Lecture 18. NER with Conditional Random Fields (CRF)

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.

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

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.

2.1 Hidden Markov Models (HMM)

Model Formulation

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)

CRFs address HMM limitations by modeling **conditional** probability of tag sequence given the observations and allowing **arbitrary features**:

a list of common CRF features in NER:

Feature	Description
`word.lower()`	Lowercased word
`word.istitle()`	Is it capitalized?
`word.isdigit()`	Is it a number?
`word[-3:]`	Last 3 letters (suffix)
`word[:3]`	First 3 letters (prefix)
`prev_word`, `next_word`	Surrounding context
`POS tag`	Part of speech
`is_in_gazetteer`	Is the word in a name/location list?

- We **predefine a small set** of suffixes that are **statistically informative** from training data.
  For example: `'ing'`, `'ion'`, `'man'`, `'son'`, `'ama'`, `'ski'`, etc.
- We only include a suffix feature if:
  - It occurs **frequently** in the training data, and
  - These are **statistically informative** suffixes for names, places, and common noun forms.

Sentence: John Goldman visited Moscow.

True Labels:

- John → B-PER
- Goldman → I-PER
- visited → O
- Moscow → B-LOC

Step 1: Feature Extraction

For each word, we extract features like:

- `word.lower`
- `word.istitle`
- `suffix3` (last 3 letters)
- `prev_word.lower`

Feature Representation for each word:

Features:

word.lower == 'john' → 1

word.istitle → 1

suffix3 == 'ohn' → 0 (not in suffix list)

prev_word.lower → <None>

Features:

word.lower == 'goldman' → 1

word.istitle → 1

suffix3 == 'man' → 1 ✅ (in suffix list)

prev_word.lower == 'john' → 1

Features:

word.lower == 'visited' → 1

word.istitle → 0

suffix3 == 'ted'   → 0 (not in suffix list)

prev_word.lower == 'goldman' → 1

Features:

word.lower == 'moscow' → 1

word.istitle   → 1

suffix3 == 'cow'   → 0 (not in suffix list)

prev_word.lower == 'visited' → 1

Step 2: Use Learned Weights for Label = I-PER

Let’s assume the CRF has learned the following feature weights (just for demo):

Let’s assume the CRF has learned the following feature weights (just for demo):

Feature	Weight (for I-PER)
word.lower == 'goldman'	+1.2
word.istitle	+1.5
suffix3 == 'man'	+2.5 ✅
prev_word.lower == 'john'	+1.0

➕ Total Emission Score for “Goldman” as I-PER:

Copy Edit

```
= 1×1.2 + 1×1.5 + 1×2.5 + 1×1.0 = 6.2
```

This emission score is passed to Viterbi decoding along with transition scores to compute the best sequence of labels.

- CRFs **don’t learn the suffixes themselves**; you **manually extract them** from training data.
- You define a set of suffixes like `['ing', 'ion', 'man', 'ski', 'ama']`
- Each becomes a **binary feature**
- CRF learns which suffixes **positively or negatively correlate** with which labels

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
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
= 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
= 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