HW4: Word Segmentation

Virgile Audi
vaudi@g.harvard.edu
Nicolas Drizard
nicolasdrizard@g.harvard.edu
Github: virgodi/cs287/HW5

April 17, 2016

1 Introduction

The goal of this assignement is to tackle the NLP task of identifying and labeling contiguous segments of text. We will use sequence models and a dynamic programming method to find the best scoring sequence.

2 Problem Description

The idea is here to label continuous sequence of words with BIO tagging of different entities. The entities are the following:

1. PER: a person
2. LOC: a location
3. ORG: an organization
4. MISC:

Furthermore, this tagging method identifies the continuous group of words belonging to the same entity: the prefix B stop the current tag and begins a new one whereas the prefix I continues adding to the previous tag. However, in our solution we just cared about predicting the entity tag and then we were grouping the contiguous predictions into the same entity because the training text does not contain any B-tag.

3 Model and Algorithms

We used three different methods to solve this problem. The first two are the equivalent of first the Naive Bayes and second the logistic regression from text classification tasks. The last one introduces a customized way to train a neural architecture for this task.

3.1 Hidden Markov Model

We implement here a standard first order hidden Markov Model. The hidden states are the tags and the observed states are the features we built (word counts, capitalization...). The model can be represented with the following graphical model and requires two distribution: emission and transition.

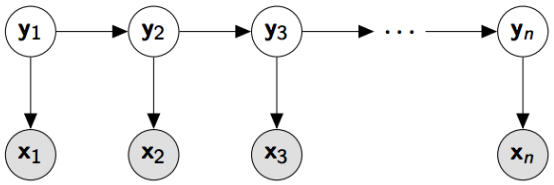


Figure 1: Graphical model of 1st order HMM with one feature

We represent the two distrubitions with multinomial as they model feature counts. As a result, we can infer them simply with the maximum likelihood estimator:

$$p(x_i = \delta(f)|y_i = \delta(c)) = \frac{F_{f,c}}{F_{.,c}}$$

$$p(y_i = \delta(c_i)|y_{i-1} = \delta(c_{i-1})) = \frac{T_{c_{i-1},c_i}}{T_{c_{i-1},.}}$$

with T_{c_{i-1},c_i} the counts of class c_{i-1} preceding class c_i and $F_{f,c}$ the counts of emission f with class c.

If we consider multiple features, then we still assume that the feature are indepent with each other (it's the main assumption in the Naive Bayes approach also). Only the emission distribution is changed and we can combine the probability together:

$$p(x_i = (\delta(f_1), \delta(f_2))|y_i = \delta(c)) = p(x_i = \delta(f_1)|y_i = \delta(c))p(x_i = \delta(f_2)|y_i = \delta(c)) = \frac{F_{f_1,c}}{F_{.,c}} \frac{F_{f_2,c}}{F_{.,c}}$$

3.2 Maximum-Entropy Markov Model

Next, we implemented a Maximum-Entropy Markov Model. The objective of the MEMM is to evaluate at each time step a distribution over the possible tags using features of the current word, denoted as $feat(x_i)$ and the tag of the previous word, c_{i-1}, using multi-class logistic regression, i.e.

$$p(\mathbf{y}_i|\mathbf{y}_{i-1}, feat(x_i)) = \text{softmax}([feat(x_i), c_{i-1}]\mathbf{W} + \mathbf{b})$$

3.3 Viterbi algorithm

The search algorithm that we implemented is the dynamic programming algorithm named after Andrew Viterbi. Its main difference with a greedy approach is that it evaluates at every step and for every previous state, the best possible next step. This guarantees a solution closer to the true optimal solution. The pseudo-code of the algorithm is given by:

procedure VITERBIWITHBP
  $\pi \in \mathbb{R}^{n+1 \times \mathcal{C}}$ initialized to $-\infty$
  $bp \in \mathcal{C}^{n \times \mathcal{C}}$ initialized to ϵ
  $\pi[0, \langle s \rangle] = 0$
  for $i = 1$ to n **do**
    for $c_{i-1} \in \mathcal{C}$ **do**
      compute $\hat{\boldsymbol{y}}(c_{i-1})$
      for $c_i \in \mathcal{C}$ **do**
        $score = \pi[i-1, c_{i-1}] + \log \hat{\boldsymbol{y}}(c_{i-1})_{c_i}$
        if $score > \pi[i, c_i]$ **then**
          $\pi[i, c_i] = score$
          $bp[i, c_i] = c_{i-1}$
  return sequence from bp

3.4 Structured Perceptron

The final model, we implemented is the structure perceptron train algorithm. The way the model is trained uses the Viterbi search algorithm, presented above. At each epoch, we uses Viterbi to predict the highest scored sequence given the state of the model. We can then find the timesteps where the actual sequence for the given sentence and the predicted one differ and compute at each of these time steps, the gradient of a hinge type loss. These gradients have a -1 entry on the true class for this given word, and a 1 on the predicted class by the model. We can then propagate these gradients in the network, and update the weights with a learning rate that can be tuned.

The model itself is similar to the model of the MEMM without the final logsoftmax layer.

4 Experiments

4.1 Feature Engineering

The original paper suggests several features to use. We focus on the word counts and a capitalization feature. We defined our capitalization feature as follow:

1. 1 : word in low caps;
2. 2 : whole word in caps;
3. 3 : first letter in cap;
4. 4 : one cap in the word;

5. 5 : other

We then produced an embedding of the word counts using a pre-trained version.

We also used the Python "pattern.en" package to extract Part-of-Speach (PoS) features. The packages generates 41 features to which we added special feature for the opening and closing tabs <s> and < \s>.

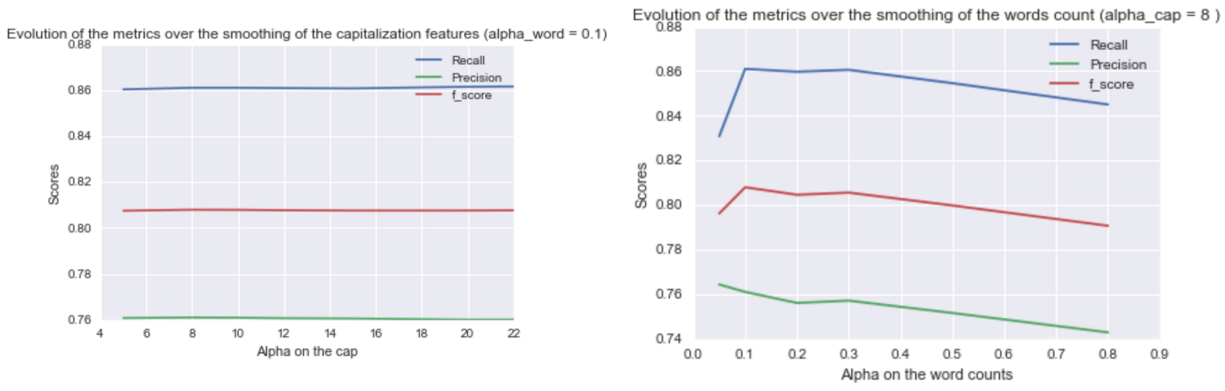
4.2 Model Evaluation

As used in the Kaggle competition, we used the f-score with the precision and recall measure to evaluate our model while tuning the hyperparameters. A positive prediction stands for a label (in the notation of the task, everything which is not the **O** tag):

1. recall: ratio of the true positive predictions among the positives tags in the correct sequence
2. precision: ratio of the true positive predictions among the positive predictions,
3. f-score (with $\beta = 1$): harmonic mean of the precision and the recall, i.e. $f_1 = \frac{2pr}{p+r}$

4.3 Hidden Markov Model

There is only the smoothing parameter α and eventually feature selection here to tune here. We evaluate the impact of adding more features and run experiments with different alpha values to tune them . One important details is to make sure to use a specific smoothing parameter for each distribution, i.e a smoothing parameter may be applied to the transition matrix but also to the emission matricx of each different feature. Each of this distribution has a different tail and need a different smoothing. For instance, the transition matrix need a very small α (around 0.1) because we are pretty confident in it but the capitalizations feature need one much bigger (around 20) because the counts are already high.



We notice that the model is less sensitive to the changes of the smoothing parameter on the capitalization feature as on the word counts. This is pretty reasonable as the feature coutns are much higher in the capitalization feature than in the word counts. Tuning this parameter provides

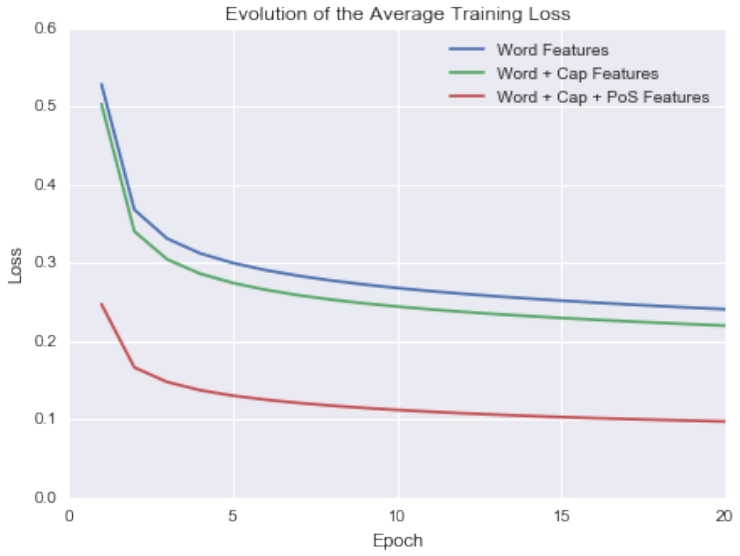
a model with a f-score of **0.808**. Using only the word counts features provide a best f-score of **0.764**.

We obtained a Kaggle score on the test set of :

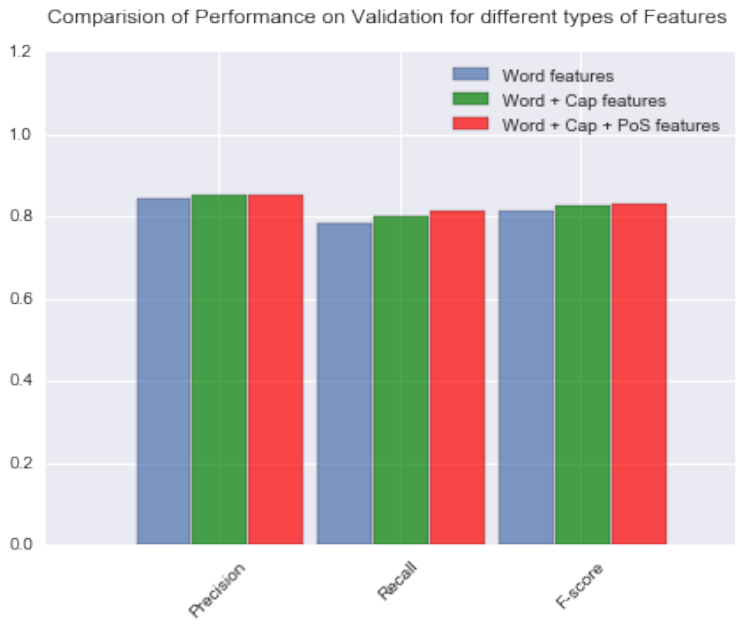
$$K_{HMM} = 0.48365$$

4.4 Maximum-Entropy Markov Model

We coded the MEMM using the nn module and trained using stochastic gradient descent. We also used the Glove embedggins using a lookup table. As for the HMM, we used two different sets of features, i.e. the words and the words and capitalisation of the words. We also added the Part of Speech features that were evaluated using the python package "pattern.en" in order to gain some time. We observed that the training algorithm converges quite rapidely, and that if adding caps to the features helped decrease the loss, the impact was not as strong as expected. On the other hand, adding PoS features impacted greatly the loss. Nevertheless, we trained the model on 20 epochs in order to learn the embeddings for the <s> and < \s> "words" added during pre-processing.



We evaluated the performance of these two models using the f-score presented above:



Adding extra features yielded better results on both Precision and Recall and therefore on the f-score. But as we expected from the small differences in loss, we did not observe an important increase on the f-score using cap features. We were nevertheless surprised to see that the impact on loss using PoS features did not translate on the f-score. These results were later confirmed on the test set, as the kaggle score obtained for these two models were:

$$K_{nocaps} = 0.52057 \quad \text{and} \quad K_{caps} = 0.55482 \quad K_{PoS} = 0.57121$$

which are both slightly better than the results of the HMM.

4.5 Structured Perceptron

We implemented the structured perceptron with the idea described in the model: weighting up the true edges in the lattice and down the incorrectly predicted. However we did not observe convincing results on our model, especially the f-score on the dev set was not increasing over the epochs but simply oscillating randomly around 0.72, which is not so bad but still less than what we obtained from the two other model.

We first tried a simple training version (our first train function in sp.lua) where for each timestep with a wrong prediction we do one forward/backward. The input is the right tag and we use a gradient with -1 on the right tag and 1 on the wrong one. We also coded an advanced version which was treating the two wrong edges of the lattice for each error (in our second train function) but did not observe the expected result.

5 Conclusion

We were disappointed to not be able to get the performance of the structured perceptron to the levels of the hidden and maximum-entropy markov models on the task of finding and labeling

named-entities in text. Due to the our difficulties at implementing the perceptron, we did not get the chance of implementing the NNMEM or add more features. Nevertheless, looking at the impact of the the few extra features implemented and the performance of the multi-class logistic regression, we believe that future work in that direction would yield better results.

Appendices

Preprocessing:

```
import numpy as np
import h5py
import re
import pattern.en
import sys
import argparse

from itertools import product


def get_tag2index():
    # Tags mapping
    tag2index = {}

    with open('data/tags.txt', 'r') as f:
        for line in f:
            line_split = line[:-1].split(' ')
            tag2index[line_split[0]] = int(line_split[1])

    # Adding tags for end/start of sentence
    tag2index['<t>'] = 8
    tag2index['<\t>'] = 9
    return tag2index


def get_pos2index():
    '''
    Part of speech tagging tags to feature index mapping
    '''
    # mapping for the POS tags
    tags = ['CC', 'CD', 'DT', 'EX', 'FW', 'IN', 'JJ', 'JJR', 'JJS', 'LS
        ', 'MD',
            'NN', 'NNS', 'NNP', 'NNPS', 'PDT', 'POS',
            'PRP', 'PRP$', 'RB', 'RBR', 'RBS', 'RP', 'SYM', 'TO', 'UH',
                'VB',
```

```
34              'VBZ', 'VBP', 'VBD', 'VBN', 'VBG', 'WDT', 'WP', 'WP$', 'WRB
                    ',
35              '.', ',', ':', '(', ')']
36
37      pos2index = {k: v+1 for v, k in enumerate(tags)}
38      return pos2index
39
40
41  def count_elements(filename, tags=True):
42      # Counting the number of elements to stored (ie num_words +
43      # 2*num_sentences)
44      num_words = 0
45      num_sentences = 0
46      with open(filename, 'r') as f:
47          for line in f:
48              if tags:
49                  line_split = line[:-1].split('\t')
50              else:
51                  line_split = line[:-1].split(' ')
52              # Case blank
53              if len(line_split) == 1:
54                  num_sentences += 1
55              else:
56                  num_words += 1
57
58      return num_words, num_sentences
59
60
61  def get_cap_feature(word):
62      # Return the caps feature for the given word
63      # 1 - low caps; 2 - all caps; 3 - first cap; 4 - one cap; 5 - other
64      if len(word) == 0 or word.islower() or re.search('[.?\-",]+', word)
            :
65          feature = 1
66      elif word.isupper():
67          feature = 2
68      elif len(word) and word[0].isupper():
69          feature = 3
70      elif sum([w.isupper() for w in word]):
71          feature = 4
72      else:
73          feature = 5
74      return feature
75
76
```

```
77  def get_tokenized_sentences(filename, tags=True):
78      # Build the part of speech tags
79      with open(filename, 'r') as f:
80          text = []
81          for line in f:
82              if tags:
83                  line_split = line[:-1].split('\t')
84              else:
85                  line_split = line[:-1].split(' ')
86              if len(line_split) != 1:
87                  text.append(line_split[2])
88
89      return pattern.en.tag(' '.join(text))
90
91
92  def build_input_matrix(filename, num_rows, tag2index, pos2index, tags=
        True, word2index=None, memm = False):
93      # Building input matrix with columns: (id, id_in_sentence, id_word,
            id_caps, id_token, id_tag)
94      # caps feature:
95      # 1 - low caps; 2 - all caps; 3 - first cap; 4 - one cap; 5 - other
96      # Tags: if correct solution given (ie 4th column)
97      # word2index: if use of previously built word2index mapping
98
99      # Features for starting/ending of sentence (3 last columns)
100     # For the POS tag, we use the same as a point (index 36)
101     # initialization
102     input_matrix = np.zeros((num_rows, 6), dtype=int)
103         if memm ==  False:
104         input_matrix[0] = [1, 1, 1, 1, 36, 8]
105         start = [1, 1, 36, 8]
106        end = [2, 1, 36, 9]
107     else:
108         input_matrix[0] = [1,1,word2index['<s>'],1,36,8]
109         start = [word2index['<s>'],1,36, 8]
110                 end = [word2index['<\s>'],1,36, 9]
111     row = 1
112
113     # Get the POS tokken
114     tokenized_sentences = get_tokenized_sentences(filename, tags=tags)
115     pos_i = 0
116
117     # Boolean to indicate if a sentence is starting
118     starting = False
```

```
119     # Boolean if a mapping is defined (last element of the mapping is
            for
120     # unknown words)
121     if word2index == None:
122         test = False
123         word2index = {'<s>': 1, '<\s>': 2}
124         id_word = 3
125     else:
126         test = True
127     with open(filename, 'r') as f:
128         for line in f:
129             if tags:
130                 line_split = line[:-1].split('\t')
131             else:
132                 line_split = line[:-1].split(' ')
133             if starting == True:
134                 # Start of sentence
135                 input_matrix[row, 0] = input_matrix[row-1, 0] + 1
136                 input_matrix[row, 1] = 1
137                 input_matrix[row, 2:] = start
138                 row += 1
139                 starting = False
140             if len(line_split) == 1:
141                 # End of sentence
142                 input_matrix[row, :2] = input_matrix[row-1, :2] + 1
143                 input_matrix[row, 2:] = end
144                 row += 1
145                 starting = True
146             else:
147                 # Indexing
148                 input_matrix[row, 0] = input_matrix[row-1, 0] + 1
149                 input_matrix[row, 1] = int(line_split[1]) + 1
150                 # Build cap feature
151                 word = line_split[2]
152                 input_matrix[row, 3] = get_cap_feature(word)
153                 # Build pos feature
154                 pos_tag = tokenized_sentences[pos_i][1].split('-')[0]
155                 if pos_tag in pos2index.keys():
156                     input_matrix[row, 4] = pos2index[pos_tag]
157                 else:
158                     input_matrix[row, 4] = len(pos2index) + 1
159                 pos_i += 1
160
161                 # Build word count feature
162                 word_clean = word.lower()
```

```
163                 if not test:
164                     if word_clean not in word2index:
165                         word2index[word_clean] = id_word
166                         id_word += 1
167                     input_matrix[row, 2] = word2index[word_clean]
168                 else:
169                     # Unseen word during train
170                     if word_clean not in word2index:
171                         input_matrix[row, 2] = len(word2index)
172                     else:
173                         input_matrix[row, 2] = word2index[word_clean]
174                 if tags:
175                     input_matrix[row, 5] = tag2index[line_split[3]]
176                 row += 1
177     # Add special word if training
178     if not test:
179         word2index['<unk>'] = len(word2index)+1
180     if tags:
181         return input_matrix, word2index
182     else:
183         return input_matrix[:, :5], word2index
184
185 #Function that formats the output of the previous function in order to
       run MEMM:
186 def input_mm_pos(matrix):
187
188     nwords = matrix.shape[0]
189
190     res = np.zeros((nwords,1 + 9 + 5 + 43 + 1),dtype = int)
191
192     res[:,0] = matrix[:,2]
193
194     for i in range(nwords):
195         tag_1_hot = np.zeros(9)
196         tag_1_hot[matrix[i,5]-1] = 1
197         tag_1_hot_cap = np.zeros(5)
198         tag_1_hot_cap[matrix[i,3]-1] = 1
199         tag_1_hot_pos = np.zeros(43)
200         tag_1_hot_pos[matrix[i,4]] = 1
201         res[i,1:10] = tag_1_hot
202         res[i,10:15] = tag_1_hot_cap
203         res[i,15:58] = tag_1_hot_pos
204     res[:,58] = matrix[:,5]
205     return res
206
```

```
207
208 def train_hmm(input_matrix, num_features, num_pos, num_tags):
209     # Emission word_count matrix:
210     # size (num_words, num_tags)
211     # row: observation / colum: tag
212     # (un-normalized if smoothing required)
213     emission_w = np.zeros((num_features, num_tags), dtype=int)
214
215     # Emission caos_count matrix:
216     # size (5, num_tags)
217     # row: observation / colum: caps
218     # (un-normalized if smoothing required)
219     emission_c = np.zeros((5, num_tags), dtype=int)
220
221     # Emission pos_count matrix:
222     # size (5, num_tags)
223     # row: observation / colum: pos tag
224     # (un-normalized if smoothing required)
225     emission_p = np.zeros((num_pos, num_tags), dtype=int)
226
227     # Building
228     for r in input_matrix:
229         emission_w[r[2]-1, r[5]-1] += 1
230         emission_c[r[3]-1, r[5]-1] += 1
231         emission_p[r[4]-1, r[5]-1] += 1
232
233     # Transition matrix
234     # size (num_tags, num_tags)
235     # row: to / colum: from
236     # (un-normalized if smoothing required)
237     transition = np.zeros((num_tags, num_tags), dtype=int)
238     for i in xrange(input_matrix.shape[0] - 1):
239         transition[input_matrix[i+1, 5]-1, input_matrix[i, 5]-1] += 1
240
241     return emission_w, emission_c, emission_p, transition
242
243
244 def main(arguments):
245     # Args
246     global args
247     parser = argparse.ArgumentParser(
248         description=__doc__,
249         formatter_class=argparse.RawDescriptionHelpFormatter)
250
251     parser.add_argument('--f', default='data/features.hdf5',
```

```
252                         type=str, help='Filename to save data')
253     args = parser.parse_args(arguments)
254     filename = args.f
255
256     # Train
257     pos2index = get_pos2index()
258     tag2index = get_tag2index()
259     num_words, num_sentences = count_elements('data/train.num.txt')
260     num_rows = num_words + 2*num_sentences
261     input_matrix_train, word2index = build_input_matrix('data/train.num
            .txt',
262                                                         num_rows,
                                                            tag2index,
263                                                         pos2index)
264
265     # Building the count matrix
266     num_tags = len(tag2index)
267     num_features = len(word2index)
268     num_pos = len(pos2index) + 1
269     emission_w, emission_c, emission_p, transition = train_hmm(
            input_matrix_train,
270                                                             num_features
                                                                ,
                                                                num_pos
                                                                ,
271                                                             num_tags
                                                                )
272
273     # Dev & test
274     num_words, num_sentences = count_elements('data/dev.num.txt')
275     # Miss 1 blank line at the end of the file for the dev set
276     num_rows = num_words + 2*num_sentences + 1
277     input_matrix_dev, word2index = build_input_matrix('data/dev.num.txt
            ',
278                                                       num_rows,
                                                            tag2index,
279                                                       pos2index,
280                                                       word2index=
                                                            word2index)
281
282     num_words, num_sentences = count_elements('data/test.num.txt',
283                                               tags=False)
284     num_rows = num_words + 2*num_sentences
285     input_matrix_test, word2index = build_input_matrix('data/test.num.
            txt',
```

```
286                                                                 num_rows,
                                                                        tag2index,
287                                                                 pos2index,
288                                                                 tags=False,
289                                                                 word2index=
                                                                        word2index)
290
291         # Saving pre-processing
292         with h5py.File(filename, "w") as f:
293             # Model
294             f['emission_w'] = emission_w
295             f['emission_c'] = emission_c
296             f['emission_p'] = emission_p
297             f['transition'] = transition
298
299             f['input_matrix_train'] = input_matrix_train
300             f['input_matrix_dev'] = input_matrix_dev
301             f['input_matrix_test'] = input_matrix_test
302
303
304 if __name__ == '__main__':
305     sys.exit(main(sys.argv[1:]))
```

Hidden Markov Model:

```
1  -- Documentation:
2  -- ----- How to call it from the command line?
3  -- For example:
4  -- $ th count_based.lua -N 5
5  -- Other argument possible (see below)
6  --
7  -- ----- Is there an Output?
8  -- By default, the predictions on the test set are saved in hdf5 format
       as classifier .. opt.f
9
10 -- Only requirements allowed
11 require("hdf5")
12 require 'helper.lua';
13
14 cmd = torch.CmdLine()
15
16 -- Cmd Args
17 cmd:option('-datafile', 'data/words_feature.hdf5',
18             'Datafile with features in hdf5 format')
19 cmd:option('-alpha_t', 0.1, 'Smoothing parameter alpha in the
       transition counts')
```

```
20 cmd:option('-alpha_w', 0.1, 'Smoothing parameter alpha in the word
       counts')
21 cmd:option('-alpha_c', 8, 'Smoothing parameter alpha in the caps counts
       ')
22 cmd:option('-alpha_p', 2, 'Smoothing parameter alpha in the pos counts
       ')
23 cmd:option('-test', 0, 'Boolean (as int) to ask for a prediction on
       test, will be saved in submission in hdf5 format')
24 cmd:option('-datafile_test', 'submission/v_seq_hmm', 'Smoothing
       parameter alpha in the word counts')
25 cmd:option('-nfeatures', 2, 'Number of type of features to use')
26 cmd:option('-cv', 0, 'Boolean (as int) to run a cross validation
       pipeline')
27
28
29
30 -- Formating as log-probability and smoothing the input
31 function format_matrix(matrix, alpha)
32     local formatted_matrix = matrix:clone():type('torch.DoubleTensor')
33     formatted_matrix:add(alpha)
34     -- Normalize
35     local norm_mat = torch.expandAs(formatted_matrix:sum(1),
           formatted_matrix)
36     formatted_matrix:cdiv(norm_mat)
37     return formatted_matrix:log()
38 end
39
40 -- log-scores of transition and emission
41 -- corresponds to the vector y in the lecture notes
42 -- i: timestep for the computed score
43 function score_hmm(observations, i, emissions, transition, C, nfeatures
       )
44     local observation_emission = torch.zeros(C)
45     for k=1,nfeatures do
46         -- print(i,k)
47         -- print(emissions[k][observations[{i,k}]])
48         observation_emission:add(emissions[k][observations[{i,k}]])
49     end
50     observation_emission = observation_emission:view(C, 1):expand(C, C)
51     -- NOTE: allocates a new Tensor
52     return observation_emission + transition
53 end
54
55 -- Viterbi algorithm.
56 -- observations: a sequence of observations, represented as integers
```

```
57 -- logscore: the edge scoring function over classes and observations in
       a history-based model
58 function viterbi(observations, logscore, emissions, transition,
       nfeatures)
59     local y
60     -- Formating tensors
61     local initial = torch.zeros(transition:size(2), 1)
62     -- initial started with a start of sentence: <t>
63     initial[{8,1}] = 1
64     initial:log()
65
66     -- number of classes
67     C = initial:size(1)
68     local n = observations:size(1)
69     local max_table = torch.Tensor(n, C)
70     local backpointer_table = torch.Tensor(n, C)
71
72     -- first timestep
73     -- the initial most likely paths are the initial state distribution
74     -- NOTE: another unnecessary Tensor allocation here
75     local init_pred = initial:clone()
76     for i=1,nfeatures do
77         init_pred:add(emissions[i][observations[{1,i}]])
78     end
79     local maxes, backpointers = init_pred:max(2)
80     max_table[1] = maxes
81
82     -- remaining timesteps ("forwarding" the maxes)
83     for i=2,n do
84         -- precompute edge scores
85         y = logscore(observations, i, emissions, transition, C,
               nfeatures)
86         scores = y + maxes:view(1, C):expand(C, C)
87
88         -- compute new maxes (NOTE: another unnecessary Tensor
               allocation here)
89         maxes, backpointers = scores:max(2)
90
91         -- record
92         max_table[i] = maxes
93         backpointer_table[i] = backpointers
94     end
95     -- follow backpointers to recover max path
96     local classes = torch.Tensor(n)
97     maxes, classes[n] = maxes:max(1)
```

```
 98      for i=n,2,-1 do
 99          classes[i-1] = backpointer_table[{i, classes[i]}]
100      end
101
102      return classes
103  end
104
105  -- Prediction pipeline
106  function predict(observations, emissions, transition, alphas, nfeatures
         )
107      -- Formating model parameters (log and alpha smoothing)
108      -- Alphas is a tensor : {alpha_t, alpha_w, alpha_c}
109      emissions_cleaned = {}
110      for i=1,nfeatures do
111          emissions_cleaned[i] = format_matrix(emissions[i], alphas[i+1])
112      end
113      local transition_cleaned = format_matrix(transition, alphas[1])
114
115      return viterbi(observations, score_hmm, emissions_cleaned,
             transition_cleaned, nfeatures)
116  end
117
118  -- Cross validation pipeline
119  function cross_validation(observations, emissions, transitions,
         true_classes,
120                            alphas_table, alpha_t)
121      -- alphas_table is a table of tensor with the range of parameters
             to use
122      -- Current implementation for 2 features only
123      -- alphas_table = {alpha_w_tensor, alpha_c_tensor}
124      -- Return a tensor with first columns the alpha value and last the
             score for each
125      local nfeatures = #alphas_table
126      local v_seq_dev, precision, recall, f
127      local alphas = torch.DoubleTensor(3)
128      local size1 = alphas_table[1]:size(1)
129      local size2 = alphas_table[2]:size(1)
130      local num_evaluations = size1*size2
131
132      -- Columns for 2 features are (alphas_w_value, alphas_c_value,
             f_score, precision, recall)
133      local scores = torch.DoubleTensor(num_evaluations, nfeatures+3)
134
135      for i=1,size1 do
136          alpha_w = alphas_table[1][i]
```

```
137         for k=1,size2 do
138             alpha_c = alphas_table[2][k]
139
140             alphas:copy(torch.Tensor({alpha_t, alpha_w, alpha_c}))
141             v_seq_dev = predict(observations, emissions, transition,
                    alphas, nfeatures)
142             precision, recall = compute_score(v_seq_dev, true_classes)
143             f = f_score(precision, recall)
144
145             -- Filling the scores tensor
146             scores[{(i-1)*size2+k, 1}] = alpha_w
147             scores[{(i-1)*size2+k, 2}] = alpha_c
148             scores[{(i-1)*size2+k, 3}] = f
149             scores[{(i-1)*size2+k, 4}] = precision
150             scores[{(i-1)*size2+k, 5}] = recall
151         end
152     end
153
154     return scores
155 end
156
157
158 function main()
159     -- Parse input params
160     opt = cmd:parse(arg)
161
162     -- Reading file from pre-processing
163     myFile = hdf5.open(opt.datafile,'r')
164     data = myFile:all()
165     emission_w = data['emission_w']
166     emission_c = data['emission_c']
167     emission_p = data['emission_p']
168     print(emission_p:size())
169     -- Table of emission tensor (one tensor per feature)
170     emissions = {emission_w, emission_c, emission_p}
171     -- Assertion on number of features
172     nfeatures = opt.nfeatures
173     if nfeatures > #emissions then
174         error('Too many features specified')
175     end
176     print('Number of features used: '..nfeatures)
177     transition = data['transition']
178     input_matrix_train = data['input_matrix_train']
179     input_matrix_dev = data['input_matrix_dev']
180     input_matrix_test = data['input_matrix_test']
```

```
181     myFile:close()
182
183     -- Parameters:
184     true_classes = input_matrix_dev:narrow(2,6,1):clone():view(
            input_matrix_dev:size(1))
185     -- contain in each column feature observation
186     -- (same order as the feature emission tensor in the emissoins
            table)
187     observations = input_matrix_dev:narrow(2,3,nfeatures):clone()
188     -- Alpha parameter
189     alphas = torch.Tensor({opt.alpha_t, opt.alpha_w, opt.alpha_c, opt.
            alpha_p})
190
191     -- Prediction on dev
192     v_seq_dev = predict(observations, emissions, transition, alphas,
            nfeatures)
193     print(v_seq_dev:size(1))
194     precision, recall = compute_score(v_seq_dev, true_classes)
195     f = f_score(precision, recall)
196
197     print('Prediction on dev')
198     print('Precision is : '..precision)
199     print('Recall is : '..recall)
200     print('F score (beta = 1) is : '..f)
201
202     -- Cross validation
203     if (opt.cv == 1) then
204         alphas_table = {}
205         -- alpha_w
206         alphas_table[1] = torch.Tensor({0.05, 0.1, 0.2, 0.3, 0.5, 0.8})
207         -- alpha_c
208         alphas_table[2] = torch.Tensor({5, 8, 10, 12, 15, 20, 22})
209
210         scores = cross_validation(observations, emissions, transitions,
                true_classes,
211                                   alphas_table, opt.alpha_t)
212         print(scores)
213
214         -- Saving the score
215         myFile = hdf5.open('plot_scores.hdf5', 'w')
216         myFile:write('scores', scores)
217         myFile:close()
218         print('CV on dev saved in '..'plot_scores.hdf5')
219     end
220
```

```
221     -- Prediction on test
222     if (opt.test == 1) then
223         print('Prediction on test')
224         observations_test = input_matrix_test:narrow(2,3,nfeatures):
                clone()
225         v_seq_test = predict(observations_test, emissions, transition,
                alphas, nfeatures)
226         -- Saving predicted sequence on test
227         myFile = hdf5.open(opt.datafile_test, 'w')
228         myFile:write('v_seq_test', v_seq_test)
229         myFile:write('v_seq_dev', v_seq_dev)
230         myFile:close()
231         print('Sequence predicted on test saved in '..opt.datafile_test
                )
232     end
233
234 end
235
236 main()
```

Max-Entropy Markov Model:

```
1  require 'hdf5';
2  require 'nn';
3  require 'helper.lua';
4
5  -- Loading data
6  myFile = hdf5.open('../data/MM_data_pos.hdf5','r')
7  data = myFile:all()
8  input_matrix_train_pos = data['input_matrix_train_pos']
9  input_matrix_dev_pos = data['input_matrix_dev_pos']
10 input_matrix_test_pos = data['input_matrix_test_pos']
11 myFile:close()
12
13 nwords = input_matrix_train_pos:size(1)
14 train_output = input_matrix_train_pos:narrow(2,59)
15 train_input_pos = torch.Tensor(nwords-1,1+9+5+43)
16 train_input_pos:narrow(2,1,1):copy(input_matrix_train_pos:narrow(2,1,1)
       :narrow(1,2,nwords-1))
17 train_input_pos:narrow(2,2,9):copy(input_matrix_train_pos:narrow(2,2,9)
       :narrow(1,1,nwords-1))
18 train_input_pos:narrow(2,11,5):copy(input_matrix_train_pos:narrow
       (2,11,5):narrow(1,1,nwords-1))
19 train_input_pos:narrow(2,16,43):copy(input_matrix_train_pos:narrow
       (2,16,43):narrow(1,1,nwords-1))
20
```

```
21 observations_dev = input_matrix_dev_pos:narrow(2,1,1):clone()
22 dev_feat = input_matrix_dev_pos:narrow(2,11, 5 + 43)
23 dev_true_classes = input_matrix_dev_pos:narrow(2, 59,1):squeeze()
24
25 observations_test_pos = input_matrix_test_pos:narrow(2,1,1)
26 observations_test_feat = input_matrix_test_pos:narrow(2,2,5+43)
27
28 -- Defining the model
29
30 model = nn.Sequential()
31 t1_pos = nn.ParallelTable()
32
33 t1_pos_1 = nn.Sequential()
34 t1_pos_1:add(LT)
35 t1_pos_1:add(nn.View(-1,50))
36
37 t1_pos_2 = nn.Identity()
38
39 t1_pos:add(t1_pos_1)
40 t1_pos:add(t1_pos_2)
41
42 model:add(t1_pos)
43 model:add(nn.JoinTable(2))
44
45 model:add(nn.Linear(50 + 9 + 5 + 43,9))
46 model:add(nn.LogSoftMax())
47
48 -- Training function:
49
50
51 function train_model_cap(train_input, train_output, model, criterion,
       din, nclass, eta, nEpochs, batchSize)
52     -- Train the model with a mini batch SGD
53     -- standard parameters are
54     -- nEpochs = 1
55     -- batchSize = 32
56     -- eta = 0.01
57     local loss = torch.Tensor(nEpochs)
58
59     -- To store the loss
60     local av_L = 0
61
62     -- Memory allocation
63     local inputs_batch = torch.DoubleTensor(batchSize, din)
64     local targets_batch = torch.DoubleTensor(batchSize)
```

```
65      local outputs = torch.DoubleTensor(batchSize, nclass)
66      local  df_do = torch.DoubleTensor(batchSize, nclass)
67
68      for i = 1, nEpochs do
69          -- timing the epoch
70          timer = torch.Timer()
71          av_L = 0
72
73          -- mini batch loop
74          for t = 1, train_input:size(1), batchSize do
75              -- Mini batch data
76              current_batch_size = math.min(batchSize,train_input:size(1)
                    -t)
77
78              inputs_batch:narrow(1,1,current_batch_size):copy(
                    train_input:narrow(1,t,current_batch_size))
79
80              targets_batch:narrow(1,1,current_batch_size):copy(
                    train_output:narrow(1,t,current_batch_size))
81
82              -- reset gradients
83              model:zeroGradParameters()
84
85              -- Forward pass (selection of inputs_batch in case the
                    batch is not full, ie last batch)
86              outputs:narrow(1,1,current_batch_size):copy(model:forward({
                    inputs_batch:narrow(1,1,current_batch_size):narrow
                    (2,1,1),
87              inputs_batch:narrow(1,1,current_batch_size):narrow(2,2,din
                    -1)}))
88              -- Average loss computation
89              f = criterion:forward(outputs:narrow(1,1,current_batch_size
                    ), targets_batch:narrow(1,1,current_batch_size))
90
91              av_L = av_L +f
92
93              -- Backward pass
94              df_do:narrow(1,1,current_batch_size):copy(criterion:
                    backward(outputs:narrow(1,1,current_batch_size),
                    targets_batch:narrow(1,1,current_batch_size)))
95              model:backward({inputs_batch:narrow(1,1,current_batch_size)
                    :narrow(2,1,1), inputs_batch:narrow(1,1,
                    current_batch_size):narrow(2,2,din-1)},
96              df_do:narrow(1,1,current_batch_size))
97
```

```
98                 model:updateParameters(eta)
99
100            end
101
102            print('Epoch '..i..': '..timer:time().real)
103            loss[i] = av_L/math.floor(train_input:size(1)/batchSize)
104            print('Average Loss: '.. loss[i])
105
106        end
107
108        return loss
109    end
110
111    -- Viterbi for MEMM:
112
113    -- Evaluates the matrix of scores for all possible  tags for the
       previous word, using the word features at timestep i
114
115    function compute_logscore_extrafeat(observations, feat, i, model, C)
116        local y = torch.zeros(C,C)
117        local hot_1 = torch.zeros(C+feat:size(2))
118        for j = 1, C do
119            hot_1:zero()
120            hot_1[j] = 1
121            hot_1:narrow(1,10,feat:size(2)):copy(feat:narrow(1,i,1))
122            y:narrow(1,j,1):copy(model:forward({observations[i]:view(1,1),
                   hot_1:view(1,C+feat:size(2))}))
123        end
124        return y
125    end
126
127    -- Evaluates the highest scoring sequence:
128    function viterbi_extrafeat(observations, feat, compute_logscore, model,
       C)
129
130        local y = torch.zeros(C,C)
131        -- Formating tensors
132        local initial = torch.zeros(C, 1)
133        -- initial started with a start of sentence: <t>
134
135        initial[{8,1}] = 1
136        initial:log()
137
138        -- number of classes
139        local n = observations:size(1)
```

```
140     local max_table = torch.Tensor(n, C)
141     local backpointer_table = torch.Tensor(n, C)
142     -- first timestep
143     -- the initial most likely paths are the initial state distribution
144     local maxes, backpointers = (initial + compute_logscore_extrafeat(
            observations, feat, 1, model, C)[8]):max(2)
145     max_table[1] = maxes
146     -- remaining timesteps ("forwarding" the maxes)
147     for i=2,n do
148         -- precompute edge scores
149
150         y:copy(compute_logscore_extrafeat(observations, feat, i, model,
                C))
151         scores = y:transpose(1,2) + maxes:view(1, C):expand(C, C)
152
153         -- compute new maxes
154         maxes, backpointers = scores:max(2)
155
156         -- record
157         max_table[i] = maxes
158         backpointer_table[i] = backpointers
159     end
160     -- follow backpointers to recover max path
161     local classes = torch.Tensor(n)
162     maxes, classes[n] = maxes:max(1)
163     for i=n,2,-1 do
164         classes[i-1] = backpointer_table[{i, classes[i]}]
165     end
166
167     return classes
168 end
169
170 -- Train Model
171
172 loss_pos = train_model_cap(train_input_pos, train_output,
        ultimate_t_pos, criterion, 1 + 9 + 5 + 43, 9, 0.1, 20, 32)
173
174 -- Evaluate performance on dev set:
175
176 cl_pos_dev = viterbi_extrafeat(observations_dev, dev_feat,
        compute_logscore_extrafeat, ultimate_t_pos, 9)
177 f = f_score(cl_pos_dev, dev_true_classes)
178
179 -- Predict on test:
180
```

```
181 v_seq_test_pos = viterbi_extrafeat(observations_test_pos,
      observations_test_feat, compute_logscore_extrafeat, ultimate_t_pos,
      9)
182
183 -- Saving predicted sequence on test
184 myFile = hdf5.open('../submission/v_seq_test_mem_pos', 'w')
185 myFile:write('v_seq_test', v_seq_test_pos)
186 myFile:close()
```

Structured Perceptron:

```
 1 function train_model(train_input, sent, train_output, observations_dev,
       model, din, nclass, eta, nEpochs)
 2     -- Train the model with the structured perceptron approach
 3     -- V1: only treating the eged leaving the error
 4
 5     -- For the verbose print
 6     observations = observations_dev:narrow(2,1,1):narrow(1,1,1000):
          clone()
 7     true_classes = observations_dev:narrow(2,16,1):narrow(1,1,1000):
          squeeze()
 8
 9     -- Memory allocation
10     inputs_batch = torch.DoubleTensor(100, din)
11     gold_sequence = torch.DoubleTensor(100)
12     high_score_seq = torch.DoubleTensor(100)
13     grad_pos = torch.zeros(9)
14     grad_neg = torch.zeros(9)
15     pr1 = torch.zeros(9)
16     pr2 = torch.zeros(9)
17
18     for i = 1, nEpochs do
19         -- timing the epoch
20         timer = torch.Timer()
21
22         -- mini batch loop
23         for t = 2, sent:size(1)-1 do
24             -- Mini batch data
25             sent_size = sent[{t,2}]
26 --              print('here1')
27
28             inputs_batch:narrow(1,1,sent_size+1):copy(train_input:
                  narrow(1,sent[{t,1}]-1,sent_size+1))
29 --              print('here2')
30
31             gold_sequence:narrow(1,1,sent_size+1):copy(train_output:
```

```
                narrow(1,sent[{t,1}]-1,sent_size+1))
32 --              print('here3')
33
34              -- reset gradients
35              model:zeroGradParameters()
36              --gradParameters:zero()
37
38              -- Forward pass on a batch subsequence:
39              high_score_seq:narrow(1,1,sent_size+1):copy(viterbi(
                   inputs_batch:narrow(1,1,sent_size+1):narrow(2,1,1),
40                                                                 compute_logscore
                                                                      ,
                                                                   model
                                                                      ,
                                                                   nclass
                                                                   ))
41 --              print('here4')
42
43
44              for ii = 1, sent_size+1 do
45                  grad_pos:zero()
46                  if high_score_seq[ii] ~= gold_sequence[ii] then
47                      -- WARNING: Need to call backward right after the
                           forward with the same input to compute correct
                           gradients
48
49                      -- Use of a single gradient (grad_pos) with a
                           penalization on the wrong class predicted (1)
50                      -- and a valorisation (-1) on the correct class to
                           predict
51                      -- We treat here only the transition after the
                           error
52                      model:forward({inputs_batch:narrow(1,ii,1):narrow
                           (2,1,1),inputs_batch:narrow(1,ii,1):narrow
                           (2,2,9)})
53                      grad_pos[gold_sequence[ii]] = -1
54                      grad_pos[high_score_seq[ii]] = 1
55                      model:backward({inputs_batch:narrow(1,ii,1):narrow
                           (2,1,1),inputs_batch:narrow(1,ii,1):narrow
                           (2,2,9)}, grad_pos:view(1,9))
56
57
58                  end
59              end
60 --              print('here7')
```

```
61              model:updateParameters(eta)
62
63          end
64
65          print('Epoch '..i..': '..timer:time().real)
66          -- Print the f-score on a the first 1000 words to follow the
               improvement of the model
67          cl = viterbi(observations, compute_logscore, model, 9)
68          print (f_score(cl, true_classes))
69
70      end
71  end
72
73  function train_model2(train_input, sent, train_output, model, din,
        nclass, eta, nEpochs, obs_val, true_val, f_score)
74      -- Train the model with the structured perceptron approach
75      -- V2: treating the two edges, the one leading to the error and the
76      -- one leaving the error.
77
78      val_res = torch.zeros(nEpochs,3)
79      -- Memory allocation
80      inputs_batch = torch.DoubleTensor(100, din)
81      gold_sequence = torch.DoubleTensor(100)
82      high_score_seq = torch.DoubleTensor(100)
83      grad_pos = torch.zeros(9)
84      grad_neg = torch.zeros(9)
85      one_hot_true = torch.zeros(1,9)
86      one_hot_false = torch.zeros(1,9)
87
88      for i = 1, nEpochs do
89          -- timing the epoch
90          timer = torch.Timer()
91
92          -- mini batch loop
93          for t = 2, sent:size(1)-1 do
94              -- Mini batch data
95              sent_size = sent[{t,2}]
96  --              print('here1')
97
98              inputs_batch:narrow(1,1,sent_size+1):copy(train_input:
                    narrow(1,sent[{t,1}]-1,sent_size+1))
99  --              print('here2')
100
101             gold_sequence:narrow(1,1,sent_size+1):copy(train_output:
                    narrow(1,sent[{t,1}]-1,sent_size+1))
```

```
102 --              print('here3')
103
104             -- reset gradients
105             model:zeroGradParameters()
106             --gradParameters:zero()
107
108             -- Forward pass on a batch subsequence:
109             high_score_seq:narrow(1,1,sent_size+1):copy(viterbi(
                    inputs_batch:narrow(1,1,sent_size+1):narrow(2,1,1),
110                                                                 compute_logscore
                                                                        ,
                                                                    model
                                                                        ,
                                                                    nclass
                                                                    ))
111 --              print('here4')
112
113             previous_error = false
114
115             for ii = 1, sent_size+1 do
116
117                 grad_neg:zero()
118                 grad_pos:zero()
119
120                 if high_score_seq[ii] ~= gold_sequence[ii] and not
                        previous_error then
121                     -- WARNING: Need to call backward right after the
                            forward with the same input to compute correct
                            gradients
122
123                     -- Use of a single gradient (grad_pos) with a
                            penalization on the wrong class predicted (1)
124                     -- and a valorisation (-1) on the correct class to
                            predict
125
126                     model:forward({inputs_batch:narrow(1,ii,1):narrow
                            (2,1,1),inputs_batch:narrow(1,ii,1):narrow
                            (2,2,9)})
127                     grad_pos[gold_sequence[ii]] = -1
128                     grad_pos[high_score_seq[ii]] = 1
129                     model:backward({inputs_batch:narrow(1,ii,1):narrow
                            (2,1,1),inputs_batch:narrow(1,ii,1):narrow
                            (2,2,9)}, grad_pos:view(1,9))
130
131                     grad_neg:zero()
```

```
132                     grad_pos:zero()
133                     if ii ~= (sent_size + 1) then
134                         one_hot_true:zero()
135                         one_hot_true[1][gold_sequence[ii]] = 1
136                         model:forward({inputs_batch:narrow(1,ii+1,1):
                                narrow(2,1,1),one_hot_true})
137                         grad_neg[gold_sequence[ii+1]] = -1
138                         model:backward({inputs_batch:narrow(1,ii+1,1):
                                narrow(2,1,1),one_hot_true}, grad_neg:view
                                (1,9) )
139
140                         one_hot_false:zero()
141                         one_hot_false[1][high_score_seq[ii]] = 1
142                         model:forward({inputs_batch:narrow(1,ii+1,1):
                                narrow(2,1,1),one_hot_false})
143                         grad_pos[gold_sequence[ii+1]] = 1
144                         model:backward({inputs_batch:narrow(1,ii+1,1):
                                narrow(2,1,1),one_hot_false}, grad_pos:view
                                (1,9) )
145                     end
146
147                     previous_error = true
148
149                 elseif high_score_seq[ii] ~= gold_sequence[ii] and
                        previous_error then
150
151                     if ii ~= sent_size + 1 then
152                         one_hot_true:zero()
153                         one_hot_true[1][gold_sequence[ii]] = 1
154                         model:forward({inputs_batch:narrow(1,ii+1,1):
                                narrow(2,1,1),one_hot_true})
155                         grad_neg[gold_sequence[ii+1]] = -1
156                         model:backward({inputs_batch:narrow(1,ii+1,1):
                                narrow(2,1,1),one_hot_true}, grad_neg:view
                                (1,9) )
157
158                         one_hot_false:zero()
159                         one_hot_false[1][high_score_seq[ii]] = 1
160                         model:forward({inputs_batch:narrow(1,ii+1,1):
                                narrow(2,1,1),one_hot_false})
161                         grad_pos[gold_sequence[ii+1]] = 1
162                         model:backward({inputs_batch:narrow(1,ii+1,1):
                                narrow(2,1,1),one_hot_false}, grad_pos:view
                                (1,9) )
163                     end
```

```
164
165                     previous_error = true
166
167                 else
168                     previous_error = false
169                 end
170             end
171 --              print('here7')
172             model:updateParameters(eta)
173
174         end
175
176         print('Epoch '..i..': '..timer:time().real)
177         cl = viterbi(obs_val, compute_logscore, model, 9)
178         val_res[i][1], val_res[i][2], val_res[i][3]  = f_score(cl,
              true_val)
179         print('f-score: '.. val_res[i][1])
180
181     end
182     return val_res
183 end
```

Helper:

```
1 -- function to evaluate the predicted sequence
2 -- need to compute precision and recall (class 1 stands for negative
     class)
3 function compute_score(predicted_classes, true_classes)
4     print('here')
5     local n = predicted_classes:size(1)
6     local right_pred = 0
7     local positive_true = 0
8     local positive_pred = 0
9     for i=1,n do
10        if predicted_classes[i] > 1 then
11            positive_pred = positive_pred + 1
12        end
13        if true_classes[i] > 1 then
14            positive_true = positive_true + 1
15        end
16        if (true_classes[i] == predicted_classes[i]) and true_classes[i
              ] > 1 then
17            right_pred = right_pred + 1
18        end
19    end
20    local precision = right_pred/positive_pred
```

```
21      local recall = right_pred/positive_true
22      return precision, recall
23  end
24
25  function f_score(precision, recall)
26      return 2*precision*recall/(precision+recall)
27  end
```