tf.train.Optimizer

Contents

Class `Optimizer`

Defined in `tensorflow/python/training/optimizer.py`.

See the guide: Training > Optimizers

Base class for optimizers.

This class defines the API to add Ops to train a model. You never use this class directly, but instead instantiate one of its subclasses such as `GradientDescentOptimizer`, `AdagradOptimizer`, or `MomentumOptimizer`.

Usage

```
# Create an optimizer with the desired parameters.
opt = GradientDescentOptimizer(learning_rate=0.1)
# Add Ops to the graph to minimize a cost by updating a list of variables.
# "cost" is a Tensor, and the list of variables contains tf.Variable
# objects.
opt_op = opt.minimize(cost, var_list=<list of variables>)
```

In the training program you will just have to run the returned Op.

```
# Execute opt_op to do one step of training:
opt_op.run()
```

Processing gradients before applying them.

Calling `minimize()` takes care of both computing the gradients and applying them to the variables. If you want to process the gradients before applying them you can instead use the optimizer in three steps:

1. Compute the gradients with `compute_gradients()`.
2. Process the gradients as you wish.
3. Apply the processed gradients with `apply_gradients()`.

Example:

```
# Create an optimizer.
opt = GradientDescentOptimizer(learning_rate=0.1)

# Compute the gradients for a list of variables.
grads_and_vars = opt.compute_gradients(loss, <list of variables>)

# grads_and_vars is a list of tuples (gradient, variable).  Do whatever you
# need to the 'gradient' part, for example cap them, etc.
capped_grads_and_vars = [(MyCapper(gv[0]), gv[1]) for gv in grads_and_vars]

# Ask the optimizer to apply the capped gradients.
opt.apply_gradients(capped_grads_and_vars)
```

Gating Gradients

Both **`minimize()`** and **`compute_gradients()`** accept a **`gate_gradients`** argument that controls the degree of parallelism during the application of the gradients.

The possible values are: **`GATE_NONE`**, **`GATE_OP`**, and **`GATE_GRAPH`**.

`GATE_NONE`: Compute and apply gradients in parallel. This provides the maximum parallelism in execution, at the cost of some non-reproducibility in the results. For example the two gradients of **`matmul`** depend on the input values: With **`GATE_NONE`** one of the gradients could be applied to one of the inputs *before* the other gradient is computed resulting in non-reproducible results.

`GATE_OP`: For each Op, make sure all gradients are computed before they are used. This prevents race conditions for Ops that generate gradients for multiple inputs where the gradients depend on the inputs.

`GATE_GRAPH`: Make sure all gradients for all variables are computed before any one of them is used. This provides the least parallelism but can be useful if you want to process all gradients before applying any of them.

Slots

Some optimizer subclasses, such as **`MomentumOptimizer`** and **`AdagradOptimizer`** allocate and manage additional variables associated with the variables to train. These are called *Slots*. Slots have names and you can ask the optimizer for the names of the slots that it uses. Once you have a slot name you can ask the optimizer for the variable it created to hold the slot value.

This can be useful if you want to log debug a training algorithm, report stats about the slots, etc.

Methods

`__init__`

```
__init__(
    use_locking,
    name
)
```

Create a new Optimizer.

This must be called by the constructors of subclasses.

Args:

- `use_locking`: Bool. If True apply use locks to prevent concurrent updates to variables.

- `name`: A non-empty string. The name to use for accumulators created for the optimizer.

Raises:

- `ValueError`: If name is malformed.

apply_gradients

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

Apply gradients to variables.

This is the second part of **`minimize()`**. It returns an **`Operation`** that applies gradients.

Args:

- `grads_and_vars`: List of (gradient, variable) pairs as returned by **`compute_gradients()`**.
- `global_step`: Optional **`Variable`** to increment by one after the variables have been updated.
- `name`: Optional name for the returned operation. Default to the name passed to the **`Optimizer`** constructor.

Returns:

An **`Operation`** that applies the specified gradients. If **`global_step`** was not None, that operation also increments **`global_step`**.

Raises:

- `TypeError`: If **`grads_and_vars`** is malformed.
- `ValueError`: If none of the variables have gradients.

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.

Class Members

`GATE_GRAPH`

`GATE_NONE`

GATE_OP

Last updated November 2, 2017.