Part-of-Speech Tagging

HMMs, Viterbi Decoding, RNNs, and BERT

Yasser Hifny

Faculty of Computers and Artificial Intelligence
Capital University

1. What Is Part-of-Speech Tagging?

POS tagging assigns a grammatical category to each token in a sentence.

$$(w_1, w_2, \ldots, w_T) \longrightarrow (y_1, y_2, \ldots, y_T)$$

Token	Illustrative tag	Meaning
`The`	DET	determiner
`students`	NOUN	noun
`read`	VERB	verb
`carefully`	ADV	adverb

- POS tags describe syntactic categories, not the full meaning of a word.
- A word can take different tags in different contexts.
- Modern Universal Dependencies uses the universal POS tag inventory (UPOS).

2. POS Tagging as Sequence Labeling

Given an observed token sequence $W = (w_1, \ldots, w_T)$, predict a tag sequence $Y = (y_1, \ldots, y_T)$.

$$\hat{Y} = \arg\max_{Y} P(Y \mid W)$$

Three approaches

- HMM: generative sequence model; explicit transition and emission probabilities.
- RNN: neural sequence model; recurrent hidden states encode left context.
- BERT: contextual Transformer encoder; token representations use left and right context.

The assignment compares an HMM baseline with a pretrained BERT tagger on Arabic-PUD.

3. Universal Dependencies and Arabic-PUD

The assignment treebank is Universal Dependencies Arabic-PUD:
`https://github.com/UniversalDependencies/UD_Arabic-PUD`

- UD treebanks use CoNLL-U, a tab-separated format with one token per row.
- The columns include token ID, form, lemma, UPOS, language-specific POS, features, and dependency annotations.
- For POS tagging, the target label is the UPOS column (column 4 in the 10-column CoNLL-U format).
- Sentences are separated by blank lines; comment lines begin with #.

Data handling

Use the repository's available train/dev/test files when provided. Keep the official test split untouched until final evaluation.

4. A Small Tagging Example

Consider the English illustration:

$$W = (\texttt{The}, \texttt{children}, \texttt{play})$$

A possible UPOS sequence is:

$$Y = (\text{DET}, \text{NOUN}, \text{VERB})$$

Position	1	2	3
Token	The	children	play
UPOS	DET	NOUN	VERB

The ambiguity is important: `play` can be a VERB in ”they play”? or a NOUN in ”the play”. The Context helps resolve the tag.

5. Hidden Markov Model: Assumptions

In an HMM tagger, the words w_t are observed and the tags y_t are hidden states.

- First-order Markov assumption: current tag depends on the previous tag.
- Emission assumption: current word depends on its current tag.

$$P(y_t \mid y_{1:t-1}) \approx P(y_t \mid y_{t-1})$$

$$P(w_t \mid w_{1:T}, y_{1:T}) \approx P(w_t \mid y_t)$$

Interpretation

The tag sequence models grammatical transitions; the emission model describes which words are likely under each tag.

6. Factorized Joint Probability

For words $W = (w_1, \ldots, w_T)$ and tags $Y = (y_1, \ldots, y_T)$:

$$\boxed{P(W, Y) = P(y_1) \prod_{t=2}^{T} P(y_t \mid y_{t-1}) \prod_{t=1}^{T} P(w_t \mid y_t)}$$

Using a start state $y_0 = \langle s \rangle$, this can be written:

$$P(W, Y) = \prod_{t=1}^{T} P(y_t \mid y_{t-1}) P(w_t \mid y_t).$$

For a sentence model, an end transition $P(\langle /s \rangle \mid y_T)$ may also be included.

Decoding

The best tag sequence is $\hat{Y} = \arg\max_Y P(W, Y)$, since W is fixed.

7. Transition Probabilities

A transition probability estimates how likely one tag follows another:

$$\hat{P}(y_j \mid y_i) = \frac{C(y_i, y_j)}{C(y_i)}$$

where $C(y_i, y_j)$ counts adjacent tag pairs and $C(y_i)$ counts outgoing transitions from y_i.

- Add a sentence-start tag $\langle s \rangle$ before the first tag.
- Count tag pairs in the training set only.
- Estimate end-of-sentence transitions if sentence endings are modeled.

Transitions encode syntactic regularities, for example DET $\rightarrow$ NOUN is often plausible.

8. Transition Scores: Numerical Example

Suppose the training corpus has 100 outgoing transitions from DET, including 70 transitions to NOUN, 20 to VERB, and 10 to DET.

$$P(\text{NOUN} \mid \text{DET}) = \frac{70}{100} = 0.70$$

$$P(\text{VERB} \mid \text{DET}) = \frac{20}{100} = 0.20, \quad P(\text{DET} \mid \text{DET}) = \frac{10}{100} = 0.10$$

The transition log-score for DET $\rightarrow$ NOUN is:

$$\log P(\text{NOUN} \mid \text{DET}) = \log(0.70) \approx -0.357$$

Important

These counts are illustrative, not measured statistics from Arabic-PUD.

9. Emission Probabilities

An emission probability estimates the likelihood of a word given its tag:

$$\boxed{\hat{P}(w \mid y) = \frac{C(y, w)}{C(y)}}$$

Here $C(y, w)$ is the number of times word w is labeled with tag y, and $C(y)$ is the total number of training tokens with tag y.

- Frequent word-tag associations receive higher probability.
- Surface forms may be sparse, especially for rare words and morphologically rich languages.
- Unknown-word handling is necessary for words absent from training.

10. Emission Scores: Numerical Example

Suppose NOUN appears 200 times in training, and the word `students` is tagged NOUN 12 times:

$$P(\texttt{students} \mid \text{NOUN}) = \frac{12}{200} = 0.06$$

Suppose VERB appears 150 times and `students` is never labeled VERB:

$$P(\texttt{students} \mid \text{VERB}) = \frac{0}{150} = 0$$

Zero counts

An unseen word-tag pair can make the probability of a complete path zero. Smoothing and unknown-word strategies help.

Again, counts are for demonstration only.

11. Add-One Smoothing for HMMs

For tag inventory $\mathcal{Y}$, add-one transition smoothing:

$$P_\alpha(y' \mid y) = \frac{C(y, y') + \alpha}{C(y) + \alpha|\mathcal{Y}|}$$

For word vocabulary V, add-one emission smoothing:

$$P_\alpha(w \mid y) = \frac{C(y, w) + \alpha}{C(y) + \alpha|V|}$$

Set $\alpha = 1$ for Laplace smoothing; smaller α may work better in practice.

- Reserve an `<UNK>` token or map rare training words to it.
- Apply the same rare-word rule to validation and test data.
- More advanced alternatives include hapax-based unknown-word models and interpolation.

12. HMM Decoding Objective

The HMM tagger seeks:

$$\hat{Y} = \arg\max_{Y} P(W, Y)$$

Substituting the factorization:

$$\hat{Y} = \arg\max_{Y} \left[P(y_1) \prod_{t=2}^{T} P(y_t \mid y_{t-1}) \prod_{t=1}^{T} P(w_t \mid y_t) \right].$$

Directly enumerating all tag sequences requires $|\mathcal{Y}|^T$ possibilities.

Dynamic programming

Viterbi decoding reuses the best partial path ending in each tag, reducing the complexity to $O(T|\mathcal{Y}|^2)$.

13. Viterbi: Log-Space Recurrence

Define $\delta_t(j)$ as the best log-probability of any path for words $w_{1:t}$ that ends in tag j.

$$\delta_1(j) = \log P(j \mid \langle s \rangle) + \log P(w_1 \mid j)$$

For $t \geq 2$:

$$\boxed{\delta_t(j) = \log P(w_t \mid j) + \max_i \left[\delta_{t-1}(i) + \log P(j \mid i)\right]}$$

Store the maximizing previous tag:

$$\boxed{\psi_t(j) = \arg\max_i \left[\delta_{t-1}(i) + \log P(j \mid i)\right]}$$

Use log probabilities to avoid numerical underflow from multiplying many small values.

14. Viterbi Algorithm: Initialization and Recursion

Step 1: Initialization For every possible tag j, combine the start transition and first-word emission:

$$\delta_1(j) = \log P(j \mid \langle s \rangle) + \log P(w_1 \mid j).$$

Step 2: Recursion For each later position t and candidate current tag j:

$$\delta_t(j) = \log P(w_t \mid j) + \max_i [\delta_{t-1}(i) + \log P(j \mid i)].$$

Record the maximizing predecessor $\psi_t(j)$ in a backpointer table.

At each cell

Compare every previous tag, add its best prefix score and transition score, then add the current word's emission score.

15. Viterbi Algorithm: Termination and Backtracking

Step 3: Termination If an end state is modeled:

$$y_T^* = \arg\max_j [\delta_T(j) + \log P(\langle /s \rangle \mid j)].$$

Otherwise choose $y_T^* = \arg\max_j \delta_T(j)$. **Step 4: Backtracking** Recover earlier tags using the stored pointers:

$$y_t^* = \psi_{t+1}(y_{t+1}^*), \quad t = T-1, \ldots, 1.$$

1. Initialize one score per tag.
2. Fill the dynamic-programming score table left to right.
3. Store the best predecessor at each cell.
4. Select the best final tag and follow backpointers to the beginning.

Numerical Example: Corpus

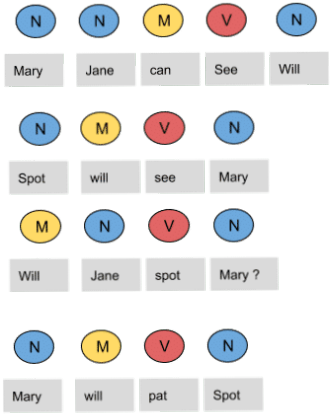


Figure: Source: `https://www.mygreatlearning.com/blog/pos-tagging/`

Numerical Example: Setup

Small corpus using tags NOUN, MODAL, and VERB, with words “Mary”, “Jane”, “Will”, “Spot”, “Can”, “See”, and “pat”.

Table: Word–POS tag frequency counts.

Word	NOUN	MODAL	VERB
Mary	4	0	0
Jane	2	0	0
Will	1	3	0
Spot	2	0	1
Can	0	1	0
See	0	0	2
pat	0	0	1

Emission Probabilities

Table: Emission probabilities $P(w \mid y)$.

Word	NOUN	MODAL	VERB
Mary	$\frac{4}{9}$	0	0
Jane	$\frac{2}{9}$	0	0
Will	$\frac{1}{9}$	$\frac{3}{4}$	0
Spot	$\frac{2}{9}$	0	$\frac{1}{4}$
Can	0	$\frac{1}{4}$	0
See	0	0	$\frac{2}{4}$
pat	0	0	$\frac{1}{4}$

$$P(\text{Will} \mid \text{MODAL}) = \frac{3}{4}, \qquad P(\text{Mary} \mid \text{NOUN}) = \frac{4}{9}.$$

Transition Counts

The special states $\langle S \rangle$ and $\langle E \rangle$ mark the beginning and end of a sentence.

Table: POS-tag transition frequency counts.

	NOUN	**MODAL**	**VERB**	$\langle E \rangle$
$\langle S \rangle$	3	1	0	0
NOUN	1	3	1	4
MODAL	1	0	3	0
VERB	4	0	0	0

Transition Probabilities

Table: POS-tag transition probabilities $P(y_i \mid y_{i-1})$.

	NOUN	**MODAL**	**VERB**	$\langle E \rangle$
$\langle S \rangle$	$\frac{3}{4}$	$\frac{1}{4}$	0	0
NOUN	$\frac{1}{9}$	$\frac{3}{9}$	$\frac{1}{9}$	$\frac{4}{9}$
MODAL	$\frac{1}{4}$	0	$\frac{3}{4}$	0
VERB	1	0	0	0

$$P(\text{MODAL} \mid \text{NOUN}) = \frac{3}{9}, \qquad P(\text{VERB} \mid \text{MODAL}) = \frac{3}{4}.$$

Candidate Sequence 1

Sentence: "Will can spot Mary."

$$\text{Will/MODAL} \quad \text{can/VERB} \quad \text{spot/NOUN} \quad \text{Mary/NOUN}$$

$$\begin{aligned} P(\mathbf{W}, \mathbf{Y}) = {} & P(\text{MODAL} \mid \langle S \rangle)\, P(\text{Will} \mid \text{MODAL}) \\ & \times P(\text{VERB} \mid \text{MODAL})\, P(\text{can} \mid \text{VERB}) \\ & \times P(\text{NOUN} \mid \text{VERB})\, P(\text{spot} \mid \text{NOUN}) \\ & \times P(\text{NOUN} \mid \text{NOUN})\, P(\text{Mary} \mid \text{NOUN}) \\ & \times P(\langle E \rangle \mid \text{NOUN}). \end{aligned}$$

Some required transitions or emissions have zero probability, so this sequence can receive a probability of zero.

Candidate Sequence 2

Will/NOUN can/MODAL spot/VERB Mary/NOUN

$$P(\mathbf{W}, \mathbf{Y}) = \frac{3}{4} \cdot \frac{1}{9} \cdot \frac{3}{9} \cdot \frac{1}{4} \times \frac{3}{4} \cdot \frac{1}{4} \cdot 1 \cdot \frac{4}{9} \cdot \frac{4}{9}$$

Therefore,

$$P(\mathbf{W}, \mathbf{Y}) \approx 0.0002572.$$

The HMM compares the probabilities of the candidate sequences and selects the one with the largest probability.

Search Space

Suppose there are K possible POS tags and the sentence has T words. If every word can take any tag, the number of tag sequences is

$$K^T.$$

Example: three tags and four words give

$$3^4 = 81.$$

The number of sequences grows **exponentially** with sentence length and tag-set size, so exhaustive evaluation is computationally inefficient.

The Viterbi Algorithm

- Uses **dynamic programming**.
- Instead of evaluating all tag sequences, it stores the probability of the best partial sequence ending in each possible POS tag.
- The best complete sequence is recovered by **backtracking** through the stored probabilities.

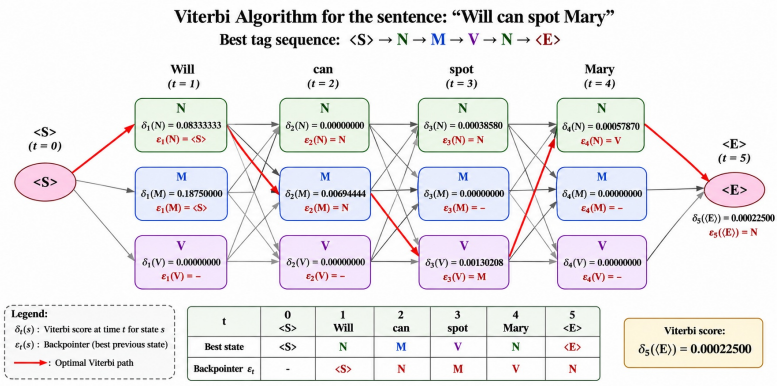
It computes

$$\hat{\mathbf{Y}} = \arg\max_{\mathbf{Y}} P(\mathbf{W}, \mathbf{Y})$$

without explicitly enumerating all K^T possible tag sequences.

Viterbi Example

Using the transition and emission probabilities from the previous example, the figure shows how Viterbi determines the optimal tag sequence for “Will can spot Mary.”



t	0 <S>	1 Will	2 can	3 spot	4 Mary	5 <E>
Best state	<S>	N	M	V	N	<E>
Backpointer ε_t	-	<S>	N	M	V	N

Figure: The Viterbi algorithm determines the optimal tag sequence for the sentence “Will can spot Mary” (AI generated).

20. Viterbi Pseudocode and Complexity

Algorithm

```
initialize delta[1, tag] and psi[1, tag]
for t = 2,...,T:
  for each current tag j:
    score(i) = delta[t-1,i] + log trans(i,j)
    psi[t,j] = argmax(i) score(i)
    delta[t,j] = max(i) score(i) + log emit(j,w[t])
select best final tag; backtrack through psi
```

$$\text{Time} = O(TK^2), \qquad \text{Space} = O(TK)$$

where T is the sentence length and K is the number of tags. Backpointer memory can be reduced if only the final score is required.

21. RNN POS Tagging: Motivation

An RNN reads token representations in sequence and builds a contextual hidden state.

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$$

For POS tagging, predict a tag at every token position rather than one label for the whole sentence.

- Each state summarizes the current input and preceding context.
- A classification head maps each state to a distribution over POS tags.
- A bidirectional RNN can incorporate both left and right context.

22. RNN Equations for Sequence Labeling

Embedding of token w_t: $\mathbf{x}_t = E[w_t]$.

$$\boxed{\mathbf{h}_t = \tanh(W_{xh}\mathbf{x}_t + W_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)}$$

Tag logits and probabilities:

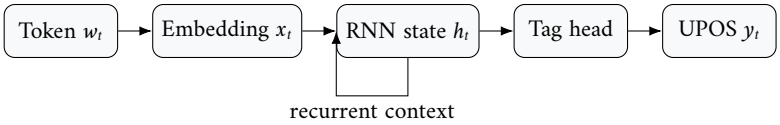
$$\mathbf{z}_t = W_{hy}\mathbf{h}_t + \mathbf{b}_y, \qquad P(y_t \mid W) = \text{softmax}(\mathbf{z}_t).$$

Token-level cross-entropy:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(y_t^{gold} \mid W).$$

For a bidirectional RNN, concatenate $\overrightarrow{\mathbf{h}_t}$ and $\overleftarrow{\mathbf{h}_t}$ before the classifier.

23. RNN POS Tagging Framework



- The head is applied at every non-padding token.
- Ignore padded positions in the loss.
- Evaluate using the original word-level gold tags.

24. RNN Training Details

- Pad sequences in each minibatch and preserve their true lengths or use masks.
- Align one gold POS tag with every input word.
- Optimize summed or mean cross-entropy over non-padding tokens.
- Use gradient clipping if training is unstable.
- Tune embedding size, hidden size, dropout, and bidirectionality on the dev set.

$$\mathcal{L} = -\frac{1}{\sum_{b,t} m_{b,t}} \sum_{b,t} m_{b,t} \log P(y_{b,t}^{gold} \mid W_b)$$

where $m_{b,t} = 1$ for a real token and 0 for padding.

25. BERT for POS Tagging

BERT is a bidirectional Transformer encoder pretrained with masked-language modeling (and, for original BERT, next-sentence prediction).

- Self-attention creates context-sensitive representations for each input position.
- Unlike a sentence classifier, POS tagging uses the representation at each token.
- A linear layer maps each contextual representation to UPOS logits.

$$H = \mathrm{BERT}(w_1, \ldots, w_T), \qquad \mathbf{z}_t = W_c \mathbf{h}_t + \mathbf{b}_c$$

$$P(y_t \mid W) = \mathrm{softmax}(\mathbf{z}_t)$$

Fine-tune the encoder and tag head jointly, or freeze the encoder as a controlled baseline.

26. BERT Tokenization and Word-Tag Alignment

A subword tokenizer may split one orthographic word into multiple pieces:

$$\text{word} \rightarrow (\text{subword}_1, \text{subword}_2, \ldots)$$

But UD supplies one UPOS tag per *syntactic* word.

Option 1: First subword

1. Assign the segment's gold UPOS tag to its first subword.
2. Assign -100 to continuation subwords and special tokens.
3. Compute loss only on first-subword positions.
4. Predict a segment's tag from its first subword.

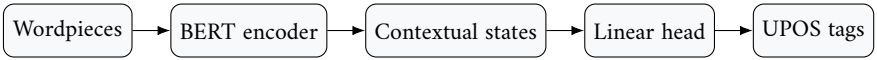
Option 2: Mean pooling

1. Average the hidden states of the segment's subwords: $\mathbf{h}_s = \frac{1}{|S_s|} \sum_{i \in S_s} \mathbf{h}_i$
2. Apply a linear classifier to $\mathbf{h}_s$.
3. Compute loss once per segment; use -100 only for padding.
4. Predict a segment's tag from $\mathbf{h}_s$.

Caveat: Arabic (and other clitic-rich languages)

An orthographic token may contain several UD syntactic words (multiword tokens), e.g. وبالكتاب → CCONJ + ADP + DET + NOUN. Segment clitics first, then align tags to the segments, or use composite tags.

27. BERT POS Tagging Architecture



For each supervised subword position t:

$$\mathcal{L} = -\sum_{t \in \mathcal{I}} \log P(y_t^{gold} \mid W)$$

where $\mathcal{I}$ contains only positions selected by the word-tag alignment policy.

28. Fine-Tuning and Evaluation Choices

Training

- Select an Arabic-capable pretrained checkpoint, such as an Arabic BERT model or multilingual BERT.
- Use the checkpoint's matching tokenizer.
- Tune learning rate, batch size, epochs, max length, and weight decay on the dev set.
- Use the same UPOS label inventory across HMM and BERT.

Evaluation

- Convert predictions back to one label per original word.
- Report token-level accuracy and macro-F1.
- Consider per-tag precision, recall, F1, and a confusion matrix.
- Do not count special tokens, padding, or ignored continuation subwords.

29. Assignment: Arabic-PUD POS Tagger

Objective

Implement an HMM POS tagger and a BERT-based POS tagger using the Universal Dependencies Arabic-PUD treebank, then compare their performance.

Part A: HMM

1. Parse CoNLL-U sentences and extract FORM and UPOS of each *segment* (syntactic word); use the segment rows, not the multiword-token range lines (e.g. `1-3`).
2. Estimate tag transition and word emission probabilities from training data.
3. Add smoothing and define a rare/unknown-word policy.
4. Implement Viterbi decoding over the segment sequence and evaluate on the official test split.

Part B: BERT

1. Use the segmented text: each segment is one input word. Tokenize segments into subwords and keep the subword-to-segment mapping (e.g. `word_ids()`).
2. Implement both alignment options from slide 33: *first-subword* labeling and *mean pooling*.
3. Fine-tune a pretrained BERT token-classification model with each option.
4. Decode one UPOS tag per *segment*, and evaluate with the same metrics and the same segmentation as the HMM. Ablation: compare first-subword and mean pooling on the same train/test splits.

30. Assignment: Experimental Protocol

1. Download the dataset from `https://github.com/UniversalDependencies/UD_Arabic-PUD`.
2. Use the provided train/dev/test splits when available; do not train on test sentences.
3. Map labels to the UPOS column, not the language-specific XPOS field.
4. Keep sentence boundaries intact for HMM transitions and sequence evaluation.
5. Use the dev set for hyperparameter selection and early stopping.
6. Use the same test sentences and UPOS labels for both models.
7. Fix random seeds and record package versions, checkpoint, and hyperparameters.

31. Assignment: Metrics and Results Table

Report at least:

- Token accuracy: correct word tags divided by evaluated words.
- Macro-F1: unweighted mean of F1 across UPOS classes.
- Per-tag precision, recall, F1, and confusion matrix.
- Training time and inference speed (optional but recommended).

Model	**Accuracy**	**Macro-F1**	**Notes**
HMM + smoothing	$\cdots$	$\cdots$	Viterbi
BERT token classifier	$\cdots$	$\cdots$	checkpoint: $\cdots$

Do not fill the table with expected values; report the scores produced by your experiments.

32. Assignment: Deliverables and Rubric

Deliverables

- Reproducible source code, environment details, and run instructions.
- HMM transition/emission estimation and Viterbi implementation.
- BERT fine-tuning script with explicit subword-to-word alignment.
- Test metrics, per-tag results, confusion matrix, and comparison table.
- Short error analysis with examples of ambiguous or difficult tokens.

Suggested rubric

- Data parsing and split integrity: 15%
- Correct HMM probabilities and Viterbi: 30%
- BERT fine-tuning and alignment: 30%
- Evaluation and fair comparison: 15%
- Reproducibility and error analysis: 10%

33. Summary

- HMM tagging maximizes a factorized joint probability of tags and words.
- Transition probabilities model tag sequences; emission probabilities model word-tag associations.
- Viterbi finds the most likely tag path with dynamic programming and backpointers.
- RNNs predict a tag from recurrent contextual states.
- BERT provides bidirectional contextual representations and predicts a tag at each supervised token.
- Arabic-PUD enables a controlled comparison using shared UPOS labels and test sentences.

Questions?

HMM + Viterbi $\longrightarrow$ RNN $\longrightarrow$ BERT