tf.contrib.kfac.estimator.FisherEstimator

Contents

Class **`FisherEstimator`**

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

Fisher estimator class supporting various approximations of the Fisher.

Properties

`damping`

`variables`

Methods

`__init__`

```
__init__(
    variables,
    cov_ema_decay,
    damping,
    layer_collection,
    estimation_mode='gradients'
)
```

Create a FisherEstimator object.

Args:

- `variables`: A list of the variables for which to estimate the Fisher. This must match the variables registered in layer_collection (if it is not None).
- `cov_ema_decay`: The decay factor used when calculating the covariance estimate moving averages.
- `damping`: The damping factor used to stabilize training due to errors in the local approximation with the Fisher information matrix, and to regularize the update direction by making it closer to the gradient. (Higher damping means the update looks more like a standard gradient update - see Tikhonov regularization.)
- `layer_collection`: The layer collection object, which holds the fisher blocks, kronecker factors, and losses associated with the graph.

- `estimation_mode` : The type of estimator to use for the Fishers. Can be 'gradients', 'empirical', 'curvature_propagation', or 'exact'. (Default: 'gradients'). 'gradients' is the basic estimation approach from the original K-FAC paper. 'empirical' computes the 'empirical' Fisher information matrix (which uses the data's distribution for the targets, as opposed to the true Fisher which uses the model's distribution) and requires that each registered loss have specified targets. 'curvature_propagation' is a method which estimates the Fisher using self-products of random 1/-1 vectors times "half-factors" of the Fisher, as described here: https://arxiv.org/abs/1206.6464 . Finally, 'exact' is the obvious generalization of Curvature Propagation to compute the exact Fisher (modulo any additional diagonal or Kronecker approximations) by looping over one-hot vectors for each coordinate of the output instead of using 1/-1 vectors. It is more expensive to compute than the other three options by a factor equal to the output dimension, roughly speaking.

Raises:

- `ValueError` : If no losses have been registered with layer_collection.

`multiply`

```
multiply(vecs_and_vars)
```

Multiplies the vectors by the corresponding (damped) blocks.

Args:

- `vecs_and_vars` : List of (vector, variable) pairs.

Returns:

A list of (transformed vector, var) pairs in the same order as vecs_and_vars.

`multiply_inverse`

```
multiply_inverse(vecs_and_vars)
```

Multiplies the vecs by the corresponding (damped) inverses of the blocks.

Args:

- `vecs_and_vars` : List of (vector, variable) pairs.

Returns:

A list of (transformed vector, var) pairs in the same order as vecs_and_vars.

Last updated November 2, 2017.