Named Entity Recognition (NER)

January 8, 2020

1 Recognize named entities on Twitter with LSTMs

Recurrent neural networks, in particular Bi-Directional Long Short Memory Networks (Bi-LSTMs), will be used to solve a Named Entity Recognition (NER) problem, extracting named entities from Twitter data.

The data is a corpus containing tweets with NE tags. Every line of a file contains a token (word/punctuation symbol-tag pair, separated by a whitespace. Different tweets are separated by an empty line.

There are three separate dataset files: (1) train data for training the model; (2) validation data for evaluation and hyperparameters tuning; (3) test data for final evaluation of the model.

1.1 Load Twitter Named Entity Recognition corpus

[1]:
```
#function to read corpus and return 2 lists - tokens and tags
def read_data(file_path):
    tokens = []
    tags = []

    tweet_tokens = []
    tweet_tags = []
    for line in open(file_path, encoding='utf-8'):
        line = line.strip()
        if not line:
            if tweet_tokens:
                tokens.append(tweet_tokens)
                tags.append(tweet_tags)
            tweet_tokens = []
            tweet_tags = []
        else:
            token, tag = line.split()
            # Replace all urls with <URL> token
            # Replace all users with <USR> token
            if token.startswith('@'):
                token = '<USR>'
            elif token.startswith('http://') or token.startswith('https://'):
                token = '<URL>'
```

```
            tweet_tokens.append(token)
            tweet_tags.append(tag)

    return tokens, tags
```

[3]:
```
train_tokens, train_tags = read_data('train.txt')
validation_tokens, validation_tags = read_data('validation.txt')
test_tokens, test_tags = read_data('test.txt')
```

[4]:
```
for i in range(3):
    for token, tag in zip(train_tokens[i], train_tags[i]):
        print('%s\t%s' % (token, tag))
    print()
```

```
RT      O
<USR>   O
:       O
Online  O
ticket  O
sales   O
for     O
Ghostland       B-musicartist
Observatory     I-musicartist
extended        O
until   O
6       O
PM      O
EST     O
due     O
to      O
high    O
demand  O
.       O
Get     O
them    O
before  O
they    O
sell    O
out     O
…       O

Apple   B-product
MacBook I-product
Pro     I-product
A1278   I-product
13.3    I-product
```

```
"       I-product
Laptop  I-product
-       I-product
MD101LL/A       I-product
(       O
June    O
,       O
2012    O
)       O
-       O
Full    O
read    O
by      O
eBay    B-company
<URL>   O
<URL>   O

Happy   O
Birthday        O
<USR>   O
!       O
May     O
Allah   B-person
s.w.t   O
bless   O
you     O
with    O
goodness        O
and     O
happiness       O
.       O
```

1.2 Prepare dictionaries

[5]:
```
from collections import defaultdict
```

[6]:
```
#function that takes in tokens or tags list and special tokens
def build_dict(tokens_or_tags, special_tokens):

    # Create dictionary with default value 0
    tok2idx = defaultdict(lambda: 0)
    idx2tok = []

    # Create mappings from tokens (or tags) to indices and vice versa
    # At first, add special tokens (or tags) to the dictionaries - the first␣
↪special token must have index 0
```

```python
    # Mapping tok2idx should contain each token or tag only once; to do so,␣
↪must:
    # 1. extract unique tokens/tags from tokens_or_tags variable not in␣
↪special_tokens
    # 2. index them
    # 3. for each token/tag save index into tok2idx

    for i, token in enumerate(special_tokens):
        tok2idx[token] = i
        idx2tok.append(token)

    nextIndex = len(special_tokens)
    for tokens in tokens_or_tags:
        for token in tokens:
            if token not in tok2idx:
                tok2idx[token] = nextIndex
                idx2tok.append(token)
                nextIndex += 1

    return tok2idx, idx2tok
```

[7]:
```python
#special token <PAD> for padding sentence to same length when create batches of␣
↪sentences
#special token <UNK> is for out of vocab tokens
special_tokens = ['<UNK>', '<PAD>']
special_tags = ['O']

# Create dictionaries
token2idx, idx2token = build_dict(train_tokens + validation_tokens,␣
↪special_tokens)
tag2idx, idx2tag = build_dict(train_tags, special_tags)
```

[8]:
```python
#check same length
len(token2idx) == len(idx2token)
```

[8]: True

[9]:
```python
tag2idx
```

[9]:
```
defaultdict(<function __main__.build_dict.<locals>.<lambda>()>,
            {'O': 0,
             'B-musicartist': 1,
             'I-musicartist': 2,
             'B-product': 3,
             'I-product': 4,
             'B-company': 5,
```

```
             'B-person': 6,
             'B-other': 7,
             'I-other': 8,
             'B-facility': 9,
             'I-facility': 10,
             'B-sportsteam': 11,
             'B-geo-loc': 12,
             'I-geo-loc': 13,
             'I-company': 14,
             'I-person': 15,
             'B-movie': 16,
             'I-movie': 17,
             'B-tvshow': 18,
             'I-tvshow': 19,
             'I-sportsteam': 20})
```

[10]:
```
idx2tag
```

[10]:
```
['O',
 'B-musicartist',
 'I-musicartist',
 'B-product',
 'I-product',
 'B-company',
 'B-person',
 'B-other',
 'I-other',
 'B-facility',
 'I-facility',
 'B-sportsteam',
 'B-geo-loc',
 'I-geo-loc',
 'I-company',
 'I-person',
 'B-movie',
 'I-movie',
 'B-tvshow',
 'I-tvshow',
 'I-sportsteam']
```

[11]:
```
#functions to help create mapping between tokens and ids for a sentence
def words2idxs(tokens_list):
    return [token2idx[word] for word in tokens_list]

def tags2idxs(tags_list):
    return [tag2idx[tag] for tag in tags_list]
```

```python
def idxs2words(idxs):
    return [idx2token[idx] for idx in idxs]

def idxs2tags(idxs):
    return [idx2tag[idx] for idx in idxs]
```

1.3 Generate batches

As Neural Networks are often trained with batches, weight updates of the network are based on several sequences at a single time. All sequences within a batch need to have the same length. To ensure this, they will be padded with a special token.

[12]:
```python
#batching function, which generates padded batches of tokens and tags
def batches_generator(batch_size, tokens, tags,
                      shuffle=True, allow_smaller_last_batch=True):

    n_samples = len(tokens)
    if shuffle:
        order = np.random.permutation(n_samples)
    else:
        order = np.arange(n_samples)

    n_batches = n_samples // batch_size
    if allow_smaller_last_batch and n_samples % batch_size:
        n_batches += 1

    for k in range(n_batches):
        batch_start = k * batch_size
        batch_end = min((k + 1) * batch_size, n_samples)
        current_batch_size = batch_end - batch_start
        x_list = []
        y_list = []
        max_len_token = 0
        for idx in order[batch_start: batch_end]:
            x_list.append(words2idxs(tokens[idx]))
            y_list.append(tags2idxs(tags[idx]))
            max_len_token = max(max_len_token, len(tags[idx]))

        #data into numpy nd-arrays filled with padding indices.
        x = np.ones([current_batch_size, max_len_token], dtype=np.int32) *␣
↪token2idx['<PAD>']
        y = np.ones([current_batch_size, max_len_token], dtype=np.int32) *␣
↪tag2idx['O']
        lengths = np.zeros(current_batch_size, dtype=np.int32)
        for n in range(current_batch_size):
            utt_len = len(x_list[n])
            x[n, :utt_len] = x_list[n]
```

```
            lengths[n] = utt_len
            y[n, :utt_len] = y_list[n]
        yield x, y, lengths
```

1.4 Build RNN

The network architecture will now be specified, based on TensorFlow building blocks. A LSTM network will be created, which will produce a probability distribution over tags for each token in a sentence. A Bi-LSTM will be used to take into account both right and left contexts of the token. A Dense layer will be used on top to perform tag classification.

[13]:
```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior() #this is to ensure v1 modules run that have been␣
 ↪deprecated in v2

import numpy as np
```

[14]:
```
class BiLSTMModel():
    pass
```

First, create placeholders to specify what data is going to be fed into the network during execution time. For this task, the following placeholders are needed: + input_batch — sequences of words (shape equals [batch_size, sequence_len]) + ground_truth_tags — sequences of tags (shape equals [batch_size, sequence_len]) + lengths — lengths of un-padded sequences (shape equals [batch_size]) + dropout_ph — dropout keep probability (predefined value 1) + learning_rate_ph — learning rate (needed as want to change value during training)

[15]:
```
#function to specify model placeholders
def declare_placeholders(self):

    # Placeholders for input and ground truth output.
    self.input_batch = tf.placeholder(dtype=tf.int32, shape=[None, None],␣
 ↪name='input_batch')
    self.ground_truth_tags = tf.placeholder(dtype=tf.int32, shape=[None, None],␣
 ↪name='ground_truth_tags')

    # Placeholder for lengths of the sequences.
    self.lengths = tf.placeholder(dtype=tf.int32, shape=[None], name='lengths')

    # Placeholder for a dropout keep probability. If we don't feed
    # a value for this placeholder, it will be equal to 1.0.
    self.dropout_ph = tf.placeholder_with_default(tf.cast(1.0, tf.float32),␣
 ↪shape=[])

    # Placeholder for a learning rate (tf.float32).
    self.learning_rate_ph = tf.placeholder(dtype=tf.float32, shape=[])
```

[16]:
```
BiLSTMModel.__declare_placeholders = classmethod(declare_placeholders)
```

Next, specify neural network layers. Preparatory steps: + Create embeddings matrix with tf.Variable. Specify its name (embeddings_matrix), type (tf.float32), and initialise with random values. + Create forward and backward LSTM cells (TensorFlow provides a number of RNN cells) + Wrap cells with DropoutWrapper (regularisation technique for neural networks)

Then, build computation graph that transforms an input_batch: + Look up embeddings for an input_batch in the prepared embedding_matrix + Pass embeddings through Bidirectional Dynamic RNN with specified forward and backward cells. + Create a dense layer on top. Its output will be used directly in loss function.

[17]:
```
#function to specify Bi_LSTM architecture and compute logits for inputs
def build_layers(self, vocabulary_size, embedding_dim, n_hidden_rnn, n_tags):

    # Create embedding variable with dtype tf.float32
    initial_embedding_matrix = np.random.randn(vocabulary_size, embedding_dim) /
 ↪ np.sqrt(embedding_dim)
    embedding_matrix_variable = tf.Variable(initial_embedding_matrix, dtype=tf.
↪float32)

    # Create RNN cells with n_hidden_rnn number of units
    # and dropout, initializing all *_keep_prob with dropout placeholder.
    forward_cell = tf.nn.rnn_cell.DropoutWrapper(tf.nn.rnn_cell.
↪LSTMCell(n_hidden_rnn),
                                                  input_keep_prob=self.
↪dropout_ph,
                                                  output_keep_prob=self.
↪dropout_ph,
                                                  state_keep_prob=self.
↪dropout_ph)
    backward_cell = tf.nn.rnn_cell.DropoutWrapper(tf.nn.rnn_cell.
↪LSTMCell(n_hidden_rnn),
                                                   input_keep_prob=self.
↪dropout_ph,
                                                   output_keep_prob=self.
↪dropout_ph,
                                                   state_keep_prob=self.
↪dropout_ph)

    # Look up embeddings for self.input_batch (tf.nn.embedding_lookup).
    # Shape: [batch_size, sequence_len, embedding_dim].
    embeddings =  tf.nn.embedding_lookup(embedding_matrix_variable, self.
↪input_batch)

    # Pass them through Bidirectional Dynamic RNN
    # Shape: [batch_size, sequence_len, 2 * n_hidden_rnn].
```

```
    # And initialise sequence_length as self.lengths and dtype as tf.float32.
    (rnn_output_fw, rnn_output_bw), _ = tf.nn.
 ↪bidirectional_dynamic_rnn(cell_fw=forward_cell,
                                                                            ␣
 ↪cell_bw=backward_cell,
                                                                            ␣
 ↪inputs=embeddings,
                                                                            ␣
 ↪sequence_length=self.lengths,
                                                                            ␣
 ↪dtype=tf.float32)
    rnn_output = tf.concat([rnn_output_fw, rnn_output_bw], axis=2)

    # Dense layer on top.
    # Shape: [batch_size, sequence_len, n_tags].
    self.logits = tf.layers.dense(rnn_output, n_tags, activation=None)
```

[18]:
```
BiLSTMModel.__build_layers = classmethod(build_layers)
```

To compute actual predictions of the neural network, need to apply softmax to last layer and find most probable tags with argmax.

[19]:
```
#function that transforms logits to probabilities and finds the most probable␣
 ↪tags
def compute_predictions(self):
    # Create softmax function
    softmax_output = tf.nn.softmax(self.logits)

    # Use argmax to get the most probable tags and set axis=-1 or argmax will␣
 ↪be calculated incorrectly
    self.predictions = tf.argmax(softmax_output, axis=-1)
```

[20]:
```
BiLSTMModel.__compute_predictions = classmethod(compute_predictions)
```

During training, need a loss function - here will use cross-entropy loss, efficiently implemented in TF as cross entropy with logits. It should be applied to logits of the model (not to softmax probabilities!). Do not want to take into account loss terms coming from tokens, so these need to be masked out before computing mean.

[21]:
```
#function that computes masked cross-entropy loss with logits
def compute_loss(self, n_tags, PAD_index):

    # Create cross entropy function function
    ground_truth_tags_one_hot = tf.one_hot(self.ground_truth_tags, n_tags)
    loss_tensor =  tf.nn.
 ↪softmax_cross_entropy_with_logits_v2(labels=ground_truth_tags_one_hot,␣
 ↪logits=self.logits)
```

```
    mask = tf.cast(tf.not_equal(self.input_batch, PAD_index), tf.float32)
    # Create loss function that ignores <PAD> tokens
    self.loss =  tf.reduce_mean(mask*loss_tensor)
```

[22]:
```
BiLSTMModel.__compute_loss = classmethod(compute_loss)
```

Here, will use Adam optimiser with a learning rate from the corresponding placeholder to optimise loss. Clipping will also be applied to eliminate exploding gradients.

[23]:
```
#function that specifies the optimiser and train_op for the model
def perform_optimization(self):

    # Create optimiser
    self.optimizer = tf.train.AdamOptimizer(learning_rate=self.learning_rate_ph)
    self.grads_and_vars = self.optimizer.compute_gradients(self.loss)

    # Gradient clipping - list comprehension used to apply only to gradients
    clip_norm = tf.cast(1.0, tf.float32)
    self.grads_and_vars = [(tf.clip_by_norm(grad, clip_norm), var) for grad,␣
↪var in self.grads_and_vars]

    self.train_op = self.optimizer.apply_gradients(self.grads_and_vars)
```

[24]:
```
BiLSTMModel.__perform_optimization = classmethod(perform_optimization)
```

All the parts of your network have now been specified. This will now be put to the constructor of our Bi-LSTM class to use it.

[25]:
```
def init_model(self, vocabulary_size, n_tags, embedding_dim, n_hidden_rnn,␣
↪PAD_index):
    self.__declare_placeholders()
    self.__build_layers(vocabulary_size, embedding_dim, n_hidden_rnn, n_tags)
    self.__compute_predictions()
    self.__compute_loss(n_tags, PAD_index)
    self.__perform_optimization()
```

[26]:
```
BiLSTMModel.__init__ = classmethod(init_model)
```

1.5 Train the network and predict tags

[27]:
```
#session.run initiates computations in the graph previously defined
#self.train_op, declared in perform_optimization, computed to train the network
def train_on_batch(self, session, x_batch, y_batch, lengths, learning_rate,␣
↪dropout_keep_probability):
    feed_dict = {self.input_batch: x_batch,
                 self.ground_truth_tags: y_batch,
```

```
                 self.learning_rate_ph: learning_rate,
                 self.dropout_ph: dropout_keep_probability,
                 self.lengths: lengths}

    session.run(self.train_op, feed_dict=feed_dict)
```

[28]:
```
BiLSTMModel.train_on_batch = classmethod(train_on_batch)
```

[29]:
```
#To predict tags, compute self.predictions
def predict_for_batch(self, session, x_batch, lengths):
    predictions = session.run(self.predictions,
                              feed_dict={self.input_batch:x_batch, self.lengths:
 ↪lengths})
    return predictions
```

[30]:
```
BiLSTMModel.predict_for_batch = classmethod(predict_for_batch)
```

1.6 Evaluation

[43]:
```
#functions for evaluation
from collections import OrderedDict

def _update_chunk(candidate, prev, current_tag, current_chunk, current_pos,␣
 ↪prediction=False):
    if candidate == 'B-' + current_tag:
        if len(current_chunk) > 0 and len(current_chunk[-1]) == 1:
                current_chunk[-1].append(current_pos - 1)
        current_chunk.append([current_pos])
    elif candidate == 'I-' + current_tag:
        if prediction and (current_pos == 0 or current_pos > 0 and prev.
 ↪split('-', 1)[-1] != current_tag):
            current_chunk.append([current_pos])
        if not prediction and (current_pos == 0 or current_pos > 0 and prev ==␣
 ↪'O'):
            current_chunk.append([current_pos])
    elif current_pos > 0 and prev.split('-', 1)[-1] == current_tag:
        if len(current_chunk) > 0:
            current_chunk[-1].append(current_pos - 1)

def _update_last_chunk(current_chunk, current_pos):
    if len(current_chunk) > 0 and len(current_chunk[-1]) == 1:
        current_chunk[-1].append(current_pos - 1)

def _tag_precision_recall_f1(tp, fp, fn):
    precision, recall, f1 = 0, 0, 0
    if tp + fp > 0:
```

```
        precision = tp / (tp + fp) * 100
    if tp + fn > 0:
        recall = tp / (tp + fn) * 100
    if precision + recall > 0:
        f1 = 2 * precision * recall / (precision + recall)
    return precision, recall, f1

def _aggregate_metrics(results, total_correct):
    total_true_entities = 0
    total_predicted_entities = 0
    total_precision = 0
    total_recall = 0
    total_f1 = 0
    for tag, tag_metrics in results.items():
        n_pred = tag_metrics['n_predicted_entities']
        n_true = tag_metrics['n_true_entities']
        total_true_entities += n_true
        total_predicted_entities += n_pred
        total_precision += tag_metrics['precision'] * n_pred
        total_recall += tag_metrics['recall'] * n_true

    accuracy = 0
    if total_true_entities > 0:
        accuracy = total_correct / total_true_entities * 100
    else:
        print('CAUTION! Accuracy equals zero because there are no '\
              'correct entities. Check the correctness of your data.')
    if total_predicted_entities > 0:
        total_precision = total_precision / total_predicted_entities
    total_recall = total_recall / total_true_entities
    if total_precision + total_recall > 0:
        total_f1 = 2 * total_precision * total_recall / (total_precision +␣
↪total_recall)
    return total_true_entities, total_predicted_entities, \
           total_precision, total_recall, total_f1, accuracy

def _print_info(n_tokens, total_true_entities, total_predicted_entities,␣
↪total_correct):
    print('processed {len} tokens ' \
          'with {tot_true} phrases; ' \
          'found: {tot_pred} phrases; ' \
          'correct: {tot_cor}.\n'.format(len=n_tokens,
                                         tot_true=total_true_entities,
                                         tot_pred=total_predicted_entities,
                                         tot_cor=total_correct))

def _print_metrics(accuracy, total_precision, total_recall, total_f1):
```

```
    print('precision:  {tot_prec:.2f}%; ' \
          'recall:  {tot_recall:.2f}%; ' \
          'F1:  {tot_f1:.2f}\n'.format(acc=accuracy,
                                        tot_prec=total_precision,
                                        tot_recall=total_recall,
                                        tot_f1=total_f1))

def _print_tag_metrics(tag, tag_results):
    print(('\t%12s' % tag) + ': precision:  {tot_prec:6.2f}%; ' \
                           'recall:  {tot_recall:6.2f}%; ' \
                           'F1:  {tot_f1:6.2f}; ' \
                           'predicted:  {tot_predicted:4d}\n'.
 ↪format(tot_prec=tag_results['precision'],
                                                                        ␣
 ↪tot_recall=tag_results['recall'],
                                                                        ␣
 ↪tot_f1=tag_results['f1'],
                                                                        ␣
 ↪tot_predicted=tag_results['n_predicted_entities']))


def precision_recall_f1(y_true, y_pred, print_results=True, short_report=False):
    # Find all tags
    tags = sorted(set(tag[2:] for tag in y_true + y_pred if tag != 'O'))

    results = OrderedDict((tag, OrderedDict()) for tag in tags)
    n_tokens = len(y_true)
    total_correct = 0

    # For eval_conll_try we find all chunks in the ground truth and prediction
    # For each chunk we store starting and ending indices
    for tag in tags:
        true_chunk = list()
        predicted_chunk = list()
        for position in range(n_tokens):
            _update_chunk(y_true[position], y_true[position - 1], tag,␣
 ↪true_chunk, position)
            _update_chunk(y_pred[position], y_pred[position - 1], tag,␣
 ↪predicted_chunk, position, True)

        _update_last_chunk(true_chunk, position)
        _update_last_chunk(predicted_chunk, position)

        # Then we find all correctly classified intervals
        # True positive results
        tp = sum(chunk in predicted_chunk for chunk in true_chunk)
```

```
        total_correct += tp

        # And then just calculate errors of the first and second kind
        # False negative
        fn = len(true_chunk) - tp
        # False positive
        fp = len(predicted_chunk) - tp
        precision, recall, f1 = _tag_precision_recall_f1(tp, fp, fn)

        results[tag]['precision'] = precision
        results[tag]['recall'] = recall
        results[tag]['f1'] = f1
        results[tag]['n_predicted_entities'] = len(predicted_chunk)
        results[tag]['n_true_entities'] = len(true_chunk)

    total_true_entities, total_predicted_entities, \
           total_precision, total_recall, total_f1, accuracy =␣
↪_aggregate_metrics(results, total_correct)

    if print_results:
        _print_info(n_tokens, total_true_entities, total_predicted_entities,␣
↪total_correct)
        _print_metrics(accuracy, total_precision, total_recall, total_f1)

        if not short_report:
            for tag, tag_results in results.items():
                _print_tag_metrics(tag, tag_results)
    return results
```

[35]:
```
#function to perform predictions and transform indices to tokens and tags
def predict_tags(model, session, token_idxs_batch, lengths):
    tag_idxs_batch = model.predict_for_batch(session, token_idxs_batch, lengths)

    tags_batch, tokens_batch = [], []
    for tag_idxs, token_idxs in zip(tag_idxs_batch, token_idxs_batch):
        tags, tokens = [], []
        for tag_idx, token_idx in zip(tag_idxs, token_idxs):
            tags.append(idx2tag[tag_idx])
            tokens.append(idx2token[token_idx])
        tags_batch.append(tags)
        tokens_batch.append(tokens)
    return tags_batch, tokens_batch


#function that computses NER quality measures
def eval_conll(model, session, tokens, tags, short_report=True):
    y_true, y_pred = [], []
```

```
    for x_batch, y_batch, lengths in batches_generator(1, tokens, tags):
        tags_batch, tokens_batch = predict_tags(model, session, x_batch,␣
↪lengths)
        if len(x_batch[0]) != len(tags_batch[0]):
            raise Exception("Incorrect length of prediction for the input, "
                            "expected length: %i, got: %i" % (len(x_batch[0]),␣
↪len(tags_batch[0])))
        predicted_tags = []
        ground_truth_tags = []
        for gt_tag_idx, pred_tag, token in zip(y_batch[0], tags_batch[0],␣
↪tokens_batch[0]):
            if token != '<PAD>':
                ground_truth_tags.append(idx2tag[gt_tag_idx])
                predicted_tags.append(pred_tag)

        # extend every prediction and ground truth sequence with 'O' tag to␣
↪indicate a possible end of entity.
        y_true.extend(ground_truth_tags + ['O'])
        y_pred.extend(predicted_tags + ['O'])

    results = precision_recall_f1(y_true, y_pred, print_results=True,␣
↪short_report=short_report)
    return results
```

1.7 Run experiment

Create BiLSTMModel model with the following parameters: + vocabulary_size — number of tokens + n_tags — number of tags + embedding_dim — dimension of embeddings + n_hidden_rnn — size of hidden layers for RNN + PAD_index — an index of the padding token ().

Set hyperparameters: + batch_size + epochs + starting value of learning_rate + learning_rate_decay + dropout_keep_probability

[39]:
```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

tf.reset_default_graph()

model = BiLSTMModel(vocabulary_size=len(token2idx), n_tags=len(tag2idx),␣
↪embedding_dim=200, n_hidden_rnn=200, PAD_index=token2idx['<PAD>'])

batch_size = 32
n_epochs = 4
learning_rate = 0.005
learning_rate_decay = np.sqrt(2)
dropout_keep_probability = 0.5
```

```
WARNING:tensorflow:From <ipython-input-17-f154aca0eafe>:10: LSTMCell.__init__
```

```
(from tensorflow.python.ops.rnn_cell_impl) is deprecated and will be removed in
a future version.
Instructions for updating:
This class is equivalent as tf.keras.layers.LSTMCell, and will be replaced by
that in Tensorflow 2.0.
WARNING:tensorflow:From <ipython-input-17-f154aca0eafe>:30:
bidirectional_dynamic_rnn (from tensorflow.python.ops.rnn) is deprecated and
will be removed in a future version.
Instructions for updating:
Please use `keras.layers.Bidirectional(keras.layers.RNN(cell))`, which is
equivalent to this API
WARNING:tensorflow:From /Users/charlottefettes/opt/anaconda3/lib/python3.7/site-
packages/tensorflow_core/python/ops/rnn.py:464: dynamic_rnn (from
tensorflow.python.ops.rnn) is deprecated and will be removed in a future
version.
Instructions for updating:
Please use `keras.layers.RNN(cell)`, which is equivalent to this API
WARNING:tensorflow:From /Users/charlottefettes/opt/anaconda3/lib/python3.7/site-
packages/tensorflow_core/python/ops/rnn_cell_impl.py:958: Layer.add_variable
(from tensorflow.python.keras.engine.base_layer) is deprecated and will be
removed in a future version.
Instructions for updating:
Please use `layer.add_weight` method instead.
WARNING:tensorflow:From /Users/charlottefettes/opt/anaconda3/lib/python3.7/site-
packages/tensorflow_core/python/ops/rnn_cell_impl.py:962: calling Zeros.__init__
(from tensorflow.python.ops.init_ops) with dtype is deprecated and will be
removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
WARNING:tensorflow:From <ipython-input-17-f154aca0eafe>:35: dense (from
tensorflow.python.layers.core) is deprecated and will be removed in a future
version.
Instructions for updating:
Use keras.layers.Dense instead.
WARNING:tensorflow:From /Users/charlottefettes/opt/anaconda3/lib/python3.7/site-
packages/tensorflow_core/python/layers/core.py:187: Layer.apply (from
tensorflow.python.keras.engine.base_layer) is deprecated and will be removed in
a future version.
Instructions for updating:
Please use `layer.__call__` method instead.
```

Now run the training.

[44]:
```python
sess = tf.Session()
sess.run(tf.global_variables_initializer())

print('Start training... \n')
```

```
for epoch in range(n_epochs):
    # For each epoch evaluate the model on train and validation data
    print('-' * 20 + ' Epoch {} '.format(epoch+1) + 'of {} '.format(n_epochs) +␣
 ↪'-' * 20)
    print('Train data evaluation:')
    eval_conll(model, sess, train_tokens, train_tags, short_report=True)
    print('Validation data evaluation:')
    eval_conll(model, sess, validation_tokens, validation_tags,␣
 ↪short_report=True)

    # Train the model
    for x_batch, y_batch, lengths in batches_generator(batch_size,␣
 ↪train_tokens, train_tags):
        model.train_on_batch(sess, x_batch, y_batch, lengths, learning_rate,␣
 ↪dropout_keep_probability)

    # Decaying the learning rate
    learning_rate = learning_rate / learning_rate_decay

print('...training finished.')
```

```
Start training…

-------------------- Epoch 1 of 4 --------------------
Train data evaluation:
processed 105778 tokens with 4489 phrases; found: 78413 phrases; correct: 228.

precision:  0.29%; recall:  5.08%; F1:  0.55

Validation data evaluation:
processed 12836 tokens with 537 phrases; found: 9552 phrases; correct: 29.

precision:  0.30%; recall:  5.40%; F1:  0.57

-------------------- Epoch 2 of 4 --------------------
Train data evaluation:
processed 105778 tokens with 4489 phrases; found: 3014 phrases; correct: 622.

precision:  20.64%; recall:  13.86%; F1:  16.58

Validation data evaluation:
processed 12836 tokens with 537 phrases; found: 222 phrases; correct: 55.

precision:  24.77%; recall:  10.24%; F1:  14.49

-------------------- Epoch 3 of 4 --------------------
Train data evaluation:
```

```
processed 105778 tokens with 4489 phrases; found: 4762 phrases; correct: 1838.

precision:  38.60%; recall:  40.94%; F1:  39.74

Validation data evaluation:
processed 12836 tokens with 537 phrases; found: 370 phrases; correct: 133.

precision:  35.95%; recall:  24.77%; F1:  29.33

-------------------- Epoch 4 of 4 --------------------
Train data evaluation:
processed 105778 tokens with 4489 phrases; found: 4567 phrases; correct: 2883.

precision:  63.13%; recall:  64.22%; F1:  63.67

Validation data evaluation:
processed 12836 tokens with 537 phrases; found: 367 phrases; correct: 167.

precision:  45.50%; recall:  31.10%; F1:  36.95

…training finished.
```

Using the eval_conll function, full quality reports will be printed for this model on train, validation, and test sets.

[45]:
```python
print('-' * 20 + ' Train set quality: ' + '-' * 20)
train_results = eval_conll(model, sess, train_tokens, train_tags,␣
 ↪short_report=False)

print('-' * 20 + ' Validation set quality: ' + '-' * 20)
validation_results = eval_conll(model, sess, validation_tokens,␣
 ↪validation_tags, short_report=False)

print('-' * 20 + ' Test set quality: ' + '-' * 20)
test_results = eval_conll(model, sess, test_tokens, test_tags,␣
 ↪short_report=False)
```

```
-------------------- Train set quality: --------------------
processed 105778 tokens with 4489 phrases; found: 4792 phrases; correct: 3465.

precision:  72.31%; recall:  77.19%; F1:  74.67

             company: precision:   81.78%; recall:   88.65%; F1:   85.07;
predicted:   697

            facility: precision:   67.65%; recall:   80.57%; F1:   73.55;
predicted:   374
```

```
             geo-loc: precision:  76.89%; recall:  94.18%; F1:  84.66;
predicted:  1220

               movie: precision:   0.00%; recall:   0.00%; F1:   0.00;
predicted:    14

         musicartist: precision:  29.85%; recall:   8.62%; F1:  13.38;
predicted:    67

               other: precision:  68.19%; recall:  78.73%; F1:  73.08;
predicted:   874

              person: precision:  77.04%; recall:  94.70%; F1:  84.96;
predicted:  1089

             product: precision:  53.79%; recall:  75.79%; F1:  62.92;
predicted:   448

          sportsteam: precision:  88.89%; recall:   3.69%; F1:   7.08;
predicted:     9

              tvshow: precision:   0.00%; recall:   0.00%; F1:   0.00;
predicted:     0

-------------------- Validation set quality: --------------------
processed 12836 tokens with 537 phrases; found: 436 phrases; correct: 186.

precision:  42.66%; recall:  34.64%; F1:  38.23

             company: precision:  61.63%; recall:  50.96%; F1:  55.79;
predicted:    86

            facility: precision:  30.30%; recall:  29.41%; F1:  29.85;
predicted:    33

             geo-loc: precision:  65.59%; recall:  53.98%; F1:  59.22;
predicted:    93

               movie: precision:   0.00%; recall:   0.00%; F1:   0.00;
predicted:     0

         musicartist: precision:   0.00%; recall:   0.00%; F1:   0.00;
predicted:     0

               other: precision:  28.57%; recall:  34.57%; F1:  31.28;
predicted:    98

              person: precision:  42.86%; recall:  26.79%; F1:  32.97;
```

```
predicted:    70

             product: precision:    7.14%; recall:   11.76%; F1:    8.89;
predicted:    56

          sportsteam: precision:    0.00%; recall:    0.00%; F1:    0.00;
predicted:     0

              tvshow: precision:    0.00%; recall:    0.00%; F1:    0.00;
predicted:     0

-------------------- Test set quality: --------------------
processed 13258 tokens with 604 phrases; found: 482 phrases; correct: 221.

precision:  45.85%; recall:  36.59%; F1:  40.70

             company: precision:   63.79%; recall:   44.05%; F1:   52.11;
predicted:    58

            facility: precision:   44.44%; recall:   34.04%; F1:   38.55;
predicted:    36

             geo-loc: precision:   67.91%; recall:   55.15%; F1:   60.87;
predicted:   134

               movie: precision:    0.00%; recall:    0.00%; F1:    0.00;
predicted:     0

         musicartist: precision:    0.00%; recall:    0.00%; F1:    0.00;
predicted:     2

               other: precision:   25.00%; recall:   32.04%; F1:   28.09;
predicted:   132

              person: precision:   51.25%; recall:   39.42%; F1:   44.57;
predicted:    80

             product: precision:    7.50%; recall:   10.71%; F1:    8.82;
predicted:    40

          sportsteam: precision:    0.00%; recall:    0.00%; F1:    0.00;
predicted:     0

              tvshow: precision:    0.00%; recall:    0.00%; F1:    0.00;
predicted:     0
```

[46]:
```
#the model will now be retrained using different parameters
tf.reset_default_graph()

model = BiLSTMModel(vocabulary_size=len(token2idx), n_tags=len(tag2idx),␣
↪embedding_dim=200, n_hidden_rnn=200, PAD_index=token2idx['<PAD>'])

batch_size = 32
n_epochs = 4
learning_rate = 0.01
learning_rate_decay = np.sqrt(2)
dropout_keep_probability = 0.6
sess = tf.Session()
sess.run(tf.global_variables_initializer())

print('Start training... \n')
for epoch in range(n_epochs):
    # For each epoch evaluate the model on train and validation data
    print('-' * 20 + ' Epoch {} '.format(epoch+1) + 'of {} '.format(n_epochs) +␣
↪'-' * 20)
    print('Train data evaluation:')
    eval_conll(model, sess, train_tokens, train_tags, short_report=True)
    print('Validation data evaluation:')
    eval_conll(model, sess, validation_tokens, validation_tags,␣
↪short_report=True)

    # Train the model
    for x_batch, y_batch, lengths in batches_generator(batch_size,␣
↪train_tokens, train_tags):
        model.train_on_batch(sess, x_batch, y_batch, lengths, learning_rate,␣
↪dropout_keep_probability)

    # Decaying the learning rate
    learning_rate = learning_rate / learning_rate_decay

print('...training finished.')
```

```
Start training…

-------------------- Epoch 1 of 4 --------------------
Train data evaluation:
processed 105778 tokens with 4489 phrases; found: 79511 phrases; correct: 206.

precision:  0.26%; recall:  4.59%; F1:  0.49

Validation data evaluation:
processed 12836 tokens with 537 phrases; found: 9576 phrases; correct: 19.
```

```
precision:  0.20%; recall:  3.54%; F1:  0.38

-------------------- Epoch 2 of 4 --------------------
Train data evaluation:
processed 105778 tokens with 4489 phrases; found: 2706 phrases; correct: 1498.

precision:  55.36%; recall:  33.37%; F1:  41.64

Validation data evaluation:
processed 12836 tokens with 537 phrases; found: 211 phrases; correct: 115.

precision:  54.50%; recall:  21.42%; F1:  30.75

-------------------- Epoch 3 of 4 --------------------
Train data evaluation:
processed 105778 tokens with 4489 phrases; found: 4723 phrases; correct: 3217.

precision:  68.11%; recall:  71.66%; F1:  69.84

Validation data evaluation:
processed 12836 tokens with 537 phrases; found: 371 phrases; correct: 182.

precision:  49.06%; recall:  33.89%; F1:  40.09

-------------------- Epoch 4 of 4 --------------------
Train data evaluation:
processed 105778 tokens with 4489 phrases; found: 4778 phrases; correct: 4028.

precision:  84.30%; recall:  89.73%; F1:  86.93

Validation data evaluation:
processed 12836 tokens with 537 phrases; found: 439 phrases; correct: 200.

precision:  45.56%; recall:  37.24%; F1:  40.98

…training finished.
```

Using the eval_conll function, full quality reports will be printed for the final model on train, validation, and test sets.

[47]:
```
print('-' * 20 + ' Train set quality: ' + '-' * 20)
train_results = eval_conll(model, sess, train_tokens, train_tags,␣
 ↪short_report=False)

print('-' * 20 + ' Validation set quality: ' + '-' * 20)
validation_results = eval_conll(model, sess, validation_tokens,␣
 ↪validation_tags, short_report=False)
```

```
print('-' * 20 + ' Test set quality: ' + '-' * 20)
test_results = eval_conll(model, sess, test_tokens, test_tags,␣
 ↪short_report=False)
```

```
-------------------- Train set quality: --------------------
processed 105778 tokens with 4489 phrases; found: 4596 phrases; correct: 4257.

precision:  92.62%; recall:  94.83%; F1:  93.71

             company: precision:   93.13%; recall:   97.05%; F1:   95.05;
predicted:   670

            facility: precision:   91.25%; recall:   92.99%; F1:   92.11;
predicted:   320

             geo-loc: precision:   96.05%; recall:   97.59%; F1:   96.81;
predicted:  1012

               movie: precision:   70.27%; recall:   76.47%; F1:   73.24;
predicted:    74

         musicartist: precision:   85.48%; recall:   91.38%; F1:   88.33;
predicted:   248

               other: precision:   91.81%; recall:   94.72%; F1:   93.24;
predicted:   781

              person: precision:   96.83%; recall:   96.39%; F1:   96.61;
predicted:   882

             product: precision:   87.57%; recall:   93.08%; F1:   90.24;
predicted:   338

          sportsteam: precision:   92.89%; recall:   90.32%; F1:   91.59;
predicted:   211

              tvshow: precision:   70.00%; recall:   72.41%; F1:   71.19;
predicted:    60

-------------------- Validation set quality: --------------------
processed 12836 tokens with 537 phrases; found: 405 phrases; correct: 192.

precision:  47.41%; recall:  35.75%; F1:  40.76

             company: precision:   67.86%; recall:   54.81%; F1:   60.64;
predicted:    84
```

```
           facility: precision:  40.00%; recall:  35.29%; F1:  37.50;
predicted:    30

            geo-loc: precision:  65.82%; recall:  46.02%; F1:  54.17;
predicted:    79

              movie: precision:   0.00%; recall:   0.00%; F1:   0.00;
predicted:     4

        musicartist: precision:  22.73%; recall:  17.86%; F1:  20.00;
predicted:    22

              other: precision:  40.00%; recall:  34.57%; F1:  37.09;
predicted:    70

             person: precision:  53.57%; recall:  26.79%; F1:  35.71;
predicted:    56

            product: precision:   9.30%; recall:  11.76%; F1:  10.39;
predicted:    43

         sportsteam: precision:  36.36%; recall:  20.00%; F1:  25.81;
predicted:    11

             tvshow: precision:   0.00%; recall:   0.00%; F1:   0.00;
predicted:     6

-------------------- Test set quality: --------------------
processed 13258 tokens with 604 phrases; found: 528 phrases; correct: 231.

precision:  43.75%; recall:  38.25%; F1:  40.81

            company: precision:  59.02%; recall:  42.86%; F1:  49.66;
predicted:    61

           facility: precision:  43.18%; recall:  40.43%; F1:  41.76;
predicted:    44

            geo-loc: precision:  72.41%; recall:  50.91%; F1:  59.79;
predicted:   116

              movie: precision:   0.00%; recall:   0.00%; F1:   0.00;
predicted:     2

        musicartist: precision:   8.33%; recall:   7.41%; F1:   7.84;
predicted:    24

              other: precision:  37.50%; recall:  37.86%; F1:  37.68;
```

```
predicted:   104

              person: precision:   65.00%; recall:   37.50%; F1:   47.56;
predicted:    60

             product: precision:    4.55%; recall:   14.29%; F1:    6.90;
predicted:    88

          sportsteam: precision:   34.78%; recall:   25.81%; F1:   29.63;
predicted:    23

              tvshow: precision:    0.00%; recall:    0.00%; F1:    0.00;
predicted:     6
```

[ ]:

```

```