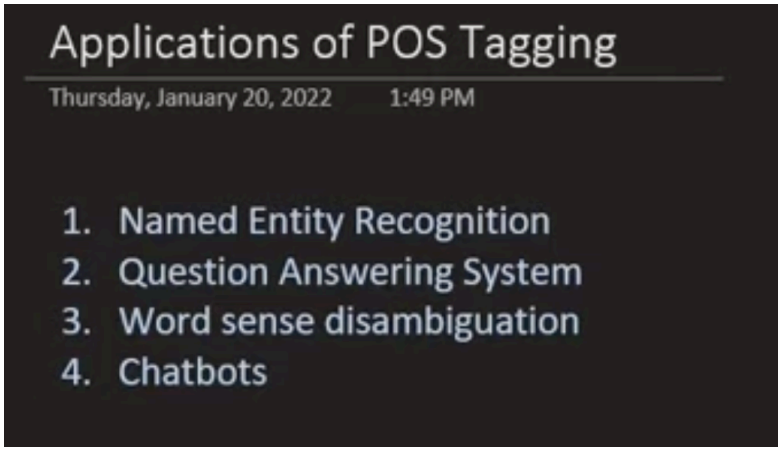
Lecture 18. NER with Conditional Random Fields (CRF)



`POS- preprocessing step`

`I left the room vs Left of the room (WSD)`

NER (Named Entity Recognition) is the task of identifying and classifying entities in text into predefined categories like:

- **PER** – Person
- **ORG** – Organization
- **LOC** – Location
- **MISC** – Miscellaneous

Given:
"Barack Obama was born in Hawaii."
NER should output:
`[B-PER, I-PER, O,  O,O, B-LOC, O]`

What Kind of Dataset Do You Need for NER?

You need a **token-level labeled dataset** where each token is assigned an **entity tag** using a scheme like:

- **BIO** (Begin, Inside, Outside)
- **BILOU** (Begin, Inside, Last, Outside, Unit)

🧾 Example (BIO Format):

Token	NER Tag
Barack	B-PER
Obama	I-PER
was	O
born	O
in	O
Hawaii	B-LOC
.	O

Popular NER Datasets

- **CoNLL-2003** — (English, German) – `PER`, `LOC`, `ORG`, `MISC`
- **OntoNotes 5.0** — broader set of entity types
- **WikiAnn** — multilingual NER
- **W-NUT 17** — emerging and rare entities

1. Rule-Based Systems (1990s)

Early NER systems relied entirely on **hand-crafted linguistic rules** and **gazetteers** (predefined lists of entity names). These systems detect entities by matching patterns and looking up tokens in dictionaries.

Key Components

1. **Gazetteers**
    - Lists of known entities (e.g., lists of person names, organization names, place names).
    - Often assembled from sources like Wikipedia, company registries, geographical databases.
2. **Pattern Rules**
    - **Regular expressions** that capture orthographic or syntactic patterns.
    - Examples:
        - Capitalization patterns: `([A-Z][a-z]+(\s[A-Z][a-z]+)*)` → sequences of capitalized words.
        - Suffix rules: words ending in “-Inc”, “-Ltd”, “–Corp” as organizations.
        - Context cues: if a token is preceded by “Dr.” or “Mr.”, tag it as a person.
3. **Contextual Heuristics**
    - Surrounding words or phrases indicating entity types:
        - “born in ___”: location after “in” likely a LOC.
        - “CEO of ___”: organization after “of” likely an ORG.

Example

"Dr. John Smith, the CEO of OpenAI Inc., was born in San Francisco."

1. Gazetteers (Entity Lists)

We assume the following from gazetteers:

- `John Smith` is a known **PERSON**
- `OpenAI Inc.` is a known **ORG**
- `San Francisco` is a known **LOC**

2. Pattern Rules (Handcrafted Rules)

Rule	Application
Titles like "Dr." or "Mr." → Person	"Dr." precedes "John Smith" → PERSON
Capitalized word sequences → Proper Nouns	"John Smith", "OpenAI Inc.", "San Francisco"
Words ending in Inc., Ltd → ORG	"OpenAI Inc." ends in "Inc."
"born in" → LOC likely follows	"born in San Francisco" → LOC

3. POS Tags Based on Rule Matching

Token	POS Tag	Rule/Reason
Dr.	NNP (title)	Title pattern
John	NNP	Capitalized, person name
Smith	NNP	Capitalized, person name
,	,	Punctuation
the	DT	Determiner
CEO	NNP	Title, capitalized
of	IN	Preposition

OpenAI	NNP	Capitalized, known org
Inc.	NNP	Ends in "Inc", org
,	,	Punctuation
was	VBD	Past tense verb
born	VBN	Passive participle
in	IN	Preposition
San	NNP	Capitalized, LOC
Francisco	NNP	Capitalized, LOC
.	.	Sentence end

Output

[Dr./NNP-PERSON] [John/NNP-PERSON] [Smith/NNP-PERSON], [the/DT] [CEO/NNP] [of/IN] [OpenAI/NNP-ORG] [Inc./NNP-ORG], [was/VBD] [born/VBN] [in/IN] [San/NNP-LOC] [Francisco/NNP-LOC].

Workflow

1. **Tokenization**: split text into tokens.
2. **Gazetteer Lookup**: if token (or sequence) appears in a gazetteer, assign its entity type.
3. **Apply Regex Rules**: for tokens not in gazetteers, match against pattern rules.
4. **Contextual Checks**: refine or disambiguate based on neighboring tokens.
5. **Conflict Resolution**: if multiple rules fire, use priority order (e.g., gazetteer over regex).

Strengths & Limitations

Strengths	Limitations
✓ Transparent and interpretable rules	✖ Labor-intensive to write and maintain
✓ Very precise for covered cases	✖ Poor recall on unseen entities
✓ Doesn't require annotated data	✖ Hard to generalize, brittle across domains/languages

Drawback

"Apple is opening a new office in Paris."

Ambiguity in Entity Recognition:

- "Apple" can refer to:
    - A fruit (common noun)
    - A company (organization)
    - A person's name (very rare, but possible in some gazetteers)

Because rule-based systems don't understand context, they rely on surface clues like capitalization or presence in static lists (gazetteers).

Issues with this approach

- Context-blind: Rules don’t capture semantic context (e.g., "Apple" + "opening an office" → org).
- Static gazetteers: Outdated or incomplete lists may lead to false negatives or positives.
- Hard to scale: Adding rules for every possible language nuance becomes unmanageable.
- Poor generalization: Unseen entities or new terms won't be recognized correctly.

2. Feature-Based Machine Learning Models

Before deep learning, NER was formulated as a **sequence labeling** problem and solved using models that rely on **hand-engineered features** to represent each token.

These are **supervised learning models** that require:

- **Labeled training data** (e.g., sentences with word-level NER tags).
- **Manually engineered features** that help the model learn patterns.

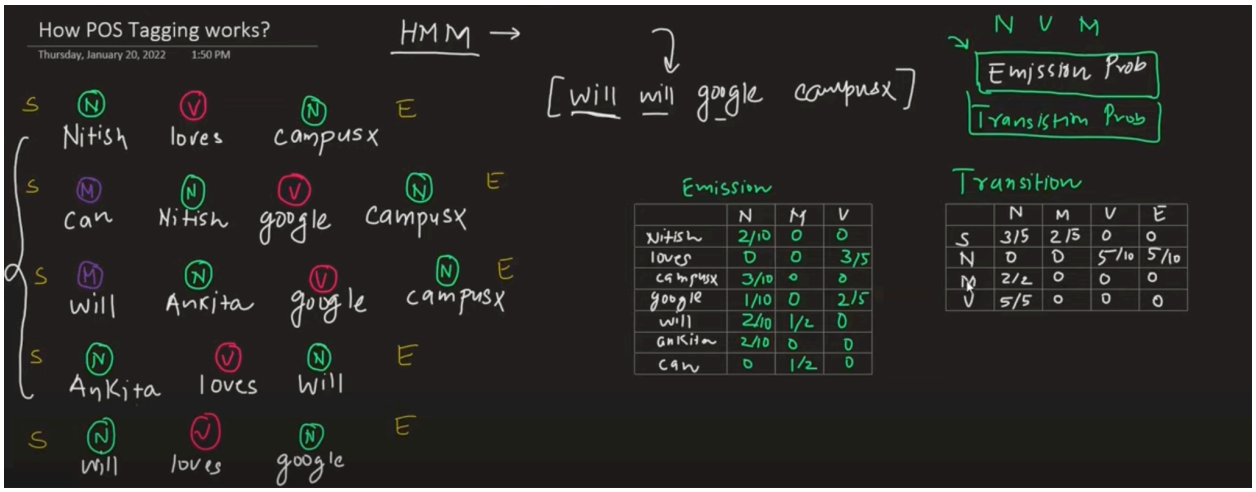
Typical algorithms used:

- Conditional Random Fields (CRF)
- Hidden Markov Models (HMMs)
- Support Vector Machines (SVM)
- Maximum Entropy Models

2.1 Hidden Markov Models (HMM)

Model Formulation

Recap: https://www.youtube.com/watch?v=269IGagoJfs



The model learns:

1. **Transition probabilities**:
   P(tag_t | tag_t-1) – likelihood of a tag following another (e.g., P(NNP | DT))

2. **Emission probabilities**:
   P(word_t | tag_t) – likelihood of a word given a tag (e.g., P(Angela | NNP))

Model Formulation

An HMM defines a joint probability over tag sequence $y_{1:n}$ and word sequence $x_{1:n}$:

$$P(x_{1:n}, y_{1:n}) = P(y_1) \prod_{t=2}^{n} P(y_t \mid y_{t-1}) \times \prod_{t=1}^{n} P(x_t \mid y_t)$$

- **Transition probability** $P(y_t \mid y_{t-1})$: probability of moving from tag y_{t-1} to y_t.
- **Emission probability** $P(x_t \mid y_t)$: probability of observing word x_t given tag y_t.

Decoding

- Use the **Viterbi algorithm** to find the most probable tag sequence:

$$\hat{y}_{1:n} = \arg\max_{y} P(x_{1:n}, y_{1:n})$$

2.2 Conditional Random Fields (CRF)

Step	Description
1. Define generic features	Create feature functions that apply to any text
2. Collect training data	Have sentences labeled with correct tags
3. Train CRF	Automatically learn weights for each feature-label pair
4. Predict on new text	Use learned weights and features to score labels

Understand the Example first:

Sample Dataset-

Step 1: Sample sentences

1. **Apple is launching a new product.**
2. **I ate a fresh apple this morning.**
3. **The apple on the tree is ripe.**
4. **Apple's stock price rose today.**

Step1: Use HMM for POS Tagging

Sentence Example 1:

"Apple is looking to hire a new engineer."

We want to tag:

- Apple as `NNP` (Proper Noun), and possibly as `ORG` (Named Entity: Organization)
- engineer as `NN` (Common Noun, Person)

POS Ambiguity: "Apple"

Problem with HMM:

- "Apple" is often seen as a fruit (NN) in training data.
- HMM relies on emission probabilities:
  If `P(Apple | NN)` > `P(Apple | NNP)`, it will tag Apple as a common noun (fruit), even in this context.

Why is this wrong?

- In context, "Apple is looking to hire..." clearly indicates it's a company, not a fruit.
- But HMM doesn't consider other words in the sentence like "hire", "engineer", "is looking" — just transitions and emissions.

How CRF gets it right:

- CRF can use features like:
  - Word capitalization: Apple → capitalized → likely proper noun.
  - Contextual cues: "Apple is looking" → verb after → likely subject is an organization/person.
  - Verb "hire" → implies subject is an employer → boosts `ORG` tag.
- CRF tags Apple correctly as `NNP` and possibly `ORG`.

CRF for NER

Example

| Sentence 1: "Barack Obama visited NASA headquarters in New York"

Labeled as:

```
css

Barack → B-PERSON
Obama → I-PERSON
visited → O
NASA → B-ORG
headquarters → O
in → O
New → B-LOC
York → I-LOC
```

Step 1: Generic feature functions

Here are 8 commonly used and **generic** feature functions that CRFs perform well with:

ID	Feature Description
f_1	Current word itself (unigram)
f_2	Is the word capitalized?
f_3	Is the word all uppercase?
f_4	POS tag (e.g., NNP, VB, NN)
f_5	Prefix in {pre, pro, un, re, dis}
f_6	Suffix in {ing, ion, ers, ist, ris}
f_7	Word in PERSON / ORG / LOC gazetteer
f_8	Previous word & previous label (transition)

Define Gazetteers

- **PERSON gazetteer:** [“John”, “Alice”, “Barack”, “Obama”, “Tim”, “Cook”]
- **LOCATION gazetteer:** [“New York City”, “London”, “Paris”, “California”, “Tokyo”]

Step 3: Prepare a feature table

Word	f_1	f_2	f_3	f_4 (POS)	f_5	f_6	f_7 (PERSON/ORG/LOC)	f_8: Previous Word & L
Barack	Barack	1	0	NNP=1	0	0	PERSON=1	None (START)
Obama	Obama	1	0	NNP=1	0	0	PERSON=1	Barack-PERSON
visited	visited	0	0	VB=1	0	0	NONE	Obama-PERSON
NASA	NASA	1	1	NNP=1	0	0	ORG=1	visited-NONE
headquarters	headquarters	0	0	NN=1	0	0	NONE	NASA-ORG
in	in	0	0	IN=1	0	0	NONE	headquarters-NONE
New	New	1	0	NNP=1	0	0	LOC=1	in-NONE
York	York	1	0	NNP=1	0	0	LOC=1	New-LOC

For each word in each sentence, the CRF creates a **feature vector** based on these features — basically a vector of 0s and 1s or values representing whether each feature is active and its corresponding label for training.

Step 4: Model Training: Learning Weights for Features

- **Goal:** The CRF wants to learn weights w_i for each feature f_i such that the **total score of the correct label sequence** is higher than incorrect sequences.
- The total score for a sequence $y = (y_1, y_2, ..., y_n)$ given input words $x = (x_1, x_2, ..., x_n)$ is:

$$Score(x, y) = \sum_{t=1}^{n} \sum_{i=1}^{8} w_i \cdot f_i(x, y, t)$$

- Here $f_i(x, y, t)$ is 1 if feature i is active at position t given label y_t (and possibly previous labels for transition features), else 0.
- Using **training data**, the model adjusts weights w_i to maximize the conditional likelihood of the correct labels.

So, training learns weights for:

- `f1 AND PERSON`
- `f1 AND ORG`
- `f1 AND LOC`
- `f1 AND O`

And the same for:

- `f2 AND PERSON`, ...
- `f3 AND ORG`, etc.

- CRF uses **gradient-based optimization** (e.g., L-BFGS, SGD) to **adjust weights w_i**
- Weights are updated to increase the probability of the **correct label sequence**
- Features that consistently **help predict correct labels** get **positive weights**.
- Features that are **not helpful** or **confuse the model** get **low or negative weights**.
- CRF **learns weights for *feature-label combinations*** — not just features alone

Step 5: Using weights for prediction (decoding)

When given a **new sentence** without labels, the CRF:

1. Extracts features for every word (f_1 to f_8).
2. Calculates the **score** for each possible label sequence using the weights.
3. Picks the label sequence with the **highest total score** (using Viterbi algorithm).

Assume these trained weights (just illustrative):

Feature ID	Weight for Label = PERSON	ORG	LOC	O
f_1	1.0	1.5	1.2	0.2
f_2	0.8	0.6	0.7	0.0
f_3	0.2	2.0	0.1	0.0
f_4	0.3	0.3	0.3	0.3
f_5	0.0	0.0	0.4	0.0
f_6	0.0	0.3	0.3	0.0
f_7	1.5	1.8	1.2	0.0
f_8	0.8	1.5	1.2	0.0

Prediction

Barack Obama visited NASA headquarters in New York"

We'll assume 8 features (as discussed earlier) and a trained CRF model with known weights.

“NASA”

f_1 = 1

f_2 = 1

f_3 = 1

f_4 = 1

f_5 = 0

f_6 = 0

f_7 = 1

f_8 = 1

- **PERSON:**
  - = 1×1.0 + 1×0.8 + 1×0.2 + 1×0.3 + 0 + 0 + 1×1.5 + 1×0.8 = **4.6**
- **ORG:**
  - = 1×1.5 + 1×0.6 + 1×2.0 + 1×0.3 + 0 + 0 + 1×1.8 + 1×1.5 = **7.7**
- **LOC:**
  - = 1×1.2 + 1×0.7 + 1×0.1 + 1×0.3 + 0 + 0 + 1×1.2 + 1×1.2 = **5.9**
- **O:**
  - = 1×0.2 + 1×0.0 + 1×0.0 + 1×0.3 + 0 + 0 + 0 + 0 = **0.5**

So, the predicted label for "NASA" is ORG, which matches the gold label.

CRF advantage over other approaches:

- Feature functions incorporate **surrounding words, part-of-speech tags, capitalization, dictionaries, and label dependencies**.
- **Flexibility:** CRFs learn from annotated data, adapting to various domains and linguistic patterns without manual rule engineering. Rules may miss exceptions or be too rigid.
- **Context Awareness:** CRFs model dependencies between neighboring labels and tokens, enabling recognition of multi-token entities (e.g., "New York City") and resolving ambiguities.

- **Handling Ambiguity:** CRFs use statistical evidence from multiple features, reducing errors when words have multiple possible tags (e.g., "Apple" as fruit or organization).
- **Generalization:** CRFs can generalize to unseen entities and variations, unlike fixed rules.
- **Example:** The phrase "United Nations" is recognized as an organization by CRFs due to context and label transitions; a rule-based system might misclassify or fail without explicit rules.

—----------END of TOPIC—----------

NER with BERT (Steps) (Optional- Not in syllabus)

Step	Description
1. Input tokens	e.g., "Barack Obama was born in Hawaii"
2. BERT Encoder	Outputs contextual embeddings for each token
3. Linear layer	Maps each embedding to label logits (raw scores)
👉 4. **Softmax / CRF**	Converts logits to tag probabilities or selects best tag sequence
5. Loss Function	Computes how wrong the predicted tags are (vs true tags)
6. Backpropagation	Updates weights to minimize loss

Why Contextual Embeddings?

Traditional models like Word2Vec or GloVe give **static embeddings**:

- "Apple" has **one vector**, regardless of whether it's a fruit or a company.

But **BERT provides contextual embeddings**:

- "Apple" in "I ate an apple" ≠ "Apple released a new iPhone"
- Meaning: BERT captures surrounding **context** dynamically.

How BERT Works for NER

BERT is a **Transformer-based model** that outputs a **vector representation for each token** in a sentence, capturing the full context (left and right) using **self-attention**.

For **each token** in the input sequence (like “Steve”, “Jobs”, “founded”),

BERT produces a **contextualized embedding**.

Linear Layer

Each token embedding is passed through a **linear layer**:

$$h_i = W \cdot e_i + b$$

Where:

- $W \in \mathbb{R}^{num\_labels \times hidden\_size}$
- $h_i \in \mathbb{R}^{num\_labels}$

in NER, we’re interested in **token-level** predictions — so we **ignore `[CLS]` and `[SEP]`** for tagging purposes.

NER with CRF (Conditional Random Field)

Component	Description
BERT Output	Contextual embeddings for each token
Linear Layer	Projects embeddings to logits (raw scores for each label)
Softmax	Converts logits to per-token probabilities (independent)
CRF	Chooses most likely valid tag sequence (with transitions)
Loss	Cross-entropy (Softmax) or Negative Log-Likelihood (CRF)
Backprop	Trains the model end-to-end

Instead of predicting tags independently, CRF considers transition scores between tags.

CRF Objective:

$$\hat{y}_{1:n} = \arg\max_{y \in \mathcal{Y}} \left(\sum_{i=1}^{n} \text{EmissionScore}(z_i, y_i) + \text{TransitionScore}(y_{i-1}, y_i) \right)$$

Where:

- **EmissionScore** comes from the linear layer (logits)
- **TransitionScore** is learned — e.g., the model learns that `B-PER → I-PER` is likely, but `B-LOC → I-PER` is not

CRF gives a globally optimal tag sequence using the **Viterbi algorithm**.

Example

🧾 Example Setup

Sentence: `"John lives in Paris"`
True Tags: `["B-PER", "O", "O", "B-LOC"]`
Tag Set: `["B-PER", "I-PER", "O", "B-LOC"]` → 4 labels

We'll assume:

- The sentence has 4 tokens (no subwords, to keep it simple)
- BERT has already provided contextual embeddings
- The **linear layer** has mapped these embeddings to **emission scores**
- The **CRF layer** has a **learned transition score matrix**

📌 Step 1: Emission Scores (From BERT + Linear)

Let's say these are the emission scores (logits) for each token:

Token	B-PER	I-PER	O	B-LOC
John	5.0	1.0	0.2	-1.0
lives	0.5	0.2	2.0	-1.5
in	-0.5	0.0	3.5	-1.0
Paris	-1.0	0.0	0.3	4.5

📌 Step 2: Transition Scores (Learned by CRF)

Assume the CRF has the following transition scores (from row to column):

From \ To	B-PER	I-PER	O	B-LOC
START	0.5	-1.0	0.0	0.0
B-PER	-1.0	2.0	0.5	0.3
I-PER	-1.0	1.5	0.2	-2.0
O	0.1	-1.5	1.0	0.6
B-LOC	-0.5	-1.0	1.2	0.9

We'll also assume:

- `START → B-PER` = `0.5`
- `B-PER → O` = `0.5`
- `O → O` = `1.0`
- `O → B-LOC` = `0.6`

📌 Step 3: Compute Total Score for Gold Sequence

Gold sequence: `["B-PER", "O", "O", "B-LOC"]`

We compute the score as:

mathematica Copy Edit

```
Score = Transition(START → B-PER)
      + Emission("John", B-PER)
      + Transition(B-PER → O)
      + Emission("lives", O)
      + Transition(O → O)
      + Emission("in", O)
      + Transition(O → B-LOC)
      + Emission("Paris", B-LOC)
```

Substituting values:

mathematica Copy Edit

```
= 0.5                  # START → B-PER
+ 5.0                  # Emission("John", B-PER)
+ 0.5                  # B-PER → O
+ 2.0                  # Emission("lives", O)
+ 1.0                  # O → O
+ 3.5                  # Emission("in", O)
+ 0.6                  # O → B-LOC
+ 4.5                  # Emission("Paris", B-LOC)
```

✅ Total score of gold path:

markdown Copy Edit

```
= 0.5 + 5.0 + 0.5 + 2.0 + 1.0 + 3.5 + 0.6 + 4.5 = **17.6**
```

📌 Step 4: Compute Partition Function (Z)

This is the total score of **all possible tag sequences**, computed efficiently via dynamic programming.

For simplicity here, let's assume we've computed `Z = 1000` (hypothetical for demonstration).

Then:

$$P(y_{\text{gold}} \mid x) = \frac{e^{17.6}}{Z} \quad \Rightarrow \quad \log P = 17.6 - \log 1000$$

$$\text{Loss} = -\log P = \log 1000 - 17.6 \approx 6.91 - 17.6 = -10.69$$

So:

$$\boxed{\text{Loss} = -(17.6 - \log Z)}$$

This is the **negative log-likelihood loss** that gets minimized during training.

📌 Step 5: During Inference

To predict the best tag sequence:

- CRF finds the path (label sequence) that gives **maximum total score** (emissions + transitions)
- This is done using the **Viterbi algorithm**

Concept	Description
Transition Table	A learned matrix that gives score of transitioning from tag A to tag B
Emission Score	Score of assigning a tag to a token (from BERT + linear)
Viterbi Algorithm	A decoding algorithm that uses emission + transition scores to find the most likely tag sequence
Purpose	Used **only at inference time** to decode the best tag sequence