Prepared By:
Sami & Ron

Deep Learning for NLP
Coding Assignment 2

CS 544
Due: Sun April 28

Introduction

The goal of this coding assignment to get you expertise in TensorFlow, especially in developing code from the grounds-up. This assignment will be not much hand-held: you have to do most things from scratch, including creating a `tf.Session`. In this task, you will be implementing a sequence-to-sequence Recursive Neural Network (RNN) model in TensorFlow. You will be using the same data from the HMM Coding Assignment 3 for Part-of-Speech tagging. In particular, you are expected to:

- Populate the starter code in designated places marked by `TODO(student)`.
  - The starter code is on: http://sami.haija.org/cs544/DL5/starter.py.
- Write code for reading the data files and producing numpy arrays that will be used for training [this has to produce expected outcomes, as measured by grading scripts]. This should be implemented in class `DatasetReader`.
- Write code for constructing and training the model [here, you should be creative, per grading scheme below!]. This should be implemented in class `SequenceModel`.
  - You can optionally fill-in the `main()` code-block, so that you can run locally (or in vocareum without submitting i.e. for debugging). However, the `main()` function will **not** be run through the submission script.
  - You must implement the functions that are annotated in the starter code. The grading script will train your model for exactly K seconds[1]. Therefore, you must explore good hyperparameters for this training budget [e.g. batch size, learning rate], which are always a function of your model architecture and the training algorithm (there is no one answer that fits all!)

Grading

The grading of this assignment is more-in-line with the first 3 coding assignments, except that you are competing with one-another, not with teaching staff, to some extent. You can also receive bonus credits. **In fact, the theoretical maximum grade for the assignment is 180%, which would be given to the single highest-accuracy student on unseen language, if her/his performance is $\approx 100\%$ on Italian and Japanese.**

There will **be no late days** for this assignment, since there is a competition. No exceptions will be made.

The grading scheme is subject to change, and will be finalized by Friday April 12.

[1] Likely, K will be 120 to 200 seconds

[30 points] Task 1: Data Processing

20 points Implement `ReadFile`. This function is used by `ReadData`. The function header is copied here for your convenience

```
def ReadFile(self, filename, term_index, tag_index):
  """Reads file into dataset, while populating term_index and tag_index.

  Args:
    filename: Path of text file containing sentences and tags. Each line is a
      sentence and each term is followed by "/tag". Note: some terms might
      have a "/" e.g. my/word/tag -- the term is "my/word" and the last "/"
      separates the tag.
    term_index: dictionary to be populated with every unique term (i.e. before
      the last "/") to point to an integer. All integers must be utilized from
      0 to number of unique terms - 1, without any gaps nor repetitions.
    tag_index: same as term_index, but for tags.

  Return:
    The parsed file as a list of lists: [parsedLine1, parsedLine2, ...]
    each parsedLine is a list: [(term1, tag1), (term2, tag2), ...]
  """
```

10 points Implement `BuildMatrices`. The function header is copied here for your convenience

```
def BuildMatrices(dataset):
  """Converts dataset [returned by ReadFile] to np arrays for tags, terms,  lengths.

  Args:
    dataset: Returned by method ReadFile. It is a list (length N) of lists:
      [sentence1, sentence2, ...], where every sentence is a list:
      [(word1, tag1), (word2, tag2), ...], where every word and tag are integers.

  Returns:
    Tuple of 3 numpy arrays: (terms_matrix, tags_matrix, lengths_arr)
      terms_matrix: shape (N, T) int64 numpy array. Row i contains the word
        indices in dataset[i].
      tags_matrix: shape (N, T) int64 numpy array. Row i contains the tag
        indices in dataset[i].
      lengths: shape (N) int64 numpy array. Entry i contains the length of
        sentence in dataset[i].

    T is the maximum length. For example, calling as:
      BuildMatrices([[(1,2), (4,10)], [(13, 20), (3, 6), (7, 8), (3, 20)]])
    i.e. with two sentences, first with length 2 and second with length 4,
    should return the tuple:
    (
```

```
      [[1, 4, 0, 0],    # Note: 0 padding.
       [13, 3, 7, 3]],

      [[2, 10, 0, 0],   # Note: 0 padding.
       [20, 6, 8, 20]],

      [2, 4]
    )
  """
```

Note: this task is completely independent of the second task. You are graded each in isolation. In fact, when grading Task 2, we use our implementation of these functions.

[60 + ? points] Task 2: Coding Sequence Models

In this task, you are expected to populate class `SequenceModel`.

You might find these functions useful:

- SimpleRNNCell. You construct the class once, and you can call it to map prev-state tensor and current values tensor, to next state tensor.
- `tf.reshape`: Changes the number of dimensions of a tensor.
- `tf.nn.embedding_lookup`: Takes a float matrix tensor (embeddings) of shape (N, d), and int64 any-dimension ($= D$) tensor (indices): returns float tensor of $D + [d]$.
- `tf.get_variable`: creates a variable (E.g. an embedding variable) and defaults it to "*trainable*".

Grades:

- You will receive **zero credit** if you do not implement any of the **required methods**:
  - `save_model`: Saves the trained model to a file.
  - `load_model`: Loads the trained model from a file. For this one and the above, you might find the first Deep Learning material useful: http://sami.haija.org/cs544/DL1
  - `run_inference`: Given sentences (i.e. matrices of term IDs and their lengths), return the part-of-speech tag ID for every word in every sentence.
- If your class does not implement `lengths_vector_to_binary_matrix` correctly, then will get -10 points (though the minimum grade for the task is 0).

- If you implement the required methods, then your grade depends on your model's accuracy on unseen data. The following grade table applies for the Italian and the Japanese texts:

Accuracy:	$< 50\%$	$50\% \leq a \leq 85\%$	85%	$a \geq 85\%$
Grade:	0	$\frac{a-50}{35} \times 30$	30	$30 + \frac{a-85}{7.5} \times 30$

Bonus Points: The top 5 performers get additional grades. The performance is measured on unseen language.

Rank	1	2	3	4	5
+Bonus Points	60	50	40	30	20

Note: We will only consider for bonus students who receive $> 92.5\%$ test accuracy on the Italian and Japanese languages. In addition, the bonus points will only be awarded for people who explain their method through a 2-page PDF or a couple of slides. The criteria for getting the bonus credits, is that the teaching staff must be able to understand how replicate the method.

You might find this code useful:

```
xemb = tf.nn.embedding_lookup( ... )
rnn_cell = tf.keras.layers.SimpleRNNCell(state_size)  # Callable instance.
states = []
cur_state = tf.zeros(shape=[1, state_size])
for i in xrange(max_length):
    cur_state = rnn_cell(xemb[:, i, :], [cur_state])[0]  # shape (batch, state_size)
    states.append(cur_state)

stacked_states = tf.stack(states, axis=1)  # Shape (batch, max_length, state_size)
```

Finally, your model will be tested by the grader script, which invokes 2 programs, the training program then the testing program. They will run as:

```
# TRAINING PROGRAM
model = SequenceModel(max_length, num_terms, num_tags)
model.build_inference()
model.build_training()
while (time_spent < K):
    model.train_epoch(terms, tags, lengths)
model.save_model('/some/file/path')

# TESTING PROGRAM [runs in a separate shell command, after training program]
model = SequenceModel(max_length, num_terms, num_tags)
model.load_model('/some/file/path')
model.build_inference()
model.run_inference(test_tags, test_lengths)  # and compare with ground-truth
```