Assignment 3: Sequence Labeling

Henry Nguyen
SID: hnguye87

Google Colab Code:
https://drive.google.com/file/d/1K666PmTMQ8j4JRnptV3aljNwkiFBVinQ/view?usp=sharing

Google Colab Folder:
https://drive.google.com/drive/folders/1XccfgonoHcMuelr4ksuKDUrej-YhHjFO?usp=sharing

Problem

The assignment is about deriving and coding algorithms for a sequence labeling task, named entity recognition. The assignment can be broken down into three parts. The first part was to show that the scoring function can be shown as a recurrence. If the function is a recurrence, it is possible to apply the Viterbi algorithm to the decoder when decoding. The second part is coding the Viterbi algorithm to the decoder in the named entity recognition system. The last part is to build a model using the decoder from the last part. For this part we had to implement a stochastic gradient descent algorithm to minimize the loss function.

1.1.1

We can rewrite the function given in the assignment by breaking apart the two given equations.

$$\begin{aligned}\hat{Y} &= \underset{y \in Y(w)}{argmax}\, \Psi(w, y) \\ &= \underset{y_{1:M}}{argmax} \sum_{m=1}^{M+1} \psi(w, y_m, y_{m-1}, m) \\ &= \underset{y_{1:M}}{argmax} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1})\end{aligned}$$

(Eq. 1)

When doing structured prediction for sequence labeling, the goal is to find the sequence of tags which maximizes the global score. So we find the sequence with the best score.

$$\max_{y_{1:M}} \Psi(w, y_{1:M}) = \max_{y_{1:M}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1})$$

(Eq. 2)

The summation can be broken into $max_{ym} * max_{y1:m-1}$.

$$\max_{y_{1:M}} \Psi(w, y_{1:M}) = \max_{y_M} \max_{y_{1:M-1}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1})$$

(Eq. 3)

If we pull max_{ym} out the summation,

$$\max_{y_{1:M}} \Psi(w, y_{1:M}) = \max_{y_M} s_{M+1}(< STOP >, y_m) + \max_{y_{1:M-1}} \sum_{m=1}^{M} s_m(y_m, y_{m-1})$$

(Eq. 4)

The second portion of the equation is in the same form as Eq. 1 which shows that the equation can be remade into a recurrence.

1.1.2

Given the pseudo-code taken from the textbook,

Algorithm 11 The Viterbi algorithm. Each $s_m(k, k')$ is a local score for tag $y_m = k$ and $y_{m-1} = k'$.

for $k \in \{0, \dots K\}$ **do**
  $v_1(k) = s_1(k, \Diamond)$
for $m \in \{2, \dots, M\}$ **do**
  for $k \in \{0, \dots, K\}$ **do**
    $v_m(k) = \max_{k'} s_m(k, k') + v_{m-1}(k')$
    $b_m(k) = \text{argmax}_{k'} s_m(k, k') + v_{m-1}(k')$
$y_M = \text{argmax}_k s_{M+1}(\blacklozenge, k) + v_M(k)$
for $m \in \{M-1, \dots 1\}$ **do**
  $y_m = b_m(y_{m+1})$
return $y_{1:M}$

where K is the total number of tags and M is the total positions in the sequence, the time complexity would be $O(MK^2)$. There would be a total of M*K Viterbi variables to compute. Each Viterbi variable also requires looking at all K entries of the predecessor tags. So for all Viterbi variables, time complexity would be $K * MK = O(MK^2)$.

3.3.1

ner.dev

Accuracy: 41.38% (non-O)
Accuracy: 89.61%

	Precision	Recall	F1
LOC	87.52%	58.31%	69.99
MISC	69.94%	63.89%	66.78
ORG	36.04%	42.65%	39.07
PER	49.34%	11.90%	19.18
OVERALL	59.80%	41.25%	48.82

ner.test

Accuracy: 37.73% (non-O)
Accuracy: 88.02%

	Precision	Recall	F1
LOC	86.52%	55.88%	67.91
MISC	54.45%	50.64%	52.48
ORG	37.26%	44.93%	40.74
PER	32.75%	4.68%	8.19
OVERALL	53.28%	37.41%	43.96

3.3.2

The output can be seen in the link to the Google Colab. Ner.dev.out and ner.test.out is also in the submission.

Dev Output

```
processed 51578 tokens with 5917 phrases; found: 4082 phrases; correct: 2441.
accuracy:  41.38%; (non-O)
accuracy:  89.61%; precision:  59.80%; recall:  41.25%; FB1:  48.82
              LOC: precision:  87.53%; recall:  58.31%; FB1:  69.99  1219
             MISC: precision:  69.94%; recall:  63.89%; FB1:  66.78  835
              ORG: precision:  36.04%; recall:  42.65%; FB1:  39.07  1587
              PER: precision:  49.43%; recall:  11.90%; FB1:  19.18  441
```

Test Output

```
processed 46666 tokens with 5616 phrases; found: 3943 phrases; correct: 2101.
accuracy:  37.73%; (non-O)
accuracy:  88.02%; precision:  53.28%; recall:  37.41%; FB1:  43.96
              LOC: precision:  86.52%; recall:  55.88%; FB1:  67.91  1076
             MISC: precision:  54.45%; recall:  50.64%; FB1:  52.48  652
              ORG: precision:  37.26%; recall:  44.93%; FB1:  40.74  1986
              PER: precision:  32.75%; recall:   4.68%; FB1:   8.19  229
```

5.1.1

Dev Set

Accuracy: 73.08% (non-O)
Accuracy: 94.54%

	Precision	Recall	F1
LOC	94.71%	71.48%	81.47
MISC	82.33%	64.22%	72.16
ORG	59.12%	68.90%	63.64
PER	87.12%	64.25%	73.96
OVERALL	79.69%	67.53%	73.15

Test Set

Accuracy: 63.99% (non-O)
Accuracy: 91.67%

	Precision	Recall	F1
LOC	89.01%	68.07%	77.14
MISC	72.59%	55.92%	63.17
ORG	56.35%	59.26%	57.77
PER	84.17%	43.82%	57.64
OVERALL	73.15%	57.05%	64.11

5.1.2 Best Model from Early Stopping

model.iter9 performed best for the dev set and model.iter8 performed best for the test set. Both output files are submitted in the file and can be downloaded from Google Colab too.

5.1.3

The biggest negative pattern I can notice is the very rare usage of B-tags in the predictions. Even when the model has to make a compromise, it doesn't seem to pick B-tags. While B-tags should be more rare, I do not think it should have as little weight as it does. It is probably due to the B-tags having very little weight in the model.

5.1.4

5 Highest Features
t=I-LOC+w=Britain 34
t=I-LOC+w=Russia 31
t=I-ORG+w=Reuters 30
t=O+w=to 29
t=I-LOC+w=Belgium 28

5 Lowest Features
t=O+w=U.S. -38
t=O+w=South -30
t=O+w=United -29
t=O+w=England -27
t=O+w=Cup -26

For the highest features, the results show that Britain was tagged with I-LOC the most of any other combination in ner.train. The other four words, Russia, Reuters, to, and Belgium, were also tagged with their respective tag many times too. So the model gives those combinations a very high weight.

The reverse is true for the 5 lowest features, the word U.S. was tagged as O the least of any combination in ner.train. The other four terms, South, United, England, and Cup were tagged with their respective tags a very small amount in the training. So the model gives those combinations a very low weight.

Strange Features
t=I-ORG+w=Atlanta 8
t=I-PER+w=36.5 -1

The first strange feature, *t=I-ORG+w=Atlanta 8,* has a strange trend I noticed in the model. A word that many people would consider as a location, *Atlanta*, was given a fairly high weight tagged as an organization(ORG) rather than a location(LOC). This held true for many other city names in the model where the weight between the city word would be nearly the same for ORG and LOC.

The second strange feature was *t=I-PER+w=36.5 -1,* where the number 36.5, was tagged as a person(PER). While weight (-1), is not high, it is also not very low which is very strange to me. In what context would 36.5 be used as a person? Maybe the name of a robot or some sci-fi character possibly.