Assignment 3 - Neural Dependency Parsers

CSC 485/2501 - Fall 2017

UNIVERSITY OF TORONTO

1 (d)

```
def complete(self):
        '''bool: return true iff the PartialParse is complete
        Assume that the PartialParse is valid
        '''
```

```
def parse_step(self, transition_id, deprel=None):
        '''Update the PartialParse with a transition'''
```

- Use your answer to 1(a) as a reference
- Remember to raise a ValueError if the transition is illegal e.g.
  ```
  raise ValueError('Something bad happened')
  ```

1 (e)

```
def minibatch_parse(sentences, model, batch_size):
    """Parses a list of sentences in minibatches using a model."""
```

- Remember that calls to parse_step may raise a ValueError exception. Remove any such ‘stuck’ parses from your list of unfinished parses e.g.

```
try:
    # Do stuff
except (ValueError):
    # Do other stuff
```

1 (f)

```
ef get_oracle(self, graph):
        '''Given a projective dependency graph, determine an appropriate transistion'''
```

- Once again, use your answer to 1 (a) as a reference

Remember to Test your code!

Tensorflow

You define a computational graph, which is then constructed at compile-time and may be run on multiple CPUs, GPUs, etc

Constants: Don’t change once the computational graph is constructed.

Variables: These may change once the computational graph is constructed. Your optimizer will try and change these.

Placeholders: Are the ____s. Fill in the ____s using your feed-dict. Use the actual placeholders rather than strings.

Tensorflow - Installing and Running It

At Home:

- We recommend the CPU version: `pip install tensorflow`
- You'll also need **numpy** and **nltk**
- Code in question 2 (b) runs in 30min
  on my potato

Tensorflow - Installing and Running It

In school:

- Tensorflow not installed on wolf
- You’ll have to use workstations in **BA3175**, **BA3185**, and **BA3195** or **BA2240**
- Code in question 2 (b) runs in **10min!**

Toy Example

Say we wanted to figure out f(x) for
x = [1,2,3] and y=[3,5,7]
Let’s try **y = mx + c**

```
import tensorflow as tf

y = tf.placeholder(tf.float32, (None, ))
x = tf.placeholder(tf.float32, (None, ))

m = tf.Variable(0.0, dtype=tf.float32)
c = tf.constant(1.0, dtype=tf.float32)

y_ = m*x + c
```

```
loss = tf.reduce_mean(tf.square(y_ - y))
optimizer =
tf.train.GradientDescentOptimizer(0.1)
train = optimizer.minimize(loss)

with tf.Session() as sess:
    # Initialize our variables
    init = tf.global_variables_initializer()
    sess.run(init)
    wrong_answer = sess.run(y_, {x: [4]})
    for i in range(10):
        sess.run(
            train,
            {y: [3,5,7], x: [1,2,3]}
        )
    right_answer = sess.run(y_, {x: [4]})
    print(wrong_answer, right_answer)
```

2 (b)

- Take a peek at the Config class and try and figure out how your model uses it.

```
def add_embeddings(self):
        """Creates embeddings that map word, tag, deprels to vectors"""
```

- Remember that the embedding matrices should be **variables** not constants. We want to learn better embedded representations if we can.

```
def add_prediction_op(self):
        """Adds the single layer neural network"""
```

- Make sure to use your Xavier initializer correctly
- Use ReLU activation rather than the cubic function.

2 (b)

```
def add_loss_op(self, pred):
        """Adds Ops for the loss function to the computational graph."""
```

- Hint: `tf.nn.softmax_cross_entropy_with_logits` calculates softmax and cross-entropy at the same time.