TensorFlow

TensorFlow API r1.4

tf.TensorArray

Class `TensorArray`

Defined in `tensorflow/python/ops/tensor_array_ops.py`.

Class wrapping dynamic-sized, per-time-step, write-once Tensor arrays.

This class is meant to be used with dynamic iteration primitives such as `while_loop` and `map_fn`. It supports gradient back-propagation via special "flow" control flow dependencies.

Properties

`dtype`

The data type of this TensorArray.

`flow`

The flow `Tensor` forcing ops leading to this TensorArray state.

`handle`

The reference to the TensorArray.

Methods

`__init__`

```
__init__(
    dtype,
    size=None,
    dynamic_size=None,
    clear_after_read=None,
    tensor_array_name=None,
    handle=None,
    flow=None,
    infer_shape=True,
    element_shape=None,
    colocate_with_first_write_call=True,
    name=None
)
```

Construct a new TensorArray or wrap an existing TensorArray handle.

A note about the parameter `name` :

The name of the `TensorArray` (even if passed in) is uniquified: each time a new `TensorArray` is created at runtime it is assigned its own name for the duration of the run. This avoids name collisions if a `TensorArray` is created within a `while_loop` .

Args:

- `dtype` : (required) data type of the TensorArray.
- `size` : (optional) int32 scalar `Tensor` : the size of the TensorArray. Required if handle is not provided.
- `dynamic_size` : (optional) Python bool: If true, writes to the TensorArray can grow the TensorArray past its initial size. Default: False.
- `clear_after_read` : Boolean (optional, default: True). If True, clear TensorArray values after reading them. This disables read-many semantics, but allows early release of memory.
- `tensor_array_name` : (optional) Python string: the name of the TensorArray. This is used when creating the TensorArray handle. If this value is set, handle should be None.
- `handle` : (optional) A `Tensor` handle to an existing TensorArray. If this is set, tensor_array_name should be None.
- `flow` : (optional) A float `Tensor` scalar coming from an existing `TensorArray.flow` .
- `infer_shape` : (optional, default: True) If True, shape inference is enabled. In this case, all elements must have the same shape.
- `element_shape` : (optional, default: None) A `TensorShape` object specifying the shape constraints of each of the elements of the TensorArray. Need not be fully defined.
- `colocate_with_first_write_call` : If `True` , the TensorArray will be colocated on the same device as the Tensor used on its first write (write operations include `write` , `unstack` , and `split`). If `False` , the TensorArray will be placed on the device determined by the device context available during its initialization.
- `name` : A name for the operation (optional).

Raises:

- `ValueError` : if both handle and tensor_array_name are provided.
- `TypeError` : if handle is provided but is not a Tensor.

`close`

```
close(name=None)
```

Close the current TensorArray.

NOTE The output of this function should be used. If it is not, a warning will be logged. To mark the output as used, call its .mark_used() method.

`concat`

```
concat(name=None)
```

Return the values in the TensorArray as a concatenated `Tensor`.

All of the values must have been written, their ranks must match, and and their shapes must all match for all dimensions except the first.

Args:

- `name`: A name for the operation (optional).

Returns:

All the tensors in the TensorArray concatenated into one tensor.

`gather`

```
gather(
    indices,
    name=None
)
```

Return selected values in the TensorArray as a packed `Tensor`.

All of selected values must have been written and their shapes must all match.

Args:

- `indices`: A `1-D` `Tensor` taking values in `[0, max_value)`. If the `TensorArray` is not dynamic, `max_value=size()`.
- `name`: A name for the operation (optional).

Returns:

The in the `TensorArray` selected by `indices`, packed into one tensor.

`grad`

```
grad(
    source,
    flow=None,
    name=None
)
```

`identity`

```
identity()
```

Returns a TensorArray with the same content and properties.

Returns:

A new TensorArray object with flow that ensures the control dependencies from the contexts will become control dependencies for writes, reads, etc. Use this object all for subsequent operations.

`read`

```
read(
    index,
    name=None
)
```

Read the value at location `index` in the TensorArray.

Args:

- `index`: 0-D. int32 tensor with the index to read from.
- `name`: A name for the operation (optional).

Returns:

The tensor at index `index`.

`scatter`

```
scatter(
    indices,
    value,
    name=None
)
```

Scatter the values of a `Tensor` in specific indices of a `TensorArray`.

Args: indices: A `1-D` `Tensor` taking values in `[0, max_value)`. If the `TensorArray` is not dynamic, `max_value=size()`. value: (N+1)-D. Tensor of type `dtype`. The Tensor to unpack. name: A name for the operation (optional).

Returns: A new TensorArray object with flow that ensures the scatter occurs. Use this object all for subsequent operations.

Raises: ValueError: if the shape inference fails.

NOTE The output of this function should be used. If it is not, a warning will be logged. To mark the output as used, call its .mark_used() method.

`size`

```
size(name=None)
```

Return the size of the TensorArray.

`split`

```
split(
    value,
    lengths,
    name=None
)
```

Split the values of a `Tensor` into the TensorArray.

Args: value: (N+1)-D. Tensor of type `dtype`. The Tensor to split. lengths: 1-D. int32 vector with the lengths to use when splitting `value` along its first dimension. name: A name for the operation (optional).

Returns: A new TensorArray object with flow that ensures the split occurs. Use this object all for subsequent operations.

Raises: ValueError: if the shape inference fails.

NOTE The output of this function should be used. If it is not, a warning will be logged. To mark the output as used, call its .mark_used() method.

`stack`

```
stack(name=None)
```

Return the values in the TensorArray as a stacked `Tensor`.

All of the values must have been written and their shapes must all match. If input shapes have rank-`R`, then output shape will have rank-`(R+1)`.

Args:

- `name`: A name for the operation (optional).

Returns:

All the tensors in the TensorArray stacked into one tensor.

`unstack`

```
unstack(
    value,
    name=None
)
```

Unstack the values of a `Tensor` in the TensorArray.

If input value shapes have rank-`R`, then the output TensorArray will contain elements whose shapes are rank-`(R-1)`.

Args: value: (N+1)-D. Tensor of type `dtype`. The Tensor to unstack. name: A name for the operation (optional).

Returns: A new TensorArray object with flow that ensures the unstack occurs. Use this object all for subsequent operations.

Raises: ValueError: if the shape inference fails.

NOTE The output of this function should be used. If it is not, a warning will be logged. To mark the output as used, call its .mark_used() method.

`write`

```
write(
    index,
    value,
    name=None
)
```

Write `value` into index `index` of the TensorArray.

Args: index: 0-D. int32 scalar with the index to write to. value: N-D. Tensor of type `dtype`. The Tensor to write to this index. name: A name for the operation (optional).

Returns: A new TensorArray object with flow that ensures the write occurs. Use this object all for subsequent operations.

Raises: ValueError: if there are more writers than specified.

NOTE The output of this function should be used. If it is not, a warning will be logged. To mark the output as used, call its .mark_used() method.

Last updated November 2, 2017.