Học máy với Python

Session 4

Tổng quan

- Word2vec
- Recurrent Neural Networks (RNNs)
- Triển khai RNNs

Word2vec

- Xét câu sau:

  "It is a heavy rain today, that makes people prefer staying at home over going out."

- people được gọi là **center word**
- {that, makes, prefer, staying} được gọi là **context words** của people
- Một cặp (**center word, context words**) được gọi là một

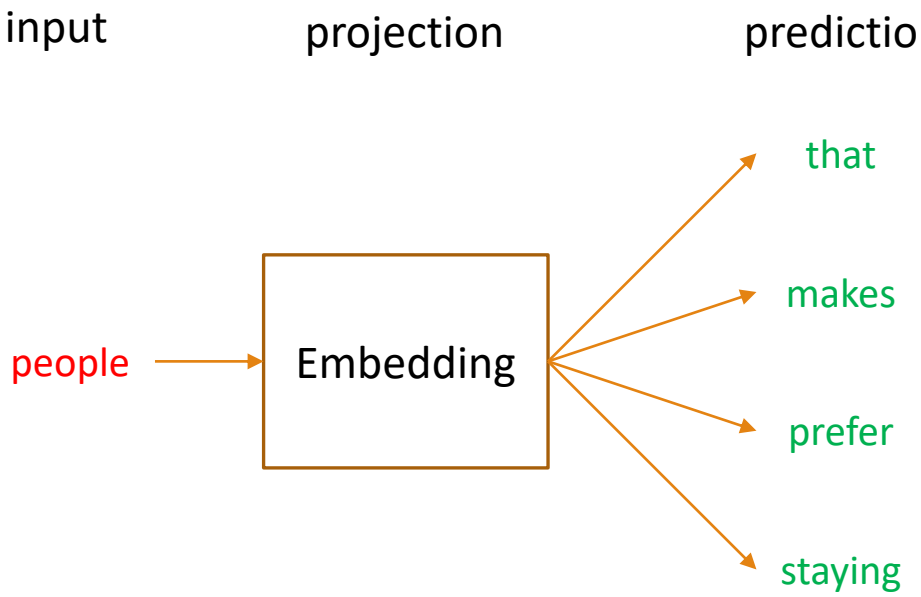
"**skip-gram**"

Word2vec

- Để thu được biểu diễn word2vec của từ, có 2 mô hình:
    1. Skip-Gram
    2. CBOW (Continuous Bag-of-Word Model)

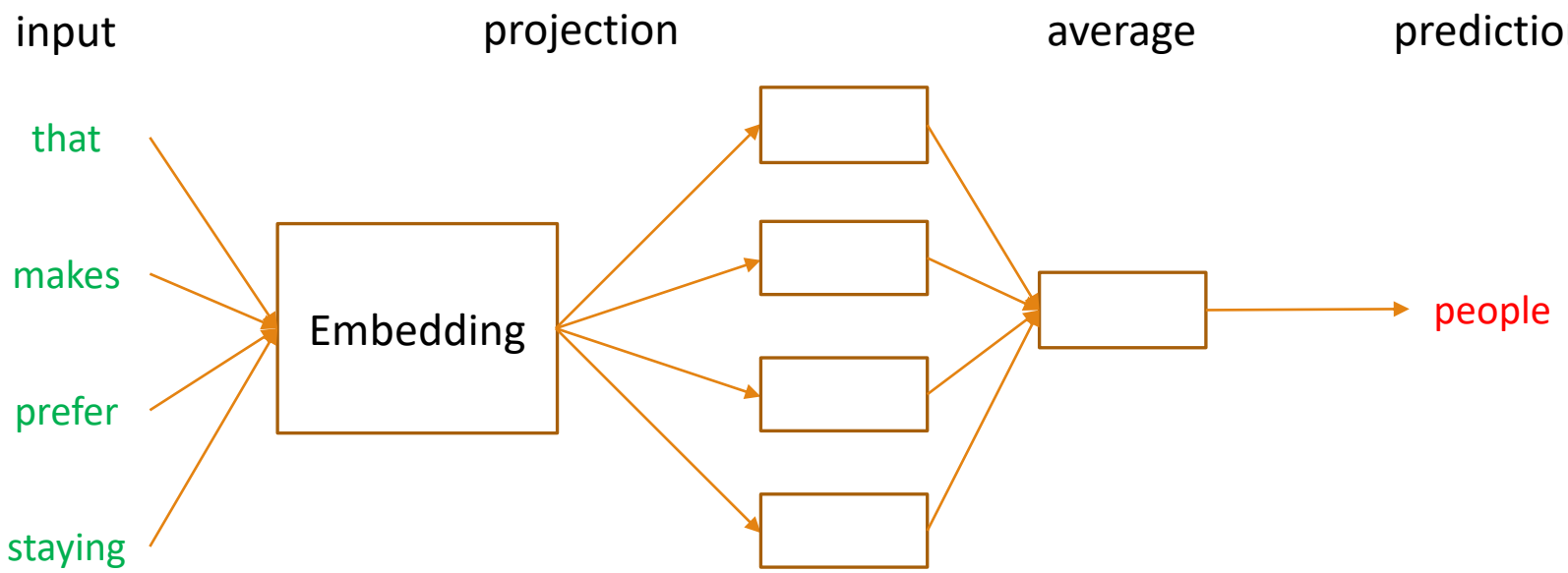
1. Skip-Gram

- Sử dụng **center word** làm input và **context words** làm target



2. CBOW

- Sử dụng **context words** làm input và **center word** làm target



Word2vec

- Sau khi huấn luyện, thu được ma trận Word Embedding, mỗi

Hàng là một vector biểu diễn cho một từ.

- Lưu ý: ma trận Word Embedding cũng thay đổi khi training
- Word Embedding thường là tầng đầu tiên trong rất nhiều mô hình Deeplearning hiện nay.
- Xem chi tiết tại:

[1] https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

[2] https://arxiv.org/pdf/1411.2738.pdf

Recurrent Neural Networks

- Chuyên dùng cho dữ liệu dạng chuỗi.
- Ví dụ: “It is a heavy rain today”

⇒x1: “it” , x2: “is”, x3: “a”, x4: “heavy”, x5: “rain”, x6: “today”

⇒**Chuỗi**: x1 -> x2 -> x3 -> x4 -> x5 -> x6

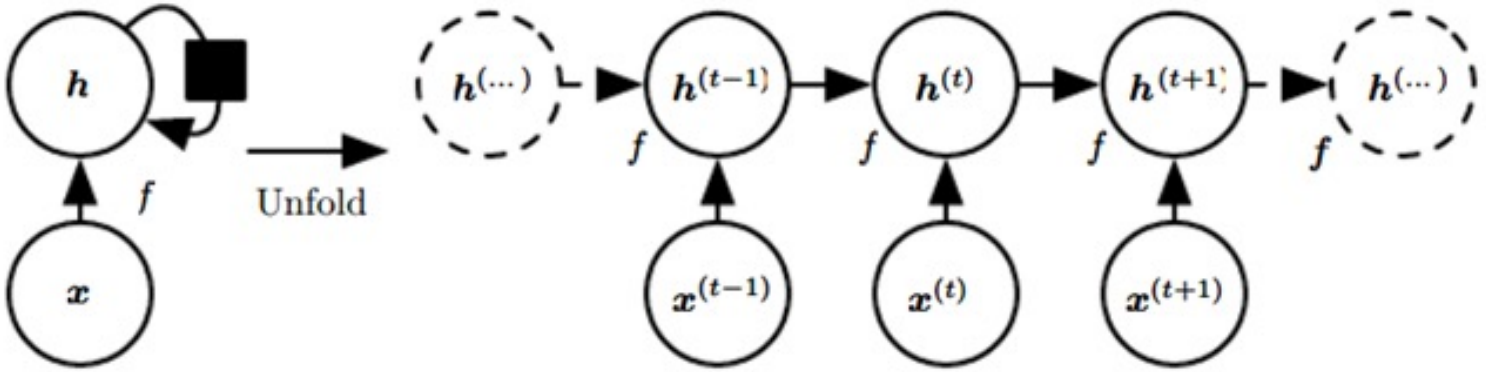
- Mỗi phần tử của chuỗi nằm ở một bước thời gian (**time step**) khác nhau và có quan hệ về mặt thứ tự với nhau
- Mô hình hóa quan hệ thứ tự giữa các phần tử trong chuỗi

=> Recurrent Neural Networks ra đời.

Recurrent Neural Networks

- Cấu trúc chung của RNNs:

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$



Recurrent Neural Networks

- Một cách mô hình hóa hàm f

⇒Long Short-Term Memory (LSTM)

Long Short-Term Memory

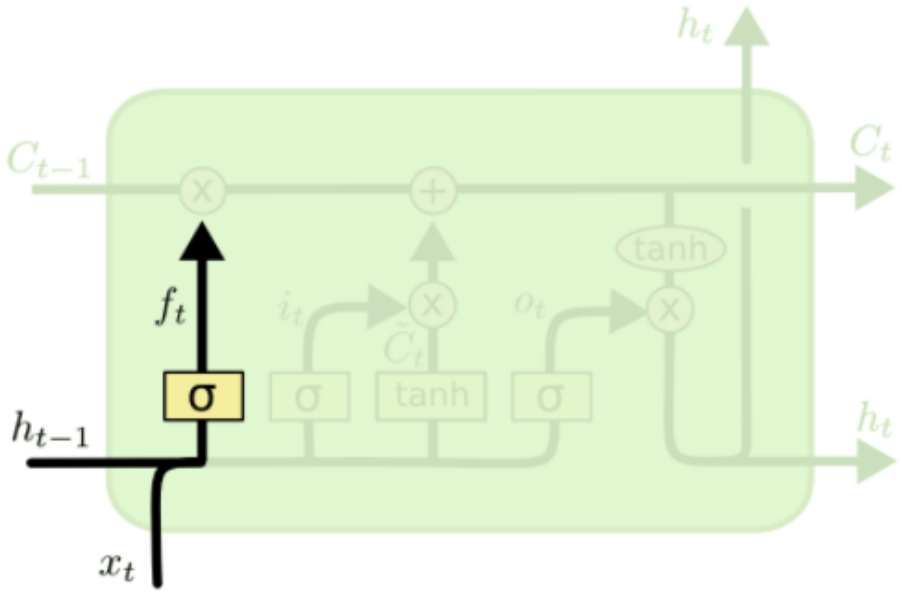
- Bao gồm các cổng để điều khiển luồng thông tin đi qua qua LSTM
- Thông tin được lưu trữ trong Memory cell (MC) C_t
- LSTM cell gồm 3 cổng:
  - > Forget gate: "quên" thông-tin-trong-MC-từ-timestep-trước, C_{t-1}
  - > Input gate: "lưu trữ" vào MC thông-tin-từ-timestep-hiện-tại, $\widetilde{C}_t$
  - > Output gate: "trích xuất" thông tin từ MC để đưa ra h_t

Long Short-Term Memory

- **Forget gate**:

  > thông-tin-trong-MC-từ-timestep-trước C_{t-1}

  > "quên" C_{t-1} nhiều hay ít ?



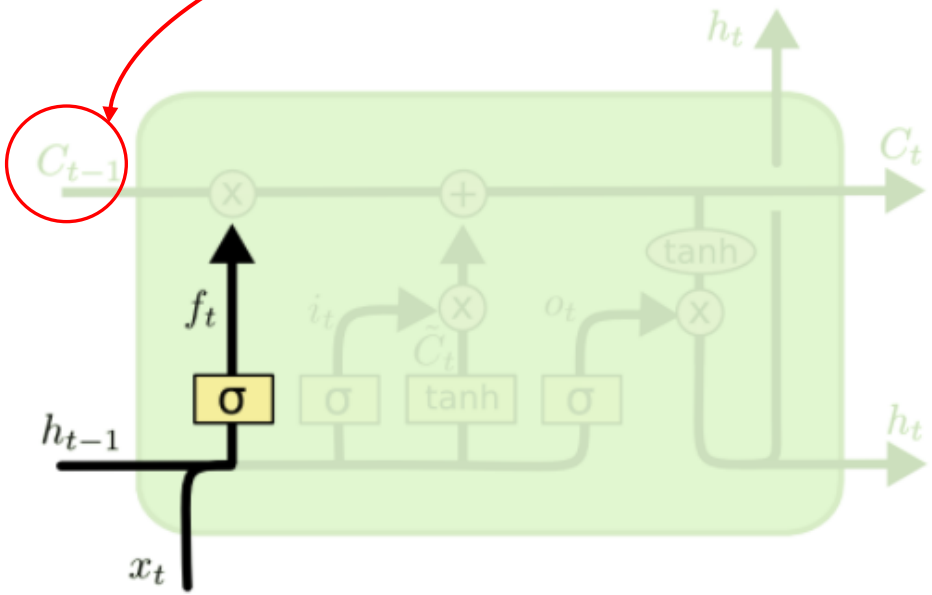
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Long Short-Term Memory

- **Forget gate**:

> thông-tin-trong-MC-từ-timestep-trước C_{t-1}

> "quên" C_{t-1} nhiều hay ít ?



$f_t \in (0,1)$

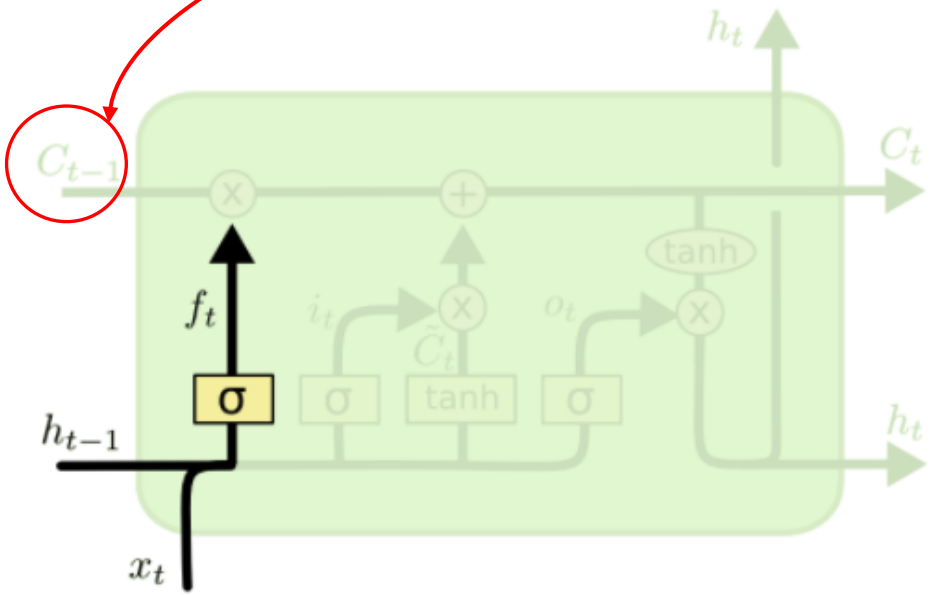
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Long Short-Term Memory

- **Forget gate**:

> thông-tin-trong-MC-từ-timestep-trước C_{t-1}

> "quên" C_{t-1} nhiều hay ít ?



$f_t \in (0,1)$

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Thông tin đi qua forget gate:

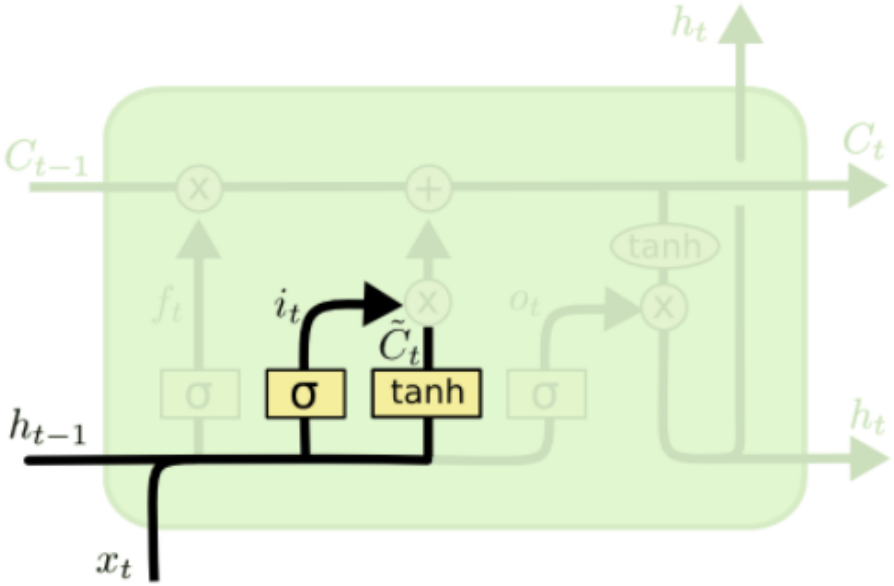
$$f_t \; * C_{t-1}$$

Long Short-Term Memory

- Input gate:

> dữ liệu từ timestep hiện tại x_t

> thông-tin-từ-x_t-muốn-cho-vào-MC là $\tilde{C}_t$, lấy bao nhiêu ?



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

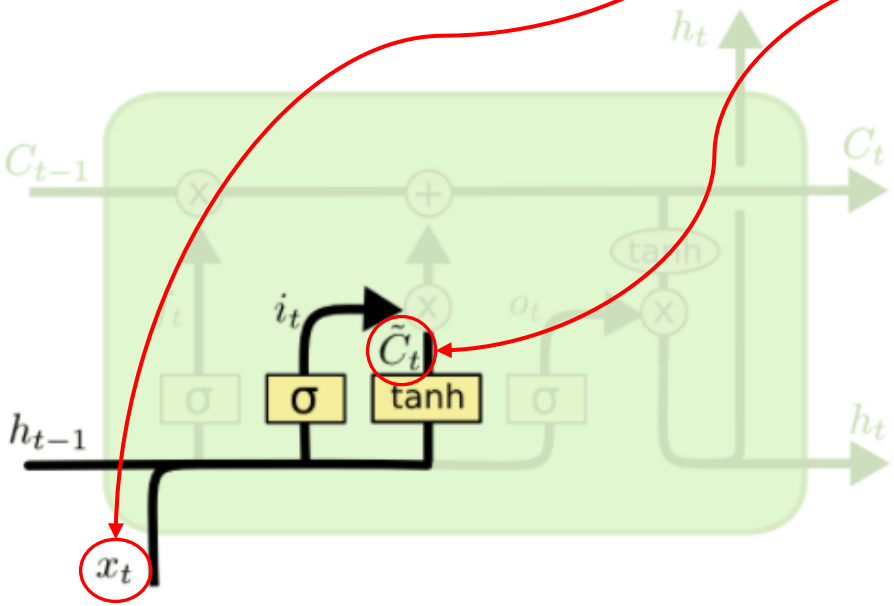
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Long Short-Term Memory

- **Input gate**:

> dữ liệu từ timestep hiện tại x_t

> thông-tin-từ-x_t-muốn-cho-vào-MC là $\tilde{C}_t$, lấy bao nhiêu ?



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

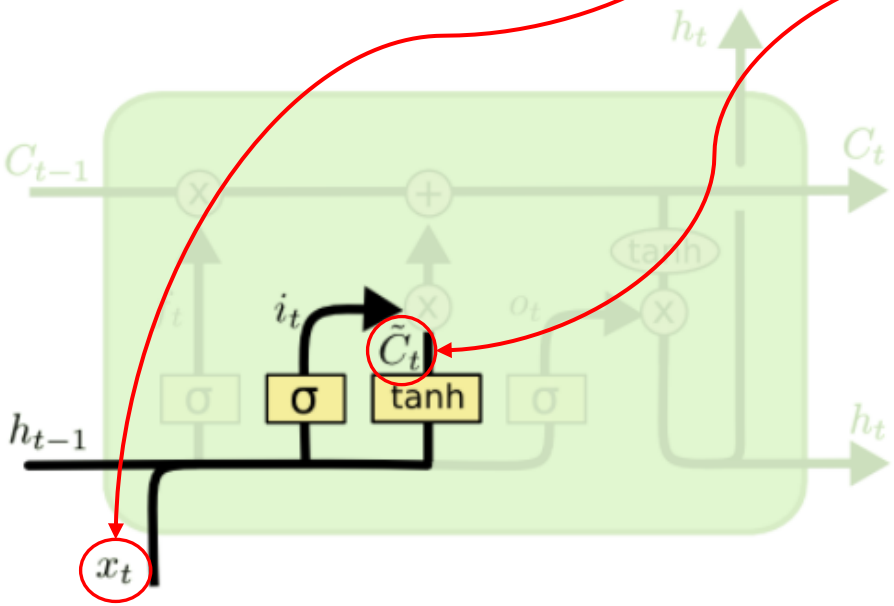
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Long Short-Term Memory

- **Input gate**:

> dữ liệu từ timestep hiện tại x_t

> thông-tin-từ-x_t-muốn-cho-vào-MC là $\tilde{C}_t$, lấy bao nhiêu ?



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Thông tin đi qua input gate:

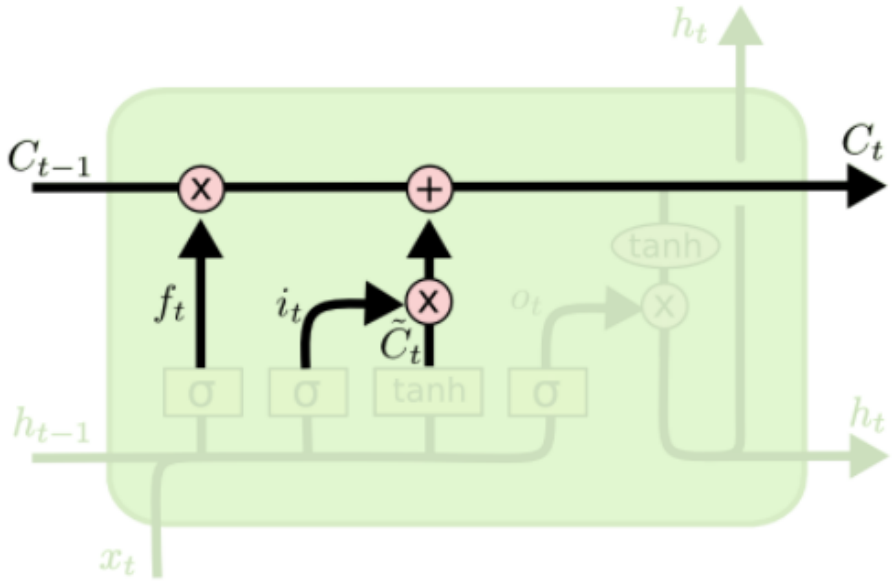
$$i_t \; * \tilde{C}_t$$

Long Short-Term Memory

- Tổng hợp thông tin:

  > Thông tin đi qua forget gate: $f_t * C_{t-1}$

  > Thông tin đi qua input gate: $i_t * \tilde{C}_t$



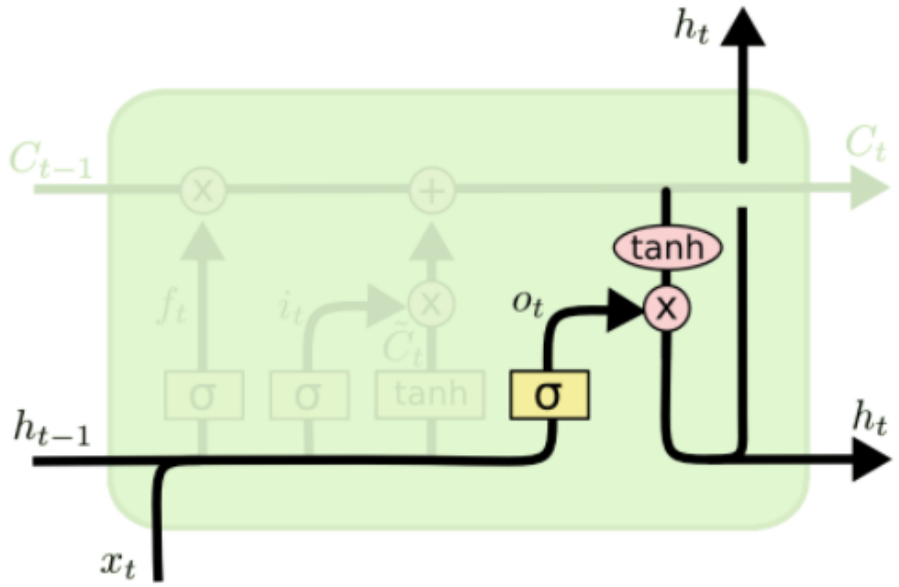
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Long Short-Term Memory

- **Output gate**:

  > "trích xuất" thông tin từ MC, h_t

  > thông tin được trích xuất trước đó, h_{t-1}



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

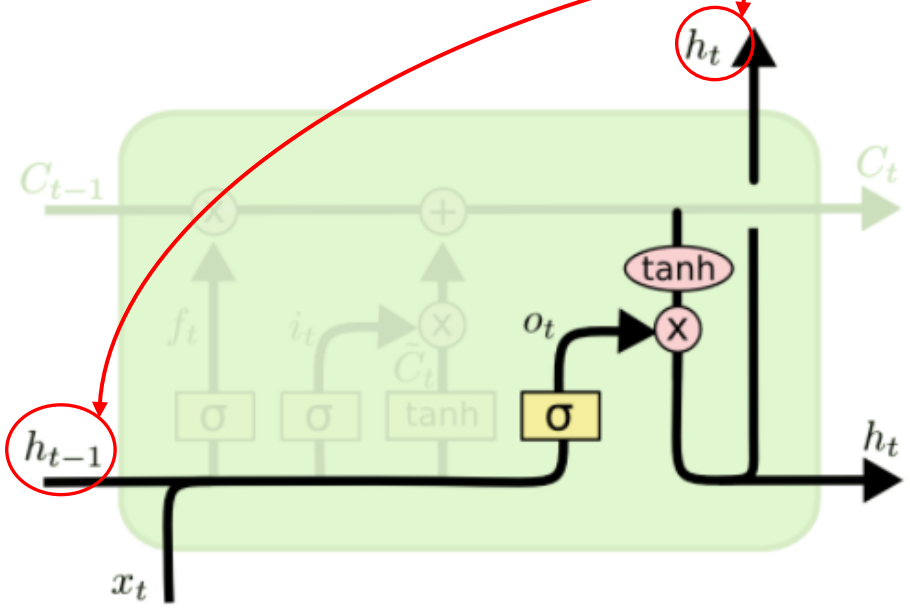
$$h_t = o_t * \tanh\left(C_t\right)$$

Long Short-Term Memory

- **Output gate**:

  > "trích xuất" thông tin từ MC, h_t

  > thông tin được trích xuất trước đó, h_{t-1}



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Long Short-Term Memory

- Nhận xét:

> Memory cell có tác dụng là vật chứa thông tin nội bộ (C_t), được tích lũy qua các timesteps cho tới thời điểm t

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

> Output gate đưa ra h_t , điều khiển việc trích xuất thông tin từ Memory cell

$$h_t = o_t * \tanh(C_t)$$

Long Short-Term Memory

- Nhận xét:

> Memory cell có tác dụng là vật chứa thông tin nội bộ (C_t),

được tích lũy qua các timesteps cho tới thời điểm t

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

> Output gate đưa ra h_t , điều khiển việc trích xuất

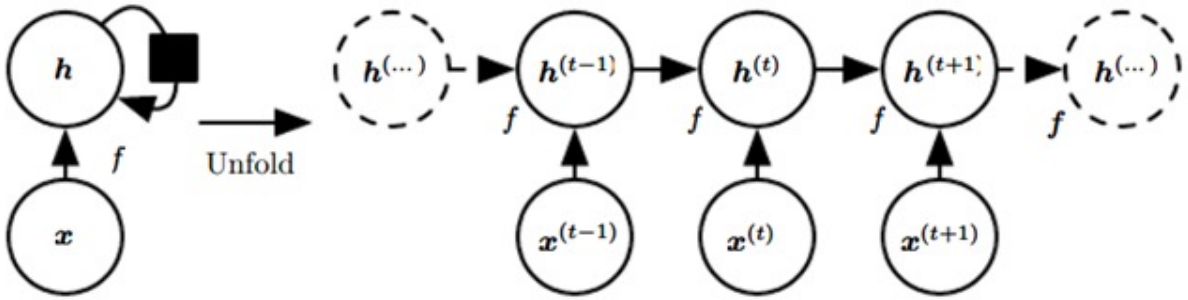
thông tin từ Memory cell

$$h_t = o_t * \tanh(C_t)$$

> C và h là 2 vector có cùng kích thước -> LSTM cell size (lstm size)

Long Short-Term Memory

- Cách sử dụng LSTM:



> cần có trạng thái khởi đầu $h^{(0)}$

> trong LSTM, memory cell cũng cần có trạng thái khởi $C^{(0)}$

Long Short-Term Memory

- Cách sử dụng LSTM: việc input đi qua LSTM giống như một vòng lặp:

```
$ init_state = {c, h }
$ outputs = []
$ for timestep in range(max_timestep):
$     new_c, new_h = LSTM(c, h)
$     outputs.append( (new_c, new_h) )
$     c, h = new_c, new_h
```

=> Cần giới hạn giá trị max_timestep

Long Short-Term Memory

- Xem thêm tại:

[1] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

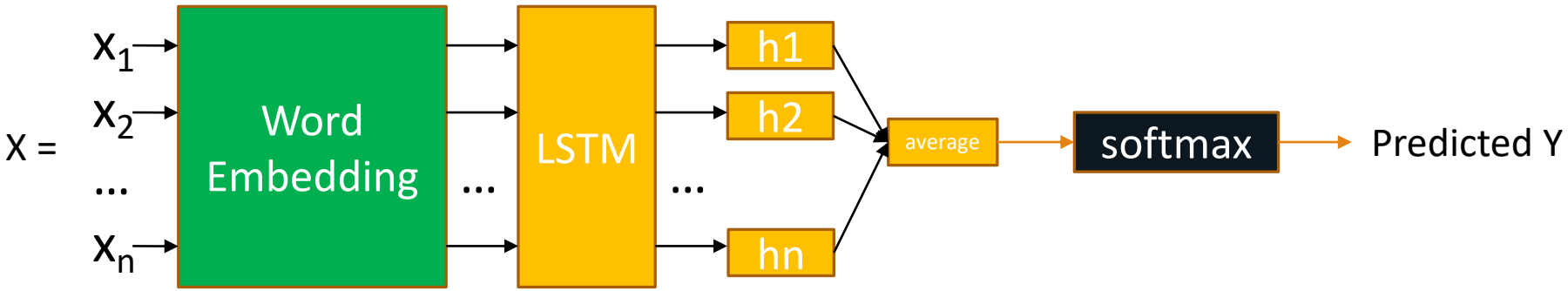
[2] http://www.bioinf.jku.at/publications/older/2604.pdf

Triển khai RNNs

- Nội dung chính:
  - > Kiến trúc mạng
  - > Triển khai

Triển khai RNNs

- Kiến trúc mạng:



Triển khai RNNs

- Triển khai:
    1. Xử lý dữ liệu
    2. Xây dựng mô hình

1. Xử lý dữ liệu

- Quy trình:
  - > Xây dựng từ điển
  - > Encode dữ liệu

1. Xử lý dữ liệu

- Xây dựng từ điển:

  > Quét qua toàn bộ văn bản và thu thập các từ xuất hiện trong tập dữ liệu

  > Loại bỏ đi những từ ít xuất hiện

1. Xử lý dữ liệu

- Xây dựng từ điển và thu thập dữ liệu từ files:

```
17  def gen_data_and_vocab():
18      def collect_data_from(parent_path, newsgroup_list, word_count=None):...
40
41      word_count = defaultdict(int)
42
43      path = '../datasets/20news-bydate/'
44      parts = [path + dir_name + '/' for dir_name in listdir(path)
45               if not isfile(path + dir_name)]
46
47      train_path, test_path = (parts[0], parts[1]) \
48          if 'train' in parts[0] else (parts[1], parts[0])
49
50      newsgroup_list = [newsgroup for newsgroup in listdir(train_path)]
51      newsgroup_list.sort()
```

1. Xử lý dữ liệu

- Xây dựng từ điển và thu thập dữ liệu từ files:

```
53      train_data = collect_data_from(
54          parent_path=train_path,
55          newsgroup_list=newsgroup_list,
56          word_count=word_count
57      )
58      vocab = [word for word, freq in
59               zip(word_count.keys(), word_count.values()) if freq > 10]
60      vocab.sort()
61      with open('../datasets/w2v/vocab-raw.txt', 'w') as f:
62          f.write('\n'.join(vocab))
```

1. Xử lý dữ liệu

- Xây dựng từ điển và thu thập dữ liệu từ files:

```
64      test_data = collect_data_from(
65          parent_path=test_path,
66          newsgroup_list=newsgroup_list
67      )
68
69      with open('../datasets/w2v/20news-train-raw.txt', 'w') as f:
70          f.write('\n'.join(train_data))
71
72      with open('../datasets/w2v/20news-test-raw.txt', 'w') as f:
73          f.write('\n'.join(test_data))
```

1. Xử lý dữ liệu

- Xây dựng từ điển …: hàm collect_data_from

```
18      def collect_data_from(parent_path, newsgroup_list, word_count=None):
19          data = []
20          for group_id, newsgroup in enumerate(newsgroup_list):
21              dir_path = parent_path + '/' + newsgroup + '/'
22
23              files = [(filename, dir_path + filename)
24                       for filename in listdir(dir_path) if
25                       isfile(dir_path + filename)]
26              files.sort()
27              label = group_id
28              print 'Processing: {}-{}'.format(group_id, newsgroup)
29
30              for filename, filepath in files:...
40          return data
```

1. Xử lý dữ liệu

- Xây dựng từ điển …: hàm collect_data_from: line 30

```
for filename, filepath in files:
    with open(filepath) as f:
        text = f.read().lower()
        words = re.split('\W+', text)
        if word_count is not None:
            for word in words:
                word_count[word] += 1
        content = ' '.join(words)
        assert len(content.splitlines()) == 1
        data.append(str(label) + '<fff>'
                    + filename + '<fff>' + content)
```

- Lưu ý: chỉ xây dựng từ điển từ train data

1. Xử lý dữ liệu

- Encode dữ liệu:

  > Đánh ID cho các từ trong từ điển: 2, 3, 4, …., V + 2

  (V là kích thước từ điển)

  > Dành riêng 2 ID đặc biệt cho:

    * unknown word: từ không xuất hiện trong từ điển

    * padding word: từ “rỗng” được thêm vào mỗi văn bản

  > Mỗi văn bản được mã hóa bằng cách nhận biết sự có mặt của các từ và thay thế tương ứng bởi các ID của chúng

1. Xử lý dữ liệu

- Encode dữ liệu:

```
96  def encode_data(data_path, vocab_path):
97      with open(vocab_path) as f:
98          vocab = dict([(word, word_ID + 2)
99                        for word_ID, word in enumerate(f.read().splitlines())])
100     with open(data_path) as f:
101         documents = [(line.split('<fff>')[0], line.split('<fff>')[1], line.split('<fff>')[2])
102                      for line in f.read().splitlines()]
103     encoded_data = []
104     for document in documents:...
123
124     dir_name = '/'.join(data_path.split('/')[:-1])
125     file_name = '-'.join(data_path.split('/')[-1].split('-')[:-1]) + '-encoded.txt'
126     with open(dir_name + '/' + file_name, 'w') as f:
127         f.write('\n'.join(encoded_data))
```

1. Xử lý dữ liệu

- Encode dữ liệu: line 104

> Lấy thông tin từ các văn bản

```
104         for document in documents:
105             label, doc_id, text = document
106             words = text.split()[:MAX_DOC_LENGTH]
107             sentence_length = len(words)
```

> ví dụ chọn MAX_SENTENCE_LENGTH = 500

> Lưu ý: không lowercase, không loại bỏ stopwords, … như khi tính tf-idf

1. Xử lý dữ liệu

- Encode dữ liệu:

> Thay thế các từ bằng ID của nó

```
109         encoded_text = []
110         for word in words:
111             if word in vocab:
112                 encoded_text.append(str(vocab[word]))
113             else:
114                 encoded_text.append(str(unknown_ID))
115 
116         if len(words) < MAX_DOC_LENGTH:...
120 
121         encoded_data.append(str(label) + '<fff>' + str(doc_id) + '<fff>' +
122                             str(sentence_length) + '<fff>' + ' '.join(encoded_text))
```

1. Xử lý dữ liệu

- Encode dữ liệu:

  > Thêm các từ “rỗng” padding words

```
116         if len(words) < MAX_DOC_LENGTH:
117             num_paddding = MAX_DOC_LENGTH - len(words)
118             for _ in range(num_paddding):
119                 encoded_text.append(str(padding_ID))
```

1. Xử lý dữ liệu

- Encode dữ liệu:

> Ghi dữ liệu đã encoded ra file

```
124         dir_name = '/'.join(data_path.split('/')[:-1])
125         file_name = '-'.join(data_path.split('/')[-1].split('-')[:-1]) + '-encoded.txt'
126         with open(dir_name + '/' + file_name, 'w') as f:
127             f.write('\n'.join(encoded_data))
```

1. Xử lý dữ liệu

- Encode dữ liệu:

  > Ghi dữ liệu đã encoded ra file: 20news-train-encoded.txt

```
1  0<fff>49960<fff>500<fff>7541 10894 10894 10784 4333 17490 16285 2063 2
2  0<fff>51060<fff>500<fff>7541 10894 10894 10784 4333 17490 16285 2063 2
3  0<fff>51119<fff>500<fff>7541 8766 5348 14853 17332 3468 5361 3025 1472
4  0<fff>51120<fff>261<fff>7541 10894 10894 10784 4333 17490 16285 14037
5  0<fff>51121<fff>123<fff>7541 16243 18255 8785 4417 14651 16243 16285 1
6  0<fff>51122<fff>500<fff>7541 8766 5348 14853 17332 3468 5361 3025 1472
7  0<fff>51123<fff>98<fff>7541 9814 3888 3685 6318 9814 2028 15040 16285
8  0<fff>51124<fff>254<fff>7541 8766 5348 14853 17332 3468 5361 3025 1472
9  0<fff>51125<fff>469<fff>7541 9814 3888 3685 6318 9814 2028 15040 16285
```

2. Xây dựng mô hình

- Xây dựng computation graph qua class RNN
- Mở một phiên làm việc (session), truyền dữ liệu (thông qua Class DataReader) và chạy mô hình

2. Xây dựng mô hình

- class RNN:

```
195  class RNN:
196      def __init__(self,
197                   vocab_size,
198                   embedding_size,
199                   lstm_size,
200                   pretrained_w2v_path,
201                   batch_size
202                   ):...
214
215      def embedding_layer(self, indices):...
230
231      def LSTM_layer(self, embeddings):...
268
269      def build_graph(self):...
303
304      def trainer(self, loss, learning_rate):...
```

2. Xây dựng mô hình

- class RNN: hàm init

```
196     def __init__(self,
197                  vocab_size,
198                  embedding_size,
199                  lstm_size,
200                  batch_size
201                  ):
202         self._vocab_size = vocab_size
203         self._embedding_size = embedding_size
204         self._lstm_size = lstm_size
205         self._batch_size = batch_size
206 
207         self._data = tf.placeholder(tf.int32, shape=[batch_size, MAX_DOC_LENGTH])
208         self._labels = tf.placeholder(tf.int32, shape=[batch_size, ])
209         self._sentence_lengths = tf.placeholder(tf.int32, shape=[batch_size, ])
210         self._final_tokens = tf.placeholder(tf.int32,
211                                             shape=[batch_size, ])
```

2. Xây dựng mô hình

- class RNN: hàm build_graph

```
267     def build_graph(self):
268         embeddings = self.embedding_layer(self._data)
269         lstm_outputs = self.LSTM_layer(embeddings)
270 
271         weights = tf.get_variable(
272             name='final_layer_weights',
273             shape=(self._lstm_size, NUM_CLASSES),
274             initializer=tf.random_normal_initializer(seed=2018),
275         )
276         biases = tf.get_variable(
277             name='final_layer_biases',
278             shape=(NUM_CLASSES),
279             initializer=tf.random_normal_initializer(seed=2018)
280         )
281 
282         logits = tf.matmul(lstm_outputs, weights) + biases
```

2. Xây dựng mô hình

- class RNN: hàm build_graph

```
282         logits = tf.matmul(lstm_outputs, weights) + biases
283
284         labels_one_hot = tf.one_hot(
285             indices=self._labels,
286             depth=NUM_CLASSES,
287             dtype=tf.float32
288         )
289
290         loss = tf.nn.softmax_cross_entropy_with_logits(
291             labels=labels_one_hot,
292             logits=logits
293         )
294         loss = tf.reduce_mean(loss)
295
296         probs = tf.nn.softmax(logits)
297         predicted_labels = tf.argmax(probs, axis=1)
298         predicted_labels = tf.squeeze(predicted_labels)
```

2. Xây dựng mô hình

- class RNN: hàm embedding_layer

```
213     def embedding_layer(self, indices):
214         pretrained_vectors = []
215         pretrained_vectors.append(np.zeros(self._embedding_size))
216         np.random.seed(2018)
217         for _ in range(self._vocab_size + 1):
218             pretrained_vectors.append(np.random.normal(loc=0., scale=1., size=self._embedding_size))
219 
220         pretrained_vectors = np.array(pretrained_vectors)
221 
222         self._embedding_matrix = tf.get_variable(
223             name='embedding',
224             shape=(self._vocab_size + 2, self._embedding_size),
225             initializer=tf.constant_initializer(pretrained_vectors)
226         )
227         return tf.nn.embedding_layer(self._embedding_matrix, indices)
```

2. Xây dựng mô hình

- class RNN: hàm LSTM_layer

```
229     def LSTM_layer(self, embeddings):
230         lstm_cell = tf.contrib.rnn.BasicLSTMCell(self._lstm_size)
231         zero_state = tf.zeros(shape=(self._batch_size, self._lstm_size))
232         initial_state = tf.contrib.rnn.LSTMStateTuple(zero_state, zero_state)
233 
234         lstm_inputs = tf.unstack(
235             tf.transpose(embeddings, perm=[1, 0, 2])
236         )
237         lstm_outputs, last_state = tf.nn.static_rnn(
238             cell=lstm_cell,
239             inputs=lstm_inputs,
240             initial_state=initial_state,
241             sequence_length=self._sentence_lengths
242         )  # a length-500 list of [num_docs, lstm_size]
```

[1] BasicLSTMCell: https://github.com/tensorflow/tensorflow/blob/r1.9/tensorflow/python/ops/rnn_cell_impl.py
[2] static_rnn : https://github.com/tensorflow/tensorflow/blob/r1.9/tensorflow/python/ops/rnn.py

2. Xây dựng mô hình

- class RNN: hàm LSTM_layer

```
244             lstm_outputs = tf.unstack(
245                 tf.transpose(lstm_outputs, perm=[1, 0, 2]))
246             lstm_outputs = tf.concat(
247                 lstm_outputs,
248                 axis=0
249             )  # [num docs * MAX_SENT_LENGTH, lstm_size]
250 
251             # self._mask : [num docs * MAX_SENT_LENGTH, ]
252             mask = tf.sequence_mask(
253                 lengths=self._sentence_lengths,
254                 maxlen=MAX_DOC_LENGTH,
255                 dtype=tf.float32
256             )  # [num_docs, MAX_SENTENCE_LENGTH]
257             mask = tf.concat(tf.unstack(mask, axis=0), axis=0)
258             mask = tf.expand_dims(mask, -1)
```

2. Xây dựng mô hình

- class RNN: hàm LSTM_layer

```
261             lstm_outputs = mask * lstm_outputs
262             lstm_outputs_split = tf.split(lstm_outputs, num_or_size_splits=self._batch_size)
263             lstm_outputs_sum = tf.reduce_sum(lstm_outputs_split, axis=1)  # [num_docs, lstm_size]
264             lstm_outputs_average = lstm_outputs_sum / tf.expand_dims(
265                 tf.cast(self._sentence_lengths, tf.float32),
266                 # expand_dims only works with tensor of float type
267                 -1)  # [num_docs, lstm_size]
268             return lstm_outputs_average
```

2. Xây dựng mô hình

- class RNN: hàm trainer

```
305     def trainer(self, loss, learning_rate):
306         train_op = tf.train.AdamOptimizer(learning_rate).minimize(loss)
307         return train_op
```

2. Xây dựng mô hình

- Xây dựng computation graph

```
322  def train_and_evaluate_RNN():
323      with open('../datasets/w2v/vocab_w2v.txt') as f:
324          vocab_size = len(f.read().splitlines())
325
326      tf.set_random_seed(2018)
327      rnn = RNN(
328          vocab_size=vocab_size,
329          embedding_size=300,
330          lstm_size=50,
331          batch_size=50
332      )
333      predicted_labels, loss = rnn.build_graph()
334      train_op = rnn.trainer(loss=loss, learning_rate=0.01)
```

- Lưu ý: giá trị LSTM size và Batch size cần được chọn qua cross-validation

2. Xây dựng mô hình

- Mở một phiên làm việc, đọc, truyện dữ liệu và chạy

```
336     with tf.Session() as sess:
337         train_data_reader = DataReader(
338             data_path='../datasets/w2v/20news-train-encoded.txt',
339             batch_size=50
340         )
341 
342         test_data_reader = DataReader(
343             data_path='../datasets/w2v/20news-test-encoded.txt',
344             batch_size=50
345         )
346         step = 0
347         MAX_STEP = 1000 ** 2
348 
349         sess.run(tf.global_variables_initializer())
```

2. Xây dựng mô hình

- Mở một phiên làm việc, đọc, truyện dữ liệu và chạy

```
349         sess.run(tf.global_variables_initializer())
350         while step < MAX_STEP:
351             next_train_batch = train_data_reader.next_batch()
352             train_data, train_labels, train_sentence_lengths, train_final_tokens = next_train_batch
353             plabels_eval, loss_eval, _ = sess.run(
354                 [predicted_labels, loss, train_op],
355                 feed_dict={
356                     rnn._data: train_data,
357                     rnn._labels: train_labels,
358                     rnn._sentence_lengths: train_sentence_lengths,
359                     rnn._final_tokens: train_final_tokens
360                 }
361             )
362             step += 1
363             if step % 20 == 0:
364                 print 'loss:', loss_eval
```

2. Xây dựng mô hình

- Khi hết một epoch -> đánh giá trên test data

```
365             if train_data_reader._current_part == 0:
366                 num_true_preds = 0
367                 while True:
368                     next_test_batch = test_data_reader.next_batch()
369                     test_data, test_labels, test_sentence_lengths, test_final_tokens = next_test_batch
370 
371                     test_plabels_eval = sess.run(
372                         predicted_labels,
373                         feed_dict={...}
379                     )
380                     matches = np.equal(test_plabels_eval, test_labels)
381                     num_true_preds += np.sum(matches.astype(float))
382 
383                     if test_data_reader._current_part == 0:
384                         break
385                 print 'Epoch:', train_data_reader._num_epoch
386                 print 'Accuracy on test data:', num_true_preds * 100. / len(test_data_reader._data)
```

Mở rộng

- Tìm hiểu them về Bidirectional RNNs
- Các biến thể của LSTM như: LSTM with peephole connection, Gated Recurrent Unit (GRU), …
- Các chiến lược tối ưu khác: learning rate decay, gradient clipping, Adadelta, …

Thank you