tf.contrib.kfac.layer_collection.LayerCollection

Contents

Class **`LayerCollection`**

Defined in `tensorflow/contrib/kfac/python/ops/layer_collection.py`.

Registry of information about layers and losses.

Note that you need to create a new one of these for each MatrixEstimator or KfacOptimizer.

Attributes:

- `fisher_blocks` : a LayersParamsDict (subclass of OrderedDict) mapping layer parameters (Tensors or tuples of Tensors) to FisherBlock instances.
- `fisher_factors` : an OrderedDict mapping tuples to FisherFactor instances.
- `generic_registrations` : a list of variables registered via a generic layer registration. Generic registrations handle any and all of the ways a variable is used in the graph, which means we don't need to check their registration when verifying the correctness of the graph.
- `losses` : a list of LossFunction objects. The loss to be optimized is their sum.

Properties

`generic_registrations`

`graph`

`subgraph`

Methods

`__init__`

```
__init__(
    graph=None,
    name='LayerCollection'
)
```

`create_subgraph`

```
create_subgraph()
```

`get_blocks`

```
get_blocks()
```

`get_factors`

```
get_factors()
```

`get_use_count_map`

```
get_use_count_map()
```

Returns a dict of variables to their number of registrations.

`make_or_get_factor`

```
make_or_get_factor(
    cls,
    args
)
```

`register_block`

```
register_block(
    layer_key,
    fisher_block
)
```

Validates and registers the layer_key associated with the fisher_block.

Validation consists of checking whether the key was already registered or if any of the elements of layer_key (if it's a tuple) were already registered as part of another tuple (throws an error if so). If any of the elements were registered by themselves, or as part of tuples that are subsets of this layer_key, those registrations are first removed.

If the layer_key is a subset of an existing registration, registration of the new, smaller layer_key is skipped.

e.g. If registrations include {'a': foo, ('b', 'c'): bar}, then - register_layer('a', baz) -> ValueError - register_layer(('b', 'c', 'd'), baz) -> {'a': foo, ('b', 'c', 'd'): baz} - register_layer('b', baz) -> {'a': foo, ('b', 'c'): bar} (No change) - register_layer(('a', 'd'), baz) -> {('a', 'd'): baz, ('b', 'c'): bar} - register_layer(('b', 'd'), baz) -> ValueError

Args:

- `layer_key`: The key to check for in existing registrations and to register if valid.
- `fisher_block`: The associated fisher block.

Raises:

- `ValueError` : If the layer_key was already registered, or if a subset of the layer_key has already been registered as part of a different tuple.

`register_categorical_predictive_distribution`

```
register_categorical_predictive_distribution(
    logits,
    seed=None,
    targets=None
)
```

Registers a categorical predictive distribution.

Args:

- `logits` : The logits of the distribution (i.e. its parameters).
- `seed` : The seed for the RNG (for debugging) (Default: None)
- `targets` : (OPTIONAL) The targets for the loss function. Only required if one wants to call total_loss() instead of total_sampled_loss(). total_loss() is required, for example, to estimate the "empirical Fisher" (instead of the true Fisher). (Default: None)

`register_conv2d`

```
register_conv2d(
    params,
    strides,
    padding,
    inputs,
    outputs,
    approx=APPROX_KRONECKER_NAME
)
```

`register_fully_connected`

```
register_fully_connected(
    params,
    inputs,
    outputs,
    approx=APPROX_KRONECKER_NAME
)
```

`register_generic`

```
register_generic(
    params,
    batch_size,
    approx=APPROX_DIAGONAL_NAME
)
```

`register_multi_bernoulli_predictive_distribution`

```
register_multi_bernoulli_predictive_distribution(
    logits,
    seed=None,
    targets=None
)
```

Registers a multi-Bernoulli predictive distribution.

Args:

- `logits`: The logits of the distribution (i.e. its parameters).
- `seed`: The seed for the RNG (for debugging) (Default: None)
- `targets`: (OPTIONAL) The targets for the loss function. Only required if one wants to call total_loss() instead of total_sampled_loss(). total_loss() is required, for example, to estimate the "empirical Fisher" (instead of the true Fisher). (Default: None)

`register_normal_predictive_distribution`

```
register_normal_predictive_distribution(
    mean,
    var=0.5,
    seed=None,
    targets=None
)
```

Registers a normal predictive distribution.

Args:

- `mean`: The mean vector defining the distribution.
- `var`: The variance (must be a scalar). Note that the default value of 0.5 corresponds to a standard squared error loss (target - prediction)**2. If your squared error loss is of the form 0.5*(target - prediction)**2 you should use var=1.0. (Default: 0.5)
- `seed`: The seed for the RNG (for debugging) (Default: None)
- `targets`: (OPTIONAL) The targets for the loss function. Only required if one wants to call total_loss() instead of total_sampled_loss(). total_loss() is required, for example, to estimate the "empirical Fisher" (instead of the true Fisher). (Default: None)

`reset_internals`

```
reset_internals(
    graph=None,
    name='LayerCollection'
)
```

`total_loss`

```
total_loss()
```

`total_sampled_loss`

```
total_sampled_loss()
```

Last updated November 2, 2017.

Stay Connected

Blog

GitHub

Twitter

Support

Issue Tracker

Release Notes

Stack Overflow

English

Terms | **Privacy**