tf.contrib.opt.MovingAverageOptimizer

Contents

Class `MovingAverageOptimizer`

Inherits From: `Optimizer`

Defined in `tensorflow/contrib/opt/python/training/moving_average_optimizer.py`.

Optimizer that computes a moving average of the variables.

Empirically it has been found that using the moving average of the trained parameters of a deep network is better than using its trained parameters directly. This optimizer allows you to compute this moving average and swap the variables at save time so that any code outside of the training loop will use by default the averaged values instead of the original ones.

Example of usage:

```
// Encapsulate your favorite optimizer (here the momentum one)
// inside the MovingAverageOptimizer.
opt = tf.train.MomentumOptimizer(learning_rate, FLAGS.momentum)
opt = tf.contrib.opt.MovingAverageOptimizer(opt)
// Then create your model and all its variables.
model = build_model()
// Add the training op that optimizes using opt.
// This needs to be called before swapping_saver().
opt.minimize(cost, var_list)
// Then create your saver like this:
saver = opt.swapping_saver()
// Pass it to your training loop.
    slim.learning.train(
        model,
        ...
        saver=saver)
```

Note that for evaluation, the normal saver should be used instead of swapping_saver().

Methods

`__init__`

```
__init__(
    opt,
    average_decay=0.9999,
    num_updates=None,
    sequential_update=True
)
```

Construct a new MovingAverageOptimizer.

Args:

- `opt`: A tf.Optimizer that will be used to compute and apply gradients.
- `average_decay`: Float. Decay to use to maintain the moving averages of trained variables. See tf.train.ExponentialMovingAverage for details.
- `num_updates`: Optional count of number of updates applied to variables. See tf.train.ExponentialMovingAverage for details.
- `sequential_update`: Bool. If False, will compute the moving average at the same time as the model is updated, potentially doing benign data races. If True, will update the moving average after gradient updates.

apply_gradients

```
apply_gradients(
    grads_and_vars,
    global_step=None,
    name=None
)
```

compute_gradients

```
compute_gradients(
    loss,
    var_list=None,
    gate_gradients=GATE_OP,
    aggregation_method=None,
    colocate_gradients_with_ops=False,
    grad_loss=None
)
```

Compute gradients of **`loss`** for the variables in **`var_list`**.

This is the first part of **`minimize()`**. It returns a list of (gradient, variable) pairs where "gradient" is the gradient for "variable". Note that "gradient" can be a **`Tensor`**, an **`IndexedSlices`**, or **`None`** if there is no gradient for the given variable.

Args:

- `loss`: A Tensor containing the value to minimize.
- `var_list`: Optional list or tuple of **`tf.Variable`** to update to minimize **`loss`**. Defaults to the list of variables collected in the graph under the key **`GraphKey.TRAINABLE_VARIABLES`**.
- `gate_gradients`: How to gate the computation of gradients. Can be **`GATE_NONE`**, **`GATE_OP`**, or **`GATE_GRAPH`**.
- `aggregation_method`: Specifies the method used to combine gradient terms. Valid values are defined in the class **`AggregationMethod`**.
- `colocate_gradients_with_ops`: If True, try colocating gradients with the corresponding op.

- `grad_loss`: Optional. A `Tensor` holding the gradient computed for `loss`.

Returns:

A list of (gradient, variable) pairs. Variable is always present, but gradient can be `None`.

Raises:

- `TypeError`: If `var_list` contains anything else than `Variable` objects.
- `ValueError`: If some arguments are invalid.

get_name

```
get_name()
```

get_slot

```
get_slot(
    var,
    name
)
```

Return a slot named `name` created for `var` by the Optimizer.

Some `Optimizer` subclasses use additional variables. For example `Momentum` and `Adagrad` use variables to accumulate updates. This method gives access to these `Variable` objects if for some reason you need them.

Use `get_slot_names()` to get the list of slot names created by the `Optimizer`.

Args:

- `var`: A variable passed to `minimize()` or `apply_gradients()`.
- `name`: A string.

Returns:

The `Variable` for the slot if it was created, `None` otherwise.

get_slot_names

```
get_slot_names()
```

Return a list of the names of slots created by the `Optimizer`.

See `get_slot()`.

Returns:

A list of strings.

minimize

```
minimize(
    loss,
    global_step=None,
    var_list=None,
    gate_gradients=GATE_OP,
    aggregation_method=None,
    colocate_gradients_with_ops=False,
    name=None,
    grad_loss=None
)
```

Add operations to minimize `loss` by updating `var_list`.

This method simply combines calls `compute_gradients()` and `apply_gradients()`. If you want to process the gradient before applying them call `compute_gradients()` and `apply_gradients()` explicitly instead of using this function.

Args:

- `loss`: A `Tensor` containing the value to minimize.
- `global_step`: Optional `Variable` to increment by one after the variables have been updated.
- `var_list`: Optional list or tuple of `Variable` objects to update to minimize `loss`. Defaults to the list of variables collected in the graph under the key `GraphKeys.TRAINABLE_VARIABLES`.
- `gate_gradients`: How to gate the computation of gradients. Can be `GATE_NONE`, `GATE_OP`, or `GATE_GRAPH`.
- `aggregation_method`: Specifies the method used to combine gradient terms. Valid values are defined in the class `AggregationMethod`.
- `colocate_gradients_with_ops`: If True, try colocating gradients with the corresponding op.
- `name`: Optional name for the returned operation.
- `grad_loss`: Optional. A `Tensor` holding the gradient computed for `loss`.

Returns:

An Operation that updates the variables in `var_list`. If `global_step` was not `None`, that operation also increments `global_step`.

Raises:

- `ValueError`: If some of the variables are not `Variable` objects.

`swapping_saver`

```
swapping_saver(
    var_list=None,
    name='swapping_saver',
    **kwargs
)
```

Create a saver swapping moving averages and variables.

You should use this saver during training. It will save the moving averages of the trained parameters under the original parameter names. For evaluations or inference you should use a regular saver and it will automatically use the moving averages for the trained variable.

You must call this function after all variables have been created and after you have called Optimizer.minimize().

Args:

- `var_list`: List of variables to save, as per `Saver()`. If set to None, will save all the variables that have been created before this call.
- `name`: The name of the saver.
- `**kwargs`: Keyword arguments of `Saver()`.

Returns:

A `tf.train.Saver` object.

Raises:

- `RuntimeError`: If apply_gradients or minimize has not been called before.

Class Members

GATE_GRAPH

GATE_NONE

GATE_OP

Last updated November 2, 2017.