CSE 291: Advanced Statistical NLP

Project 2: Neural Conditional Random Fields for Named Entity Recognition

Due Friday May 28th by 11:59pm PT

1 Overview

In this assignment, you will be building a neural conditional random field (CRF) for the task of named entity recognition (NER). After introducing the task and modeling approach in Sections 2–4, we provide implementation instructions in Section 5. Finally, we describe deliverables in Section 6 and optional extensions to the project in Section 8.

2 Named Entity Recognition

Named Entity Recognition (NER)[1] is a type of information extraction task with the goal of identifying and classifying word spans (sequences of consecutive words in a sentence) into categories such as person, organization, location, time, and quantity. NER is an important processing step used to extract keywords for applications such as structured knowledge base creation, dictionary building, search, and voice command understanding. The task is best illustrated with an example:

`Andrew Viterbi co-founded Qualcomm, a company headquartered in San Diego.`

After labeling each named entity span with its category:

$[\texttt{Andrew Viterbi}]_{\texttt{PERSON}}$ `co-founded` $[\texttt{Qualcomm}]_{\texttt{ORGANIZATION}}$`, a company headquartered in` $[\texttt{San Diego}]_{\texttt{LOCATION}}$`.`

Even though, NER is a span/chunk identification and classification task, it is typically set up as a word tagging problem by annotating tokenized named entities with class-labeled Beginning-Inside-Outside (BIO) tags. In this format, words outside named entity chunks are labeled with an 'O' tag and words inside named entity chunks are labeled with their class label prefixed with 'I'. For entities that are immediately next to each other, the first word of the second entity gets labeled as 'B'. For example, the above, tokenized sentence under this labeling scheme looks like this:

$\texttt{Andrew}_{\texttt{I-PER}}$ $\texttt{Viterbi}_{\texttt{I-PER}}$ $\texttt{co-founded}_{\texttt{O}}$ $\texttt{Qualcomm}_{\texttt{I-ORG}}$ $,_{\texttt{O}}$ $\texttt{a}_{\texttt{O}}$ $\texttt{company}_{\texttt{O}}$ $\texttt{headquartered}_{\texttt{O}}$ $\texttt{in}_{\texttt{O}}$ $\texttt{San}_{\texttt{I-LOC}}$ $\texttt{Diego}_{\texttt{I-LOC}}$ $.\texttt{O}$

For this assignment, you will be using data from the CoNLL-2003 Shared Task[2], which only considers 4 classes of named entities: *persons*, *locations*, *organizations* and names of *miscellaneous* entities that do not belong to the previous three groups.

[1] https://en.wikipedia.org/wiki/Named-entity_recognition
[2] https://www.clips.uantwerpen.be/conll2003/ner/

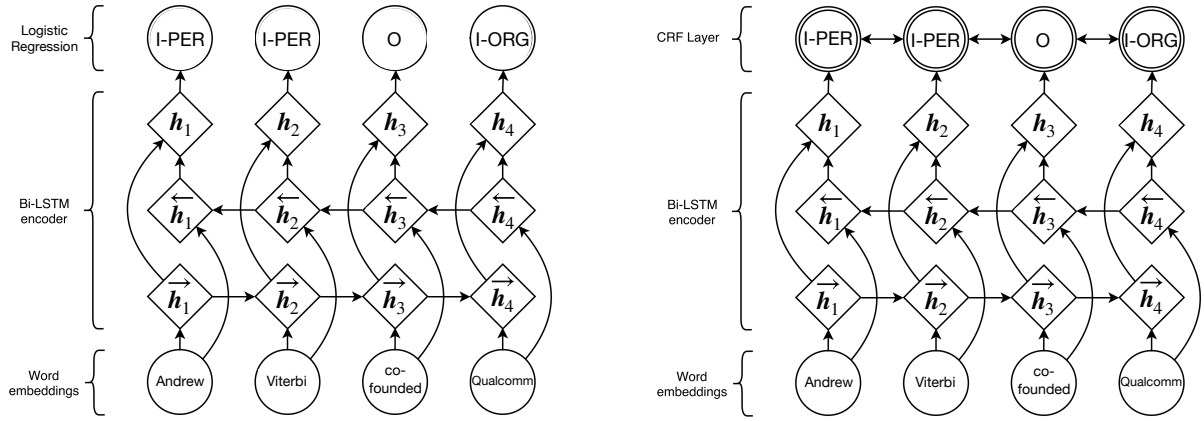

Figure 1: Neural network-based tagging models. On the left, a bidirectional RNN makes independent tag classifications per time step. On the right, a bidirectional RNN-CRF learns local sequential dependencies between neighboring output tags to predict the structured output tag sequence. Diagram based on [4].

3 Baseline: RNN for NER

As we've learned from lectures and the previous language modeling assignments, recurrent neural networks (RNNs) are a natural choice for computing learned feature representations of sequences. Given a labeled sentence $\{\boldsymbol{X}, \boldsymbol{Y}\}$ composed of a sequence of word embedding vectors $\boldsymbol{x}_1 \dots \boldsymbol{x}_T$ and tag embedding vectors $\boldsymbol{y}_1 \dots \boldsymbol{y}_T$, RNNs apply a learned recurrent function using input weights/bias $\boldsymbol{W_{ih}}, \boldsymbol{b_{ih}}$ and recurrent weights/bias $\boldsymbol{W_{hh}}, \boldsymbol{b_{hh}}$

$$\overrightarrow{\boldsymbol{h}}_t = \tanh(\boldsymbol{W_{ih}}\boldsymbol{x}_t + \boldsymbol{b_{ih}} + \boldsymbol{W_{hh}}\boldsymbol{h}_{(t-1)} + \boldsymbol{b_{hh}}). \tag{1}$$

For bidirectional RNNs, hidden states $\overrightarrow{\boldsymbol{h}}_t$, $\overleftarrow{\boldsymbol{h}}_t$ are computed in forward and backward directions, respectively, and concatenated to obtain $\boldsymbol{h}_t = \left[\overrightarrow{\boldsymbol{h}}_t; \overleftarrow{\boldsymbol{h}}_t\right]$.

In order to predict the associated tag for time step t, we apply a logistic regression classifier on input features $\boldsymbol{h}_t$ using learned weights/bias $\boldsymbol{W_{out}}, \boldsymbol{b_{out}}$ and the softmax function to obtain a probability distribution over the tag set vocabulary.

$$\boldsymbol{f}_t = \boldsymbol{W_{out}}\boldsymbol{h}_t + \boldsymbol{b_{out}} \tag{2}$$

$$p(\boldsymbol{y}_t^* \mid \boldsymbol{X}; \theta) = \text{softmax}(\boldsymbol{f}_t) \tag{3}$$

We show a diagram of a bidirectional RNN for NER on the left of Figure 1. (NB: We defined the model with a vanilla RNN for illustration, but you can (and probably should) use an LSTM here instead of an RNN.)

Learning: Similar to the previous assignments, parameters can be learned by minimizing the sum of the negative log likelihood under the model parameters θ for each prediction at each time step over training set $\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}^*$ using the ground truth tags at each time step denoted by $\boldsymbol{y}_t^*$:

$$\text{NLL} = \sum^{\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}^*} \sum_{t=1}^{T} -\log p(\boldsymbol{y}_t^* \mid \boldsymbol{X}; \theta) \tag{4}$$

In practice, we compute the loss stochastically over batches instead.

Decoding: We simply apply an argmax at each time step independently since each classification is performed independently.

4 Advanced: RNN-CRF for NER

Although the RNN tagger (Sec. 3) produces rich, context-sensitive feature representations of the input sentence, the independent tag classification decisions on top of these features are suboptimal for sequence labeling. Without any dependency between tags, the model is unable to learn the constraints that are common to sequence labeling tasks. For instance, in NER I-PER does not directly follow I-ORG, and in English part-of-speech tagging, adjectives only precede nouns. To remedy this, we can use a Conditional Random Field (CRF) layer to incorporate dependency structure between neighboring labels.

The CRF, introduced in [3], is a class of globally normalized, discriminative model that estimates the probability of an *entire sequence* $p(\boldsymbol{Y} \mid \boldsymbol{X})$ in a log-linear fashion on feature potentials (i.e., non-negative functions) Φ:

$$p(\boldsymbol{Y} \mid \boldsymbol{X}; \theta) = \frac{\Phi(\boldsymbol{X}, \boldsymbol{Y})}{\sum_{\boldsymbol{Y}' \in \mathcal{O}} \Phi(\boldsymbol{X}, \boldsymbol{Y}')}, \tag{5}$$

The denominator is also referred to as the "partition function'. The $\mathcal{O}$ in the denominator represents the set of all possible sequences of output tags over the input sentence. (NB: Strictly speaking, $\mathcal{O}$ should depend on the input sentence, $\boldsymbol{X}$, since the length of the input determines the length of the tag sequence. We omit this dependence for notational brevity.) This can present a major computational burden unless we assume that Φ is composed of a bunch of local potential functions that only operate on the preceding *local* neighbor $\boldsymbol{y}_{t-1}$:

$$\Phi(\boldsymbol{X}, \boldsymbol{Y}) = \sum_{t=1}^{T} \phi(\boldsymbol{X}, t, \boldsymbol{y}_{t-1}, \boldsymbol{y}_t). \tag{6}$$

Under this *linear chain* assumption, we can achieve tractable learning/decoding with dynamic programming algorithms.

Instead of manually defining our feature potentials, which takes considerable human effort, we can automatically learn nonlinear features in a deep neural network. To do this, we can use our transformed RNN input feature potentials (or "emissions"), $\boldsymbol{f}_t$, from Sec. 3 as our log emission potentials. Our log transition potentials can simply be a square weight matrix $\boldsymbol{U}$ over the tag set representing the weight of the transition from one tag to the next.

$$\Phi(\boldsymbol{X}, \boldsymbol{Y}) = \sum_{t=1}^{T} \phi_{\text{emission}}(\boldsymbol{X}, \boldsymbol{y}_t) + \phi_{\text{transition}}(\boldsymbol{y}_{t-1}, \boldsymbol{y}_t) \tag{7}$$

$$= \sum_{t=1}^{T} \exp(\boldsymbol{f}_{t, y_t}) + \exp(\boldsymbol{U}_{y_{t-1}, y_t}). \tag{8}$$

We show a diagram of a CRF layer on top of bidirectional RNN emissions on the right of Figure 1.

Learning: In order to estimate model parameters θ, we minimize the negative log likelihood over *entire sequences* in the dataset $\{\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}^*\}$:

$$\text{NLL} = \sum^{\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}^*} -\log p(\boldsymbol{Y}^* \mid \boldsymbol{X}; \theta) = \sum^{\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}^*} -\Big[\log \Phi(\boldsymbol{X}, \boldsymbol{Y}^*) - \log \sum_{\boldsymbol{Y} \in \mathcal{O}} \Phi(\boldsymbol{X}, \boldsymbol{Y})\Big] \tag{9}$$

Due to the sequential dependencies between output tags, the log partition term (last term in equation above) must be computed as part of a recurrence formula dependent on each time step's previous state. We can do this recurrence using the Forward algorithm dynamic program (NB: don't confuse this with the "forward" function API we call on PyTorch computation graphs), which computes the partition function. Once you have implemented the *Forward algorithm* as part of your forward pass in PyTorch, call `backward()` on the final output value and backpropagation will ensue, computing gradients of NLL with respect to your model parameters.

Decoding: During decoding we aim to find the most likely tag sequence under the model parameters θ

$$\arg\max_{\boldsymbol{Y} \in \mathcal{O}} p(\boldsymbol{Y} \mid \boldsymbol{X}; \theta), \tag{10}$$

which can be computed exactly using the Viterbi algorithm (see pseudocode at Wikipedia[3]), which has a recurrence very similar to the Forward algorithm dynamic program.

5 Implementation Instructions

Implement a linear chain CRF using the bidirectional recurrent parameterization defined above and use it for the task of NER. You are encouraged to clone and use the starter code from the CSE291 repository[4], which includes portions of the CoNLL-2003 Shared Task NER data train/dev splits, data loading functionality, and evaluation using span-based precision, recall, and F_1 score. Additionally, we provide a bidirectional LSTM tagger baseline `BiLSTMTagger` based on Section 3 that performs independent classifications over the tag set at each time step. We suggest that you add the required Forward algorithm for CRF negative log likelihood learning and the Viterbi algorithm for exact CRF decoding on top of the existing `BiLSTMTagger` class.

Please note that the added dependency structure of the CRF layer results in a longer runtime for both learning and prediction. We encourage you to use Google Colab[5] if you do not have access to a GPU. Feel free to report results on a subset of the data in your report if you do not have enough compute to train on the entire training set.

Using the provided evaluation procedure, we can visualize the chunk-level precision, recall, and F1 for each class of named entity. Here's the best scoring evaluation result (72.42 avg $F_{\beta=1}$) on the `dev.data.quad` for `BiLSTMTagger` after 30 epochs of training on `train.data.quad` (6 mins training time on Colab GPU, not vectorized or batched):

```
processed 11170 tokens with 1231 phrases; found: 1180 phrases; correct: 873.
accuracy:  75.00%; (non-O)
accuracy:  94.42%; precision:  73.98%; recall:  70.92%; FB1:  72.42
              LOC: precision:  83.85%; recall:  81.54%; FB1:  82.68  353
             MISC: precision:  80.14%; recall:  58.85%; FB1:  67.87  141
              ORG: precision:  62.25%; recall:  61.24%; FB1:  61.74  302
              PER: precision:  71.88%; recall:  74.80%; FB1:  73.31  384
```

And here's the best scoring evaluation result (74.03 avg $F_{\beta=1}$) on the `dev.data.quad` for our `BiLSTMCRFTagger` (20 mins training time on Colab GPU, vectorized but not batched):

```
processed 11170 tokens with 1231 phrases; found: 1141 phrases; correct: 878.
accuracy:  75.72%; (non-O)
```

[3] https://en.wikipedia.org/wiki/Viterbi_algorithm
[4] https://github.com/tberg12/cse291spr21/assignment2
[5] https://colab.research.google.com/

```
accuracy:  94.56%; precision:  76.95%; recall:  71.32%; FB1:  74.03
              LOC: precision:  87.65%; recall:  78.24%; FB1:  82.68  324
             MISC: precision:  81.43%; recall:  59.38%; FB1:  68.67  140
              ORG: precision:  68.86%; recall:  61.24%; FB1:  64.83  273
              PER: precision:  72.28%; recall:  79.13%; FB1:  75.55  404
```

As a reminder, we will not be grading based on the performance of your method, but the quality of the presentation and analysis in your report write-up.

6 Deliverables

Please submit a 2–3 page report along with the source code on Gradescope. We suggest using the ACL style files,[6] which are also available as an Overleaf template.[7] In your submission, you should describe your implementation choices and report performance using the evaluation code in the appropriate graphs and tables. In addition, we expect you to include some of your own investigation or error analysis. We are more interested in knowing what observations you made about the models or data than having a reiteration of the formal definitions of the various models. If you choose to complete an optional task, please present an analysis of your findings, along with the particular implementation decisions you made.

7 Useful References

While you can use the following PyTorch tutorial as a reference, the code in your implementation must be your own. `http://pytorch.org/tutorials/beginner/nlp/advanced_tutorial.html`

8 Optional Extensions

You may choose to carry out any or none of the following extra credit options:

- Choose a pre-trained language model (like BERT) and use its embeddings as input to the CRF model.
- Add local, hand-designed features to your neural CRF by creating feature functions aimed to encode important information for the task of NER, such as capitalization, word form, or the presence of a word in a gazetteer. Do these features help performance in low-resource training scenarios?
- Choose another sequence labeling task to apply your Neural CRF to. In your write-up, describe the task, the dataset, and the evaluation setup. Here are some suggested tasks:
  - Part of speech tagging in a language of your choice. How does the performance of your method scale with tag set size?
  - Co-reference resolution
  - NP Chunking (i.e., shallow parsing)
  - Word sense disambiguation

[6] `https://2021.aclweb.org/downloads/acl-ijcnlp2021-templates.zip`

[7] `https://www.overleaf.com/latex/templates/instructions-for-acl-ijcnlp-2021-proceedings/mhxffkjdwymb`

- Extend your linear chain CRF into a semi-Markov CRF for NER that *jointly* segments the text into the appropriate named entity phrase class (PER, LOC, etc.) and tags them with the appropriate I/O label.
- Choose your own adventure! Propose and implement your own analysis or extension.

Collaboration Policy

You are allowed to discuss the assignment with other students and collaborate on developing algorithms – you're even allowed to help debug each other's code! However, every line of your write-up and the new code you develop must be written by your own hand.

References

[1] Michael Collins. Log-Linear Models, MEMMs, and CRFs.

[2] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. arXiv preprint arXiv:1508.01991. 2015.

[3] John Lafferty, Andrew McCallum, Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. Proc. 18th International Conf. on Machine Learning. Morgan Kaufmann. pp. 282–289. 2001.

[4] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. arXiv preprint arXiv:1603.01360. 2016.

[5] Xuezhe Ma and Eduard Hovy. End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. *ACL*. 2016.

[6] Erik F. Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. *CoNLL.* 2003.