tf.data.Iterator

Class `Iterator`

Aliases:

- Class `tf.contrib.data.Iterator`
- Class `tf.data.Iterator`

Defined in `tensorflow/python/data/ops/iterator_ops.py`.

Represents the state of iterating through a `Dataset`.

Properties

`initializer`

A `tf.Operation` that should be run to initialize this iterator.

Returns:

A `tf.Operation` that should be run to initialize this iterator

Raises:

- `ValueError`: If this iterator initializes itself automatically.

`output_shapes`

Returns the shape of each component of an element of this iterator.

Returns:

A nested structure of `tf.TensorShape` objects corresponding to each component of an element of this iterator.

`output_types`

Returns the type of each component of an element of this iterator.

Returns:

A nested structure of `tf.DType` objects corresponding to each component of an element of this iterator.

Methods

`__init__`

```
__init__(
    iterator_resource,
    initializer,
    output_types,
    output_shapes
)
```

Creates a new iterator from the given iterator resource.

> **Note:** Most users will not call this initializer directly, and will instead use `Dataset.make_initializable_iterator()` or `Dataset.make_one_shot_iterator()`.

Args:

- `iterator_resource`: A `tf.resource` scalar `tf.Tensor` representing the iterator.
- `initializer`: A `tf.Operation` that should be run to initialize this iterator.
- `output_types`: A nested structure of `tf.DType` objects corresponding to each component of an element of this iterator.
- `output_shapes`: A nested structure of `tf.TensorShape` objects corresponding to each component of an element of this dataset.

`from_string_handle`

```
@staticmethod
from_string_handle(
    string_handle,
    output_types,
    output_shapes=None
)
```

Creates a new, uninitialized `Iterator` based on the given handle.

This method allows you to define a "feedable" iterator where you can choose between concrete iterators by feeding a value in a `tf.Session.run` call. In that case, `string_handle` would a `tf.placeholder`, and you would feed it with the value of `tf.data.Iterator.string_handle` in each step.

For example, if you had two iterators that marked the current position in a training dataset and a test dataset, you could choose which to use in each step as follows:

```
train_iterator = tf.data.Dataset(...).make_one_shot_iterator()
train_iterator_handle = sess.run(train_iterator.string_handle())

test_iterator = tf.data.Dataset(...).make_one_shot_iterator()
test_iterator_handle = sess.run(test_iterator.string_handle())

handle = tf.placeholder(tf.string, shape=[])
iterator = tf.data.Iterator.from_string_handle(
    handle, train_iterator.output_types)

next_element = iterator.get_next()
loss = f(next_element)

train_loss = sess.run(loss, feed_dict={handle: train_iterator_handle})
test_loss = sess.run(loss, feed_dict={handle: test_iterator_handle})
```

Args:

- `string_handle`: A scalar `tf.Tensor` of type `tf.string` that evaluates to a handle produced by the `Iterator.string_handle()` method.
- `output_types`: A nested structure of `tf.DType` objects corresponding to each component of an element of this iterator.
- `output_shapes`: (Optional.) A nested structure of `tf.TensorShape` objects corresponding to each component of an element of this dataset. If omitted, each component will have an unconstrainted shape.

Returns:

An `Iterator`.

`from_structure`

```
@staticmethod
from_structure(
    output_types,
    output_shapes=None,
    shared_name=None
)
```

Creates a new, uninitialized `Iterator` with the given structure.

This iterator-constructing method can be used to create an iterator that is reusable with many different datasets.

The returned iterator is not bound to a particular dataset, and it has no `initializer`. To initialize the iterator, run the operation returned by `Iterator.make_initializer(dataset)`.

The following is an example

```
iterator = Iterator.from_structure(tf.int64, tf.TensorShape([]))

dataset_range = Dataset.range(10)
range_initializer = iterator.make_initializer(dataset_range)

dataset_evens = dataset_range.filter(lambda x: x % 2 == 0)
evens_initializer = iterator.make_initializer(dataset_evens)

# Define a model based on the iterator; in this example, the model_fn
# is expected to take scalar tf.int64 Tensors as input (see
# the definition of 'iterator' above).
prediction, loss = model_fn(iterator.get_next())

# Train for `num_epochs`, where for each epoch, we first iterate over
# dataset_range, and then iterate over dataset_evens.
for _ in range(num_epochs):
  # Initialize the iterator to `dataset_range`
  sess.run(range_initializer)
  while True:
    try:
      pred, loss_val = sess.run([prediction, loss])
    except tf.errors.OutOfRangeError:
      break

  # Initialize the iterator to `dataset_evens`
  sess.run(evens_initializer)
  while True:
    try:
      pred, loss_val = sess.run([prediction, loss])
    except tf.errors.OutOfRangeError:
      break
```

Args:

- `output_types`: A nested structure of **`tf.DType`** objects corresponding to each component of an element of this iterator.
- `output_shapes`: (Optional.) A nested structure of **`tf.TensorShape`** objects corresponding to each component of an element of this dataset. If omitted, each component will have an unconstrainted shape.
- `shared_name`: (Optional.) If non-empty, this iterator will be shared under the given name across multiple sessions that share the same devices (e.g. when using a remote server).

Returns:

An **`Iterator`**.

Raises:

- `TypeError`: If the structures of **`output_shapes`** and **`output_types`** are not the same.

`get_next`

```
get_next(name=None)
```

Returns a nested structure of **`tf.Tensor`**s containing the next element.

Args:

- `name`: (Optional.) A name for the created operation.

Returns:

A nested structure of `tf.Tensor` objects.

`make_initializer`

```
make_initializer(
    dataset,
    name=None
)
```

Returns a `tf.Operation` that initializes this iterator on `dataset`.

Args:

- `dataset`: A `Dataset` with compatible structure to this iterator.
- `name`: (Optional.) A name for the created operation.

Returns:

A `tf.Operation` that can be run to initialize this iterator on the given `dataset`.

Raises:

- `TypeError`: If `dataset` and this iterator do not have a compatible element structure.

`string_handle`

```
string_handle(name=None)
```

Returns a string-valued `tf.Tensor` that represents this iterator.

Args:

- `name`: (Optional.) A name for the created operation.

Returns:

A scalar `tf.Tensor` of type `tf.string`.

GitHub

Twitter

Support

Issue Tracker

Release Notes

Stack Overflow

English

Terms | **Privacy**