tfdbg.DebugDumpDir

Class **DebugDumpDir**

Defined in `tensorflow/python/debug/lib/debug_data.py`.

See the guide: TensorFlow Debugger > Classes for debug-dump data and directories

Data set from a debug-dump directory on filesystem.

An instance of `DebugDumpDir` contains all `DebugTensorDatum` instances in a tfdbg dump root directory.

Properties

`core_metadata`

Metadata about the `Session.run()` call from the core runtime.

Of the three counters available in the return value, `global_step` is supplied by the caller of the debugged `Session.run()`, while `session_run_index` and `executor_step_index` are determined by the state of the core runtime, automatically. For the same fetch list, feed keys and debug tensor watch options, the same executor will be used and `executor_step_index` should increase by one at a time. However, runs with different fetch lists, feed keys and debug_tensor watch options that all share the same `Session` object can lead to gaps in `session_run_index`.

Returns:

If core metadata are loaded, a `namedtuple` with the fields: `global_step`: A global step count supplied by the caller of `Session.run()`. It is optional to the caller. If the caller did not supply this parameter, its value will be -1. `session_run_index`: A sorted index for Run() calls to the underlying TensorFlow `Session` object. `executor_step_index`: A counter for invocations of a given runtime executor. The same executor is re-used for the same fetched tensors, target nodes, input feed keys and debug tensor watch options. `input_names`: Names of the input (feed) Tensors. `output_names`: Names of the output (fetched) Tensors. `target_nodes`: Names of the target nodes. If the core metadata have not been loaded, `None`. If more than one core metadata files exist, return a list of the `nametuple` described above.

`dumped_tensor_data`

Retrieve dumped tensor data.

`python_graph`

Get the Python graph.

Returns:

If the Python graph has been set, returns a `tf.Graph` object. Otherwise, returns None.

`run_feed_keys_info`

Get a str representation of the feed_dict used in the Session.run() call.

Returns:

If the information is available from one `Session.run` call, a `str` obtained from `repr(feed_dict)`. If the information is available from multiple `Session.run` calls, a `list` of `str` obtained from `repr(feed_dict)`. If the information is not available, `None`.

`run_fetches_info`

Get a str representation of the fetches used in the Session.run() call.

Returns:

If the information is available from one `Session.run` call, a `str` obtained from `repr(fetches)`. If the information is available from multiple `Session.run` calls, a `list` of `str` from `repr(fetches)`. If the information is not available, `None`.

`size`

Total number of dumped tensors in the dump root directory.

Returns:

(`int`) The total number of dumped tensors in the dump root directory.

`t0`

Absolute timestamp of the first dumped tensor across all devices.

Returns:

(`int`) absolute timestamp of the first dumped tensor, in microseconds.

Methods

`__init__`

```
__init__(
    dump_root,
    partition_graphs=None,
    validate=True
)
```

`DebugDumpDir` constructor.

Args:

- `dump_root`: (`str`) path to the dump root directory.
- `partition_graphs`: A repeated field of GraphDefs representing the partition graphs executed by the TensorFlow runtime.
- `validate`: (`bool`) whether the dump files are to be validated against the partition graphs.

Raises:

- `IOError`: If dump_root does not exist as a directory.
- `ValueError`: If more than one core metadata file is found under the dump root directory.

`debug_watch_keys`

```
debug_watch_keys(
    node_name,
    device_name=None
)
```

Get all tensor watch keys of given node according to partition graphs.

Args:

- `node_name`: (`str`) name of the node.
- `device_name`: (`str`) name of the device. If there is only one device or if node_name exists on only one device, this argument is optional.

Returns:

(`list` of `str`) all debug tensor watch keys. Returns an empty list if the node name does not correspond to any debug watch keys.

Raises:

`LookupError`: If debug watch information has not been loaded from partition graphs yet.

`devices`

```
devices()
```

Get the list of device names.

Returns:

(`list` of `str`) names of the devices.

`find`

```
find(
    predicate,
    first_n=0,
    device_name=None
)
```

Find dumped tensor data by a certain predicate.

Args:

- `predicate`: A callable that takes two input arguments:

  `python def predicate(debug_tensor_datum, tensor): # returns a bool`

  where `debug_tensor_datum` is an instance of `DebugTensorDatum`, which carries the metadata, such as the `Tensor`'s node name, output slot timestamp, debug op name, etc.; and `tensor` is the dumped tensor value as a `numpy.ndarray`. `first_n`: (`int`) *return only the first n* `DebugTensotDatum` *instances (in time order) for which the predicate returns True. To return all the* `DebugTensotDatum` *instances, let first_n be <= 0.* `device_name`: optional device name.

Returns:

A list of all `DebugTensorDatum` objects in this `DebugDumpDir` object for which predicate returns True, sorted in ascending order of the timestamp.

`find_some_path`

```
find_some_path(
    src_node_name,
    dst_node_name,
    include_control=True,
    include_reversed_ref=False,
    device_name=None
)
```

Find a path between a source node and a destination node.

Limitation: the source and destination are required to be on the same device, i.e., this method does not yet take into account Send/Recv nodes across devices.

TODO(cais): Make this method work across device edges by tracing Send/Recv nodes.

Args:

- `src_node_name`: (`str`) name of the source node or name of an output tensor of the node.
- `dst_node_name`: (`str`) name of the destination node or name of an output tensor of the node.
- `include_control`: (`bool`) whrther control edges are considered in the graph tracing.
- `include_reversed_ref`: Whether a ref input, say from A to B, is to be also considered as an input from B to A. The rationale is that ref inputs generally let the recipient (e.g., B in this case) mutate the value of the source (e.g., A in this case). So the reverse direction of the ref edge reflects the direction of information flow.
- `device_name`: (`str`) name of the device. If there is only one device or if node_name exists on only one device, this argument is optional.

Returns:

A path from the src_node_name to dst_node_name, as a `list` of `str`, if it exists. The list includes src_node_name as the first item and dst_node_name as the last. If such a path does not exist, `None`.

Raises:

- `ValueError`: If the source and destination nodes are not on the same device.

`get_dump_sizes_bytes`

```
get_dump_sizes_bytes(
    node_name,
    output_slot,
    debug_op,
    device_name=None
)
```

Get the sizes of the dump files for a debug-dumped tensor.

Unit of the file size: byte.

Args:

- `node_name`: (`str`) name of the node that the tensor is produced by.
- `output_slot`: (`int`) output slot index of tensor.
- `debug_op`: (`str`) name of the debug op.
- `device_name`: (`str`) name of the device. If there is only one device or if the specified debug_watch_key exists on only one device, this argument is optional.

Returns:

(`list` of `int`): list of dump file sizes in bytes.

Raises:

- `WatchKeyDoesNotExistInDebugDumpDirError`: If the tensor watch key does not exist in the debug dump data.

`get_rel_timestamps`

```
get_rel_timestamps(
    node_name,
    output_slot,
    debug_op,
    device_name=None
)
```

Get the relative timestamp from for a debug-dumped tensor.

Relative timestamp means (absolute timestamp - `t0`), where `t0` is the absolute timestamp of the first dumped tensor in the dump root. The tensor may be dumped multiple times in the dump root directory, so a list of relative timestamps (`numpy.ndarray`) is returned.

Args:

- `node_name` : (**`str`**) name of the node that the tensor is produced by.
- `output_slot` : (**`int`**) output slot index of tensor.
- `debug_op` : (**`str`**) name of the debug op.
- `device_name` : (**`str`**) name of the device. If there is only one device or if the specified debug_watch_key exists on only one device, this argument is optional.

Returns:

(**`list`** of **`int`**) list of relative timestamps.

Raises:

- `WatchKeyDoesNotExistInDebugDumpDirError` : If the tensor watch key does not exist in the debug dump data.

`get_tensor_file_paths`

```
get_tensor_file_paths(
    node_name,
    output_slot,
    debug_op,
    device_name=None
)
```

Get the file paths from a debug-dumped tensor.

Args:

- `node_name` : (**`str`**) name of the node that the tensor is produced by.
- `output_slot` : (**`int`**) output slot index of tensor.
- `debug_op` : (**`str`**) name of the debug op.
- `device_name` : (**`str`**) name of the device. If there is only one device or if the specified debug_watch_key exists on only one device, this argument is optional.

Returns:

List of file path(s) loaded. This is a list because each debugged tensor may be dumped multiple times.

Raises:

- `WatchKeyDoesNotExistInDebugDumpDirError` : If the tensor does not exist in the debug-dump data.

`get_tensors`

```
get_tensors(
    node_name,
    output_slot,
    debug_op,
    device_name=None
)
```

Get the tensor value from for a debug-dumped tensor.

The tensor may be dumped multiple times in the dump root directory, so a list of tensors (`numpy.ndarray`) is returned.

Args:

- `node_name`: (`str`) name of the node that the tensor is produced by.
- `output_slot`: (`int`) output slot index of tensor.
- `debug_op`: (`str`) name of the debug op.
- `device_name`: (`str`) name of the device. If there is only one device or if the specified debug_watch_key exists on only one device, this argument is optional.

Returns:

List of tensors (`numpy.ndarray`) loaded from the debug-dump file(s).

Raises:

- `WatchKeyDoesNotExistInDebugDumpDirError`: If the tensor does not exist in the debug-dump data.

`loaded_partition_graphs`

```
loaded_partition_graphs()
```

Test whether partition graphs have been loaded.

`node_attributes`

```
node_attributes(
    node_name,
    device_name=None
)
```

Get the attributes of a node.

Args:

- `node_name`: Name of the node in question.
- `device_name`: (`str`) name of the device. If there is only one device or if node_name exists on only one device, this argument is optional.

Returns:

Attributes of the node.

Raises:

- `LookupError`: If no partition graphs have been loaded.

`node_device`

```
node_device(node_name)
```

Get the names of the devices that has nodes of the specified name.

Args:

- `node_name` : (`str`) name of the node.

Returns:

(`str` or `list` of `str`) name of the device(s) on which the node of the given name is found. Returns a `str` if there is only one such device, otherwise return a `list` of `str` .

Raises:

- `LookupError` : If node inputs and control inputs have not been loaded from partition graphs yet.
- `ValueError` : If the node does not exist in partition graphs.

`node_exists`

```
node_exists(
    node_name,
    device_name=None
)
```

Test if a node exists in the partition graphs.

Args:

- `node_name` : (`str`) name of the node to be checked.
- `device_name` : optional device name. If None, will search for the node on all available devices. Otherwise, search for the node only on the given device.

Returns:

A boolean indicating whether the node exists.

Raises:

- `LookupError` : If no partition graphs have been loaded yet.
- `ValueError` : If device_name is specified but cannot be found.

`node_inputs`

```
node_inputs(
    node_name,
    is_control=False,
    device_name=None
)
```

Get the inputs of given node according to partition graphs.

Args:

- `node_name`: Name of the node.
- `is_control`: (`bool`) Whether control inputs, rather than non-control inputs, are to be returned.
- `device_name`: (`str`) name of the device. If there is only one device or if node_name exists on only one device, this argument is optional.

Returns:

(`list` of `str`) inputs to the node, as a list of node names.

Raises:

- `LookupError`: If node inputs and control inputs have not been loaded from partition graphs yet.

`node_op_type`

```
node_op_type(
    node_name,
    device_name=None
)
```

Get the op type of given node.

Args:

- `node_name`: (`str`) name of the node.
- `device_name`: (`str`) name of the device. If there is only one device or if node_name exists on only one device, this argument is optional.

Returns:

(`str`) op type of the node.

Raises:

- `LookupError`: If node op types have not been loaded from partition graphs yet.

`node_recipients`

```
node_recipients(
    node_name,
    is_control=False,
    device_name=None
)
```

Get recipient of the given node's output according to partition graphs.

Args:

- `node_name`: (`str`) name of the node.

- `is_control` : (**`bool`**) whether control outputs, rather than non-control outputs, are to be returned.
- `device_name` : (**`str`**) name of the device. If there is only one device or if node_name exists on only one device, this argument is optional.

Returns:

(**`list`** of **`str`**) all inputs to the node, as a list of node names.

Raises:

- `LookupError` : If node inputs and control inputs have not been loaded from partition graphs yet.

node_traceback

```
node_traceback(element_name)
```

Try to retrieve the Python traceback of node's construction.

Args:

- `element_name` : (**`str`**) Name of a graph element (node or tensor).

Returns:

(list) The traceback list object as returned by the **`extract_trace`** method of Python's traceback module.

Raises:

- `LookupError` : If Python graph is not available for traceback lookup.
- `KeyError` : If the node cannot be found in the Python graph loaded.

nodes

```
nodes(device_name=None)
```

Get a list of all nodes from the partition graphs.

Args:

- `device_name` : (**`str`**) name of device. If None, all nodes from all available devices will be included.

Returns:

All nodes' names, as a list of str.

Raises:

- `LookupError` : If no partition graphs have been loaded.
- `ValueError` : If specified node name does not exist.

`partition_graphs`

```
partition_graphs()
```

Get the partition graphs.

Returns:

Partition graphs as a list of GraphDef.

Raises:

- `LookupError`: If no partition graphs have been loaded.

`reconstructed_non_debug_partition_graphs`

```
reconstructed_non_debug_partition_graphs()
```

Reconstruct partition graphs with the debugger-inserted ops stripped.

The reconstructed partition graphs are identical to the original (i.e., non-debugger-decorated) partition graphs except in the following respects: 1) The exact names of the runtime-inserted internal nodes may differ. These include _Send, _Recv, _HostSend, _HostRecv, _Retval ops. 2) As a consequence of 1, the nodes that receive input directly from such send- and recv-type ops will have different input names. 3) The parallel_iteration attribute of while-loop Enter ops are set to 1.

Returns:

A dict mapping device names (`str` s) to reconstructed `tf.GraphDef` s.

`set_python_graph`

```
set_python_graph(python_graph)
```

Provide Python `Graph` object to the wrapper.

Unlike the partition graphs, which are protobuf `GraphDef` objects, `Graph` is a Python object and carries additional information such as the traceback of the construction of the nodes in the graph.

Args:

- `python_graph`: (ops.Graph) The Python Graph object.

`transitive_inputs`

```
transitive_inputs(
    node_name,
    include_control=True,
    include_reversed_ref=False,
    device_name=None
)
```

Get the transitive inputs of given node according to partition graphs.

Args:

- `node_name`: Name of the node.
- `include_control`: Include control inputs (True by default).
- `include_reversed_ref`: Whether a ref input, say from A to B, is to be also considered as an input from B to A. The rationale is that ref inputs generally let the recipient (e.g., B in this case) mutate the value of the source (e.g., A in this case). So the reverse direction of the ref edge reflects the direction of information flow.
- `device_name`: (`str`) name of the device. If there is only one device or if node_name exists on only one device, this argument is optional.

Returns:

(`list` of `str`) all transitive inputs to the node, as a list of node names.

Raises:

- `LookupError`: If node inputs and control inputs have not been loaded from partition graphs yet.

`watch_key_to_data`

```
watch_key_to_data(
    debug_watch_key,
    device_name=None
)
```

Get all `DebugTensorDatum` instances corresponding to a debug watch key.

Args:

- `debug_watch_key`: (`str`) debug watch key.
- `device_name`: (`str`) name of the device. If there is only one device or if the specified debug_watch_key exists on only one device, this argument is optional.

Returns:

A list of `DebugTensorDatum` instances that correspond to the debug watch key. If the watch key does not exist, returns an empty list.

Raises:

- `ValueError`: If there are multiple devices that have the debug_watch_key, but device_name is not specified.

GitHub

Twitter

Support

Issue Tracker

Release Notes

Stack Overflow

English

Terms | **Privacy**