tf.profiler.Profiler

Contents

Class `Profiler`

Defined in `tensorflow/python/profiler/model_analyzer.py`.

TensorFlow multi-step profiler.

https://github.com/tensorflow/tensorflow/tree/master/tensorflow/core/profiler/README.md

```
Typical use case:
  # Currently we are only allowed to create 1 profiler per process.
  profiler = Profiler(sess.graph)

  for i in xrange(total_steps):
    if i % 10000 == 0:
      run_meta = tf.RunMetadata()
      _ = sess.run(...,
                   options=tf.RunOptions(
                       trace_level=tf.RunOptions.FULL_TRACE),
                   run_metadata=run_meta)
      profiler.add_step(i, run_meta)

      # Profile the parameters of your model.
      profiler.profile_name_scope(options=(option_builder.ProfileOptionBuilder
          .trainable_variables_parameter()))

      # Or profile the timing of your model operations.
      opts = option_builder.ProfileOptionBuilder.time_and_memory()
      profiler.profile_operations(options=opts)

      # Or you can generate a timeline:
      opts = (option_builder.ProfileOptionBuilder(
              option_builder.ProfileOptionBuilder.time_and_memory())
              .with_step(i)
              .with_timeline_output(filename).build())
      profiler.profile_graph(options=opts)
    else:
      _ = sess.run(...)
  # Auto detect problems and generate advice.
  profiler.advise()
```

Methods

`__init__`

```
__init__(
    graph,
    op_log=None
)
```

Constructor.

Args:

- `graph`: tf.Graph.
- `op_log`: optional. tensorflow::tfprof::OpLogProto proto. Used to define extra op types.

`add_step`

```
add_step(
    step,
    run_meta
)
```

Add statistics of a step.

Args:

- `step`: int, A step used to identify the RunMetadata. Must be different across different AddStep() calls.
- `run_meta`: RunMetadata proto that contains statistics of a session run.

`advise`

```
advise(options)
```

Automatically detect problems and generate reports.

Args:

- `options`: A dict of options. See ALL_ADVICE example above.

Returns:

A Advise proto that conains the reports from all checkers.

`profile_graph`

```
profile_graph(options)
```

Profile the statistics of graph nodes, organized by dataflow graph.

Args:

- `options`: A dict of options. See core/profiler/g3doc/options.md.

Returns:

a GraphNodeProto that records the results.

`profile_name_scope`

```
profile_name_scope(options)
```

Profile the statistics of graph nodes, organized by name scope.

Args:

- `options`: A dict of options. See core/profiler/g3doc/options.md.

Returns:

a GraphNodeProto that records the results.

`profile_operations`

```
profile_operations(options)
```

Profile the statistics of the Operation types (e.g. MatMul, Conv2D).

Args:

- `options`: A dict of options. See core/profiler/g3doc/options.md.

Returns:

a MultiGraphNodeProto that records the results.

`profile_python`

```
profile_python(options)
```

Profile the statistics of the Python codes.

By default, it shows the call stack from root. To avoid redundant output, you may use options to filter as below options['show_name_regexes'] = ['*.my_code.py*.']

Args:

- `options`: A dict of options. See core/profiler/g3doc/options.md.

Returns:

a MultiGraphNodeProto that records the results.

Last updated November 2, 2017.