NLP HW3

Nimrod De La Vega 313547911

Tomer Peretz 305002206

Barak Levy 311431894

Q1:

a) In code.

Q2:

b) In code.

c) Most frequent accuracy on dev-set: 0.80.

```
Token-level scores:
label   acc     prec    rec     f1
PER     0.97    0.72    0.95    0.82
ORG     0.98    0.83    0.53    0.65
LOC     0.98    0.86    0.72    0.79
MISC    0.99    0.83    0.69    0.75
O       0.98    0.99    0.99    0.99
micro   0.98    0.95    0.95    0.95
macro   0.98    0.85    0.78    0.80
not-O   0.98    0.78    0.76    0.77

Entity level P/R/F1: 0.78/0.82/0.80
```

Q3:

a) In code.

b) + c) After performing grid search we conclude that our best achievable results are:

```
Token-level scores:
label    acc     prec    rec     f1
PER      0.98    0.77    0.95    0.85
ORG      0.98    0.74    0.71    0.72
LOC      0.98    0.88    0.69    0.77
MISC     0.99    0.87    0.73    0.79
O        0.98    0.99    0.99    0.99
micro    0.98    0.96    0.96    0.96
macro    0.98    0.85    0.81    0.83
not-O    0.98    0.80    0.79    0.80

Entity level P/R/F1: 0.82/0.82/0.82
Train and dev evaluation elapsed: 2.3074512481689453 seconds
```

We get these result with (for example) $\lambda_1 = 0.37, \lambda_2 = 0.47, \lambda_3 = 0.16$.

- We used pruning of the potential tag sets for each word by only considering tags which we know to have co-corruced with the word. This leads to a speedup in viterbi's run (of about 2 second (from around 4 to around 2)).

d)

```
/content/drive/My Drive/NLP_HW3/q1-4# python3 forward.py
CATEGORIES SOLUTION USED
P(x_8 = 'LOC') = 0.0226291346749790002
/content/drive/My Drive/NLP_HW3/q1-4#
```

Q.3e:

As-symbol →

'O' 'LOC' 'O'

[1] Google Jerusalem today

'ORG' 'O'

[2] Microsoft .

'ORG' 'O'

[3] Google ?

'O' 'LOC' 'O'

test-sentence: Google Jerusalem .

$c(*,*,O) = 1$; $c(*,*,ORG) = 2$; $c(*,'O','LOC') = 1$; $c(*,ORG,LOC) = 1$

$q(O \mid *,*) = \frac{1}{3}$; $q(ORG \mid *,*) = \frac{2}{3}$

$q(O \mid 'O',LOC) = 1$; $q(LOC \mid *,O) = 1$

$e(Google \mid O') = \frac{1}{4}$; $e(Google \mid ORG) = \frac{1}{2}$

$e(Jerusalem \mid Loc) = 1$; $q(O \mid O,Loc) = 1$

; $e(\cdot \mid 'O') = \frac{1}{4}$; $e(? \mid O) = \frac{1}{4}$; $e(Google \mid O) = \frac{1}{4}$

$e(today \mid O) = \frac{1}{4}$; $e(microsoft \mid ORG) = \frac{1}{2}$)

⇓

(1)

e)

$y_1 = \arg\max_{y' \in \mathcal{Y}} [q(y' | *, x) \cdot e(\text{Google} | y')] = \text{ORG}$

w.p. $(\frac{2}{3})(\frac{1}{2}) = \frac{1}{3}$

$y_2 = \arg\max_{y' \in \mathcal{Y}} [q(y' | x, \text{ORG}) \cdot e(\text{Jerusalem} | y')] = \text{any-tag}$

w.p. 0 [trivially, all y' yield pr = 0].

$y_3 = \text{any-tag}$.

As, $p_1 \cdot p_2 \cdot p_3 = (\frac{1}{3}) \cdot 0 \cdot p_3 = 0$

, regardless of the choice of y_3, y_2

the overall prob. of the greedy-sequence is zero.

In contrast, we could have used a non-greedy approach, and set $y_1 = \text{'O'}$, w.p. $(\frac{1}{4}) \cdot \frac{1}{3} = \frac{1}{12}$;

; $y_2 = \text{LOC}$ w.p. 1 , $y_3 = \text{'O'}$ w.p. $1 \cdot \frac{1}{4} = \frac{1}{4}$

$\Rightarrow p_1 \cdot p_2 \cdot p_3 = \frac{1}{12} \cdot 1 \cdot \frac{1}{4} = \frac{1}{48} > 0$

This shows a case for which a greedy-sequence can't achieve "global-arg max".

□

Q4 - Maximum Entropy Markov Model (MEMM) tagger

Optimization methods:

we have used 2 main optimization methods:

1. word-tags memorization – instead of rebuilding the data structure S during evaluation, we held a dictionary S which holds all the tags a certain word received in the training set. The base assumption was that a word we already seen, won't likely to get new tags. Of course this assumption does not always holds, yet in the running time – accuracy trade off, given the specific dev set, we thought it's a good deal. Whenever we encountered a new word, we predicted the tags.
2. Pruning – while iterating the pi table, we tried to minimize the number of times we need to predict the tag of word, since it is the expensive part, computational time wise. Hence, whenever the previous pi value was smaller than a given threshold, we skipped the calculation. This was done only if we have had this tag in the table, to prevent cases that certain cells in the pi table won't be initialized at all.

Another possible optimization would have been to hold the predictions vector given the features vector as a key. We have not implemented this optimization due to a lack of time.

Results

Viterbi: Entity level P/R/F1: 0.84/0.85/0.84

Greedy: Entity level P/R/F1: 0.85/0.86/0.86

Errors analysis

go\gu	PER	ORG	LOC	MISC	O	
PER	2818	96	49	60	**126**	3149
ORG	111	1622	101	62	**196**	2092
LOC	39	**176**	1751	47	81	2094
MISC	45	84	52	950	**137**	1268
O	65	**166**	105	226	42197	42759
	3078	2144	2058	1345	42737	51362

As can be seen from the chart, we were most likely to make mistakes when predicting 'O' tag, a possible reason could be the fact that our train set isn't balanced, and biased towards the 'O' tag.

Another interesting mistake is in locations that were most likely to be recognize as PER. From looking at the mistakes, we suspect it might relate to the fact that both start with capital letters. Thus, we needed to weighted more the parts of speech coming before hand, since the difference between 'in' and 'the' could indicate better on the following tag.

Q5 - BiLSTM tagger

Since our training data is padded with meaningless tags, calculating the grad with naïve CE may result in unwanted predictions. The padded data goal is only to align the vectors size entering the net. In order to overcome this problem, we mask the loss function, and thus prevent the network weights to be effect by these dummy-tags.

Results from the latest log:

label	acc	prec	rec	f1
PER	0.99	0.95	0.94	0.95
ORG	0.99	0.90	0.86	0.88
LOC	0.99	0.93	0.92	0.92
MISC	0.99	0.85	0.86	0.86
O	0.99	0.99	1.00	1.00
micro	0.99	0.98	0.98	0.98
macro	0.99	0.92	0.92	0.92
not-O	0.99	0.92	0.91	0.91

2020-12-28 11:14:53,827:INFO: Entity level P/R/F1: 0.88/0.88/0.88

2020-12-28 11:14:53,832:INFO: Training completed, saving predictions of the best model...