NER USING CRF Model

Deevyansh Khadria(2022EE31883)

March 2025

1 What is NER?

Named Entity Recognition (NER) is a task that identifies and categorizes entities in text, such as names of people, organizations, locations, dates, and more. It is widely used in information extraction, search engines. Effective NER improves text understanding and is crucial for tasks like machine translation and sentiment analysis.

2 What is CRF Model? Why they are used for the sequence modeling?

Conditional Random Field are probabilistic(deterministic) model that captures the probability of a given squence over sequence.

HMM (Hidden Markov Model)	**CRF (Conditional Random Field)**
It is a generative model	It is a discriminative model
Models joint probability $P(y, x)$	Models conditional probability $P(y\|x)$
Emission probability at a point depends only on the corresponding word x_i	Emission probability depends on x_i and the entire sequence x
It cannot captures the features of the X and y	It can capture the features of X and y.

Table 1: Comparison between HMM and CRF

By comparision, we can conclude that the CRF is much better than HMM and can be used for the squence modelling tasks like POS tagging, NER.

3 Probability Modeling

Suppose we want to calculate the probability $P(y_1, y_2, \ldots, y_L \mid x_1, x_2, \ldots, x_L)$ for a particular index k. We define the following functions:

$$\phi^{(t)}(y_{i-1}, y_i) = W^\top f_t(y_{i-1}, y_i) \tag{1}$$
$$\phi^{(e)}(y_i, x, i) = W^\top f_e(y_i, x, i) \tag{2}$$

The probability is given by the product of transition and emission probabilities:

$$P(y_k \mid x_k) = P(y_k, y_{k-1}) \cdot P(y_k \mid x_k, k) \tag{3}$$

where:

$$P(y_k, y_{k-1}) = \exp(\phi^{(t)}(y_{k-1}, y_k)) \tag{4}$$
$$P(y_k \mid x_k, k) = \exp(\phi^{(e)}(y_k, k, x)) \tag{5}$$

Multiplying probabilities across all indices from 1 to L, we get:

$$P(y_1, y_2, \ldots, y_L \mid x_1, x_2, \ldots, x_L) = \exp\left(\sum_{i=1}^{L-1} \phi^{(t)}(y_i, y_{i+1}) + \sum_{i=1}^{L} \phi^{(e)}(y_i, i, x)\right) \tag{6}$$

To ensure that the obtained function is a valid probability distribution, we introduce the normalizing constant Z, defined as:

$$Z = \sum_{y'} \exp\left(\sum_{i=1}^{L-1} \phi^{(t)}(y'_i, y'_{i+1}) + \sum_{i=1}^{L} \phi^{(e)}(y'_i, i, x)\right) \tag{7}$$

Thus, the final expression for the conditional probability is:

$$P(y_1, y_2, \ldots, y_k \mid x_1, x_2, \ldots, x_k) = \frac{1}{Z} \exp\left(\sum_{i=1}^{k-1} \phi^{(t)}(y_i, y_{i+1}) + \sum_{i=1}^{k} \phi^{(e)}(y_i, i, x)\right) \tag{8}$$

Calculating the negative log probability

$$-\log P(y_1, y_2, \ldots, y_k \mid x_1, x_2, \ldots, x_k) = \log Z - \left(\sum_{i=0}^{k-1} \phi^{(t)}(y_i, y_{i-1}) + \sum_{i=0}^{k} \phi^{(e)}(y_i, i, x)\right)$$

4 Inference in CRFs using the Viterbi Algorithm

Inference in CRFs involves finding the most probable sequence of labels $y_1, y_2, \ldots, y_L$ given an input sequence $x_1, x_2, \ldots, x_L$. This is done using **Maximum A Posteriori (MAP) Inference**, where we seek:

$$\hat{y} = \arg\max_{y} P(y \mid x)$$

Since CRFs model the conditional probability as:

$$P(y \mid x) = \frac{1}{Z(x)} \exp\left(\sum_{i=1}^{L-1} \phi^{(t)}(y_i, y_{i+1}) + \sum_{i=1}^{L} \phi^{(e)}(y_i, i, x)\right)$$

finding the most likely sequence corresponds to maximizing the **log-probability**:

$$\hat{y} = \arg\max_{y} \sum_{i=1}^{L-1} \phi^{(t)}(y_i, y_{i+1}) + \sum_{i=1}^{L} \phi^{(e)}(y_i, i, x)$$

This can be efficiently solved using the **Viterbi Algorithm**, a dynamic programming approach.

5 Explanation for the Code:

Here is a description of all the functions and classes implemented in the provided code:

- **Compute log-likelihood**: It computes the negative log-likelihood as shown in my derivation. It consists of two terms: calculating the partition function and computing the score of y given x, which are handled in corresponding functions.
- **Compute score**: It uses the transition and emission matrices along with masks to batch-wise add the log transition and log emission probabilities. The time complexity is $O(\text{seq.length} \times \text{batch size})$.
- **Compute DP**: It employs a dynamic programming approach to calculate $\sum_{y'} P(y' \mid x)$. The time complexity is $O(\text{seq.length} \times \text{batch size} \times \text{num\_labels}^2)$. The recurrence relation is given by:

$$\alpha_t(l_i) = \alpha_{t-1}(l_i) + \sum_{0}^{k} (\log P_{\text{trans}}(l_k, l_i) + \log P_{\text{emit}}(l_i, i, x_i))$$

- **Viterbi algorithm**: It follows the same dynamic programming approach but computes the maximum over all scores instead of summing. It backtracks using a backpointer matrix to obtain the most probable sequence and its score. The recurrence relation is:

$$\alpha_t(l_i) = \left[\alpha_{t-1}(l_i) + \max_{l_k} (\log P_{\text{trans}}(l_k, l_i) + \log P_{\text{emit}}(l_i, i, x_i))\right]$$

6 Challenges Faced:

- A major challenge was calculating the gradient over a large number of examples. I used batch gradient descent, which significantly reduced the training time.
- Extracting emission probabilities from the features was difficult. I used a BiLSTM to encode the emission probabilities, as they depend on the entire input sequence.
- The high time complexity of computing the partition function and the Viterbi algorithm posed a challenge. I optimized this by PyTorch's matrix operations for parallel computation.

7 Results

NOTE: These results are trained on 1000 training set examples only.

NER Tag	**Precision**	**Recall**	**F1-Score**	**No. of Instances**
O	0.97	0.99	0.98	18248
B-gpe	0.75	0.34	0.47	333
B-tim	0.79	0.47	0.59	409
B-geo	0.68	0.80	0.74	792
I-geo	0.62	0.48	0.54	165
B-per	0.73	0.64	0.68	369
I-per	0.72	0.81	0.77	372
B-eve	0.00	0.00	0.00	2
I-eve	0.00	0.00	0.00	2
B-org	0.47	0.36	0.41	385
I-org	0.48	0.60	0.54	359
I-tim	0.36	0.32	0.34	145
B-nat	0.00	0.00	0.00	6
I-nat	0.00	0.00	0.00	2
I-gpe	0.00	0.00	0.00	6
B-art	0.00	0.00	0.00	3
I-art	0.00	0.00	0.00	4
Total	**91.9**	**92.6**	**92.24**	**21602**

Table 2: NER Traning Tagging Performance Metrics

NOTE: These results are tested on all test set examples.

NER Tag	Precision	Recall	F1-Score	No. of Instances
O	0.97	0.98	0.98	176877
B-gpe	0.63	0.31	0.41	3175
B-tim	0.74	0.41	0.53	4049
B-geo	0.66	0.77	0.71	7664
I-geo	0.54	0.42	0.48	1450
B-per	0.68	0.63	0.65	3389
I-per	0.71	0.83	0.77	3445
B-eve	0.00	0.00	0.00	60
I-eve	0.00	0.00	0.00	53
B-org	0.50	0.40	0.45	3913
I-org	0.49	0.59	0.54	3315
I-tim	0.29	0.24	0.27	1300
B-nat	0.00	0.00	0.00	50
I-nat	0.00	0.00	0.00	12
I-gpe	0.00	0.00	0.00	39
B-art	0.00	0.00	0.00	86
I-art	0.00	0.00	0.00	58
Total	**91.6**	**91.7**	**91.64**	**208690**

Table 3: NER Testing Tagging Performance Metrics

8 Insights

- The model performs well on the most frequent NER tags, such as `O`, `B-geo`, and `I-per`.
- Rare entity tags like `B-eve`, `B-art`, and `I-nat` have very poor recall and precision due to limited training examples.
- Increasing the size of word embedding does not increases the performance significantly.
- The model is very unstable at high learning rates like 0.1 and 0.01.