Simple_algorithms_using_AutoGraph

April 24, 2019

0.1 AutoGraph: examples of simple algorithms

This notebook shows how you can use AutoGraph to compile simple algorithms and run them in TensorFlow.

It requires the nightly build of TensorFlow, which is installed below.

```
In [0]: !pip install -U -q tf-nightly
```

0.1.1 Fibonacci numbers

https://en.wikipedia.org/wiki/Fibonacci_number

```
In [0]: import tensorflow as tf
        from tensorflow.contrib import autograph as ag


        def fib(n):
          f1 = 0
          f2 = 1
          for i in range(n):
            tmp = f2
            f2 = f2 + f1
            f1 = tmp
            print(i, ': ', f2)
          return f2


        with tf.Graph().as_default():
          final_fib = ag.to_graph(fib)(tf.constant(10))
          with tf.Session() as sess:
            sess.run(final_fib)
```

```
0 :  1
1 :  2
2 :  3
3 :  5
4 :  8
5 :  13
```

```
6 :  21
7 :  34
8 :  55
9 :  89
```

Generated code

```
In [0]: print(ag.to_code(fib))
```

```
from __future__ import print_function
import tensorflow as tf

def tf__fib(n):
  try:
    with tf.name_scope('fib'):
      f1 = 0
      f2 = 1

      def extra_test(f1_1, f2_1):
        with tf.name_scope('extra_test'):
          return True

      def loop_body(i, f1_1, f2_1):
        with tf.name_scope('loop_body'):
          tmp = f2_1
          f2_1 = f2_1 + f1_1
          f1_1 = tmp
          with ag__.utils.control_dependency_on_returns(ag__.utils.
              dynamic_print(i, ': ', f2_1)):
            f2, i_1 = ag__.utils.alias_tensors(f2_1, i)
            return f1_1, f2
      f1, f2 = ag__.for_stmt(ag__.utils.dynamic_builtin(range, n),
          extra_test, loop_body, (f1, f2))
      return f2
  except:
    ag__.rewrite_graph_construction_error(ag_source_map__)
```

0.1.2 Fizz Buzz

https://en.wikipedia.org/wiki/Fizz_buzz

```
In [0]: import tensorflow as tf
        from tensorflow.contrib import autograph as ag

        def fizzbuzz(i, n):
```

```
          while i < n:
            msg = ''
            if i % 3 == 0:
              msg += 'Fizz'
            if i % 5 == 0:
              msg += 'Buzz'
            if msg == '':
              msg = tf.as_string(i)
            print(msg)
            i += 1
          return i

        with tf.Graph().as_default():
          final_i = ag.to_graph(fizzbuzz)(tf.constant(10), tf.constant(16))
          with tf.Session() as sess:
            sess.run(final_i)
```

```
Buzz
11
Fizz
13
14
FizzBuzz
```

Generated code

```
In [0]: print(ag.to_code(fizzbuzz))
```

```
from __future__ import print_function
import tensorflow as tf

def tf__fizzbuzz(i, n):
  try:
    with tf.name_scope('fizzbuzz'):

      def loop_test(i_1):
        with tf.name_scope('loop_test'):
          return tf.less(i_1, n)

      def loop_body(i_1):
        with tf.name_scope('loop_body'):
          msg = ''

          def if_true():
            with tf.name_scope('if_true'):
              msg_1, = msg,
              msg_1 += 'Fizz'
```

```
            return msg_1,

          def if_false():
            with tf.name_scope('if_false'):
              return msg,
          msg = ag__.utils.run_cond(tf.equal(i_1 % 3, 0), if_true, if_false)

          def if_true_1():
            with tf.name_scope('if_true_1'):
              msg_2, = msg,
              msg_2 += 'Buzz'
              return msg_2,

          def if_false_1():
            with tf.name_scope('if_false_1'):
              return msg,
          msg = ag__.utils.run_cond(tf.equal(i_1 % 5, 0), if_true_1, if_false_1
              )

          def if_true_2():
            with tf.name_scope('if_true_2'):
              msg_3, = msg,
              msg_3 = tf.as_string(i_1)
              return msg_3,

          def if_false_2():
            with tf.name_scope('if_false_2'):
              return msg,
          msg = ag__.utils.run_cond(tf.equal(msg, ''), if_true_2, if_false_2)
          with ag__.utils.control_dependency_on_returns(ag__.utils.
              dynamic_print(msg)):
            msg_4 = ag__.utils.alias_tensors(msg)
            i_1 += 1
            return i_1,
      i = ag__.while_stmt(loop_test, loop_body, (i,), (tf, n, ag__, i))
      return i
  except:
    ag__.rewrite_graph_construction_error(ag_source_map__)
```

0.1.3 Conway's Game of Life

https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

Testing boilerplate

```
In [0]: NUM_STEPS = 1
```

Game of Life for AutoGraph

In [0]:
```
#@test {"skip": true}
NUM_STEPS = 100
```

In [0]:
```
import time
import traceback

from matplotlib import pyplot as plt
from matplotlib import animation as anim
import tensorflow as tf
from tensorflow.contrib import autograph as ag
from IPython import display


@ag.do_not_convert(ag.RunMode.PY_FUNC)
def render(boards):
  fig = plt.figure()

  ims = []
  for b in boards:
    im = plt.imshow(b, interpolation='none')
    im.axes.get_xaxis().set_visible(False)
    im.axes.get_yaxis().set_visible(False)
    ims.append([im])

  try:
    ani = anim.ArtistAnimation(
        fig, ims, interval=100, blit=True, repeat_delay=5000)
    plt.close()

    display.display(display.HTML(ani.to_html5_video()))
  except RuntimeError:
    print('Coult not render animation:')
    traceback.print_exc()


def gol_episode(board):
  directions = tf.constant(
      ((-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)))

  new_board = []
  ag.set_element_type(new_board, tf.int32)

  for i in range(len(board)):
    for j in range(len(board[i])):
      num_neighbors = 0
      for d in directions:
        ni = i + d[0]
```

```
        nj = j + d[1]
        if ni >= 0 and nj >= 0 and ni < len(board) and nj < len(board[i]):
          num_neighbors += board[ni][nj]

      new_cell = 0
      if num_neighbors == 2:
        new_cell = board[i][j]
      elif num_neighbors == 3:
        new_cell = 1

      new_board.append(new_cell)
  final_board = ag.stack(new_board)
  final_board = tf.reshape(final_board, board.shape)
  return final_board


def gol(initial_board):
  board = initial_board
  boards = []
  ag.set_element_type(boards, tf.int32)
  # We are being explicit about tensor constants to ensure the loop
  # is not unrolled in the graph. This may change in the future.
  for i in range(tf.constant(NUM_STEPS)):
    board = gol_episode(board)
    boards.append(board)
  boards = ag.stack(boards)
  render(boards)
  return tf.no_op()


with tf.Graph().as_default():
  # Gosper glider gun
  # Adapted from http://www.cplusplus.com/forum/lounge/75168/
  _ = 0
  initial_board = tf.constant((
      ( _,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_ ),
      ( _,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,1,_,_,_,_,_,_ ),
      ( _,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,1,_,1,_,_,_,_,_ ),
      ( _,_,_,_,_,_,_,_,_,_,_,_,_,1,1,_,_,_,_,_,_,1,1,_,_,_,_,_,_,_,_,_,_,_,1,1,_ ),
      ( _,_,_,_,_,_,_,_,_,_,_,_,1,_,_,_,1,_,_,_,_,1,1,_,_,_,_,_,_,_,_,_,_,_,1,1,_ ),
      ( _,1,1,_,_,_,_,_,_,_,_,1,_,_,_,_,_,1,_,_,_,1,1,_,_,_,_,_,_,_,_,_,_,_,_,_ ),
      ( _,1,1,_,_,_,_,_,_,_,_,1,_,_,_,1,_,1,1,_,_,_,_,1,_,1,_,_,_,_,_,_,_,_,_,_ ),
      ( _,_,_,_,_,_,_,_,_,_,_,1,_,_,_,_,_,_,1,_,_,_,_,_,_,_,1,_,_,_,_,_,_,_,_,_ ),
      ( _,_,_,_,_,_,_,_,_,_,_,1,_,_,_,1,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_ ),
      ( _,_,_,_,_,_,_,_,_,_,_,1,1,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_ ),
      ( _,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_ ),
      ( _,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_ ),
  ))
```

```
          initial_board = tf.pad(initial_board, ((0, 20), (0, 10)))

          tf_gol = ag.to_graph(gol)
          game_ops = tf_gol(initial_board)
          with tf.Session() as sess:
            sess.run(game_ops)
```

```
<IPython.core.display.HTML at 0x7f84b2253b50>
```

Generated code

```
In [0]: print(ag.to_code(gol))
```

```
from __future__ import print_function
import tensorflow as tf

def tf__gol_episode(board):
  try:
    with tf.name_scope('gol_episode'):
      directions = tf.constant(((-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1),
          (1, -1), (1, 0), (1, 1)))
      new_board = ag__.new_list([])

      def extra_test_2(new_board_2):
        with tf.name_scope('extra_test_2'):
          return True

      def loop_body_2(i, new_board_2):
        with tf.name_scope('loop_body_2'):

          def extra_test_1(new_board_1):
            with tf.name_scope('extra_test_1'):
              return True

          def loop_body_1(j, new_board_1):
            with tf.name_scope('loop_body_1'):
              num_neighbors = 0

              def extra_test(num_neighbors_2):
                with tf.name_scope('extra_test'):
                  return True

              def loop_body(d, num_neighbors_2):
                with tf.name_scope('loop_body'):
                  ni = i + ag__.get_item(d, (0), opts=ag__.GetItemOpts(
                      element_dtype=None))
                  nj = j + ag__.get_item(d, (1), opts=ag__.GetItemOpts(
```

```
        element_dtype=None))

    def if_true():
      with tf.name_scope('if_true'):
        num_neighbors_1, = num_neighbors_2,
        num_neighbors_1 += ag__.get_item(ag__.get_item(board,
            (ni), opts=ag__.GetItemOpts(element_dtype=None)),
            (nj), opts=ag__.GetItemOpts(element_dtype=None))
        return num_neighbors_1,

    def if_false():
      with tf.name_scope('if_false'):
        return num_neighbors_2,
    num_neighbors_2 = ag__.utils.run_cond(tf.logical_and(tf.
        greater_equal(ni, 0), tf.logical_and(tf.greater_equal
        (nj, 0), tf.logical_and(tf.less(ni, ag__.utils.
        dynamic_builtin(len, board)), tf.less(nj, ag__.utils.
        dynamic_builtin(len, ag__.get_item(board, (i), opts=
        ag__.GetItemOpts(element_dtype=None)))))), if_true,
        if_false)
    return num_neighbors_2,
num_neighbors = ag__.for_stmt(directions, extra_test,
    loop_body, (num_neighbors,))
new_cell = 0

def if_true_2():
  with tf.name_scope('if_true_2'):
    new_cell_2, = new_cell,
    new_cell_2 = ag__.get_item(ag__.get_item(board, (i), opts
        =ag__.GetItemOpts(element_dtype=None)), (j), opts=
        ag__.GetItemOpts(element_dtype=None))
    return new_cell_2,

def if_false_2():
  with tf.name_scope('if_false_2'):
    new_cell_3, = new_cell,

    def if_true_1():
      with tf.name_scope('if_true_1'):
        new_cell_1, = new_cell_3,
        new_cell_1 = 1
        return new_cell_1,

    def if_false_1():
      with tf.name_scope('if_false_1'):
        return new_cell_3,
    new_cell_3 = ag__.utils.run_cond(tf.equal(num_neighbors,
        3), if_true_1, if_false_1)
```

```
                return new_cell_3,
            new_cell = ag__.utils.run_cond(tf.equal(num_neighbors, 2),
                if_true_2, if_false_2)
            new_board_1 = ag__.list_append(new_board_1, new_cell)
            return new_board_1,
        new_board_2 = ag__.for_stmt(ag__.utils.dynamic_builtin(range,
            ag__.utils.dynamic_builtin(len, ag__.get_item(board, (i),
            opts=ag__.GetItemOpts(element_dtype=None)))), extra_test_1,
            loop_body_1, (new_board_2,))
        return new_board_2,
      new_board = ag__.for_stmt(ag__.utils.dynamic_builtin(range, ag__.
          utils.dynamic_builtin(len, board)), extra_test_2, loop_body_2, (
          new_board,))
      final_board = ag__.list_stack(new_board, opts=ag__.ListStackOpts(
          element_dtype=tf.int32, original_call=ag.stack))
      final_board = tf.reshape(final_board, board.shape)
      return final_board
  except:
    ag__.rewrite_graph_construction_error(ag_source_map__)

def tf__gol(initial_board):
  try:
    with tf.name_scope('gol'):
      board = initial_board
      boards = ag__.new_list([])

      def extra_test(board_1, boards_1):
        with tf.name_scope('extra_test'):
          return True

      def loop_body(i, board_1, boards_1):
        with tf.name_scope('loop_body'):
          board_1 = tf__gol_episode(board_1)
          boards_1 = ag__.list_append(boards_1, board_1)
          return board_1, boards_1
      board, boards = ag__.for_stmt(ag__.utils.dynamic_builtin(range, tf.
          constant(NUM_STEPS)), extra_test, loop_body, (board, boards))
      boards = ag__.list_stack(boards, opts=ag__.ListStackOpts(
          element_dtype=tf.int32, original_call=ag.stack))
      with ag__.utils.control_dependency_on_returns(render(boards)):
        boards_2 = ag__.utils.alias_tensors(boards)
        return tf.no_op()
  except:
    ag__.rewrite_graph_construction_error(ag_source_map__)
```