

Viterbi algorithm usesDP to find optimal path with $O(N^2T)$ instead of $O(N^T)$

Insight

Optimal Substructure: Best path to state j at time t depends on best paths at time t-1



Complexity Comparison

Brute Force	Viterbi
$O(N^T)$	$O(N^2T)$

Viterbi Recurrence

STEP 1: Initialization (t = 1)

$$v_1(j) = \pi_j \cdot b_j(w_1)$$

Probability of state j × probability emiting at the begining

STEP 2: Recursion (t = 2...T)

[illegible]

v(i): best score to state i at time t-1
a: transition prob from i → j
b(w): emission prob of w from j

STEP 2b: Store Backpointer

$$bp_t(j) = \text{argmax}_i \, [v_{t-1}(i) \cdot a_{ij}]$$

Save the best state to j → used to trace back

STEP 3: Termination

best_score = max v(j)
best_last_tag = argmax v(j)

Numerical Underflow

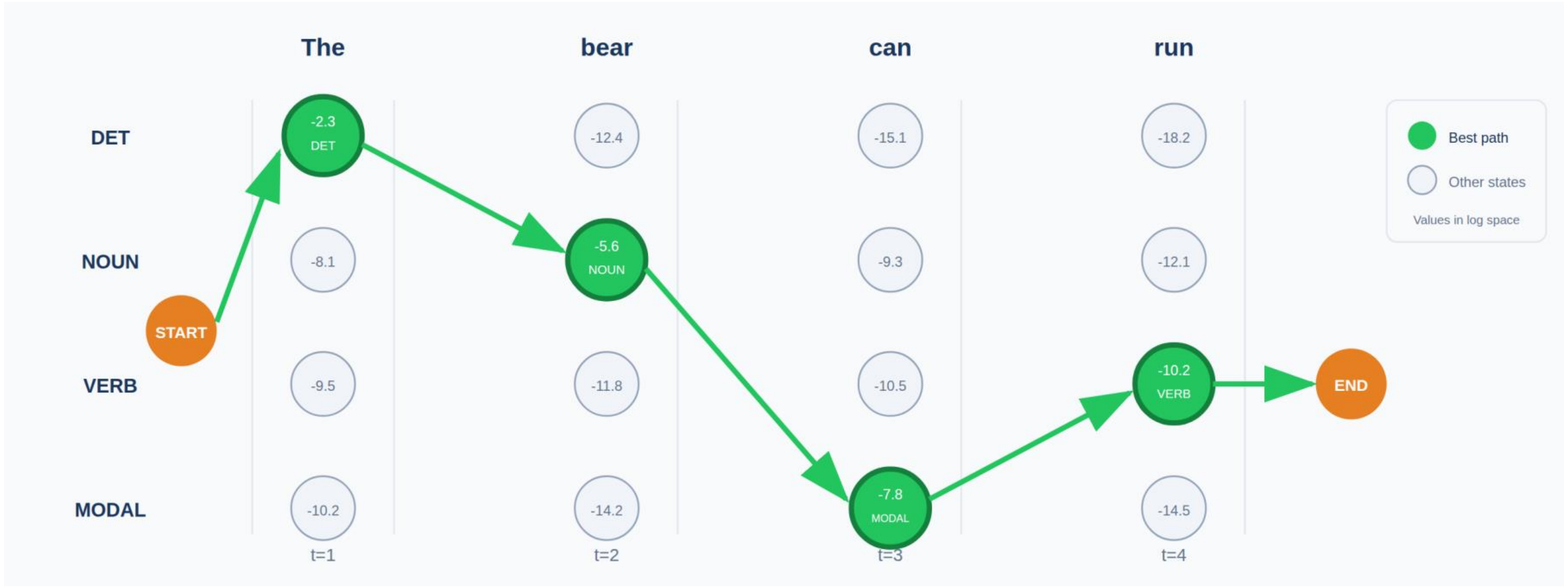
Product of small prob → underflow (≈ 0)

Solution: Log Space

log vt(j) = maxi[log vt-1(i) + log aij] + log bj(wt)

Viterbi Trellis Diagram

Input: "The bear can run" → **Output:** DET NOUN MODAL VERB



Example: Step by step

1 **Initialization: t=1, word="The"**

$$v_1(\text{DET}) = \pi(\text{DET}) \times b_{\text{DET}}(\text{"The"})$$

$$= 0.35 \times 0.50 = \mathbf{0.175}$$

DET: **0.175** ✓ NOUN: 0.001

2 **Recursion: t=2, word="bear"**

$$v_2(\text{NOUN}) = \max[v_1(i) \times a_{i,\text{NOUN}}] \times b_{\text{NOUN}}(\text{"bear"})$$

$$= 0.14 \times 0.05 = \mathbf{0.007}$$

$bp_2(\text{NOUN}) = \text{DET}$

NOUN: **0.007** ✓ DET: 0.0001

3 **Continue: t=3,4**

t=3: "can"
Best: MODAL, bp=NOUN

t=4: "run"
Best: VERB, bp=MODAL

Backpointer Table

t	1	2	3	4
Tag	DET	NOUN	MODAL	VERB
bp	START	DET	NOUN	MODAL

Backtrace: Reconstruct Path

VERB ← MODAL ← NOUN ← DET

Result: **The/DET bear/NOUN can/MODAL run/VERB**

Lab & Practice

Autocorrect System

HMM POS Tagger

Lab 1: Autocorrect System

Implementation Pipeline

Step 1: Build Vocabulary
Load corpus, count word frequencies

Step 2: Implement Edit Distance
DP algorithm with backtrace

Step 3: Generate Candidates
Edit dist 1 and 2, filter by vocab

Step 4: Rank Candidates
Noisy channel model scoring

Dataset: Shakespeare corpus hoặc Wikipedia text dump

✓ **Expected:** "speling" → "spelling", "correc" → "correct"

Lab 2: HMM POS Tagger

Implementation Steps

Step 1: Load & Preprocess Data
NLTK Brown corpus với POS tags

Step 2: Estimate Probabilities
Build A, B, π matrices từ counts

Step 3: Apply Smoothing
Add-k smoothing for unknown transitions

Step 4: Implement Viterbi
DP decoding with backpointers

Step 5: Evaluate
Accuracy trên test set

Summary

1. Autocorrect Systems

Minimum Edit Distance (Levenshtein), Dynamic Programming với backtrace, Noisy Channel Model

3. Hidden Markov Models

λ = (A, B, π) definition, Transition và Emission probabilities, Supervised training từ corpus

2. POS Tagging

Sequence labeling task, Penn Treebank (45 tags) và Universal (17 tags), Word ambiguity challenge

4. Viterbi Algorithm

DP decoding $O(N^2T)$, Log space computation, Backpointer for path reconstruction

Edit Distance

D[i,j] = min{D[i-1,j]+1, D[i,j-1]+1, D[i-1,j-1]+cost}

Viterbi Recursion

QPĂĆÅĚ Ö MŔÒLQPĆĈĂÒÅffĭ MÒĹ ffĭ
NŐĂR PÅ