TensorFlow

TensorFlow API r1.4

tf.contrib.opt.ScipyOptimizerInterface

Contents

Class **ScipyOptimizerInterface**

Inherits From: `ExternalOptimizerInterface`

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

Wrapper allowing `scipy.optimize.minimize` to operate a `tf.Session`.

Example:

```
vector = tf.Variable([7., 7.], 'vector')

# Make vector norm as small as possible.
loss = tf.reduce_sum(tf.square(vector))

optimizer = ScipyOptimizerInterface(loss, options={'maxiter': 100})

with tf.Session() as session:
  optimizer.minimize(session)

# The value of vector should now be [0., 0.].
```

Example with simple bound constraints:

```
vector = tf.Variable([7., 7.], 'vector')

# Make vector norm as small as possible.
loss = tf.reduce_sum(tf.square(vector))

optimizer = ScipyOptimizerInterface(
    loss, var_to_bounds={vector: ([1, 2], np.infty)})

with tf.Session() as session:
  optimizer.minimize(session)

# The value of vector should now be [1., 2.].
```

Example with more complicated constraints:

```
vector = tf.Variable([7., 7.], 'vector')

# Make vector norm as small as possible.
loss = tf.reduce_sum(tf.square(vector))
# Ensure the vector's y component is = 1.
equalities = [vector[1] - 1.]
# Ensure the vector's x component is >= 1.
inequalities = [vector[0] - 1.]

# Our default SciPy optimization algorithm, L-BFGS-B, does not support
# general constraints. Thus we use SLSQP instead.
optimizer = ScipyOptimizerInterface(
    loss, equalities=equalities, inequalities=inequalities, method='SLSQP')

with tf.Session() as session:
  optimizer.minimize(session)

# The value of vector should now be [1., 1.].
```

Methods

`__init__`

```
__init__(
    loss,
    var_list=None,
    equalities=None,
    inequalities=None,
    var_to_bounds=None,
    **optimizer_kwargs
)
```

Initialize a new interface instance.

Args:

- `loss`: A scalar `Tensor` to be minimized.
- `var_list`: Optional `list` of `Variable` objects to update to minimize `loss`. Defaults to the list of variables collected in the graph under the key `GraphKeys.TRAINABLE_VARIABLES`.
- `equalities`: Optional `list` of equality constraint scalar `Tensor`s to be held equal to zero.
- `inequalities`: Optional `list` of inequality constraint scalar `Tensor`s to be held nonnegative.
- `var_to_bounds`: Optional `dict` where each key is an optimization `Variable` and each corresponding value is a length-2 tuple of `(low, high)` bounds. Although enforcing this kind of simple constraint could be accomplished with the `inequalities` arg, not all optimization algorithms support general inequality constraints, e.g. L-BFGS-B. Both `low` and `high` can either be numbers or anything convertible to a NumPy array that can be broadcast to the shape of `var` (using `np.broadcast_to`). To indicate that there is no bound, use `None` (or `+/- np.infty`). For example, if `var` is a 2x3 matrix, then any of the following corresponding `bounds` could be supplied:
  - `(0, np.infty)`: Each element of `var` held positive.
  - `(-np.infty, [1, 2])`: First column less than 1, second column less than 2.
  - `(-np.infty, [[1], [2], [3]])`: First row less than 1, second row less than 2, etc.

    - `(-np.infty, [[1, 2, 3], [4, 5, 6]])` : Entry `var[0, 0]` less than 1, `var[0, 1]` less than 2, etc.
- `**optimizer_kwargs` : Other subclass-specific keyword arguments.

`minimize`

```
minimize(
    session=None,
    feed_dict=None,
    fetches=None,
    step_callback=None,
    loss_callback=None,
    **run_kwargs
)
```

Minimize a scalar `Tensor` .

Variables subject to optimization are updated in-place at the end of optimization.

Note that this method does *not* just return a minimization `Op` , unlike `Optimizer.minimize()` ; instead it actually performs minimization by executing commands to control a `Session` .

Args:

- `session` : A `Session` instance.
- `feed_dict` : A feed dict to be passed to calls to `session.run` .
- `fetches` : A list of `Tensor` s to fetch and supply to `loss_callback` as positional arguments.
- `step_callback` : A function to be called at each optimization step; arguments are the current values of all optimization variables flattened into a single vector.
- `loss_callback` : A function to be called every time the loss and gradients are computed, with evaluated fetches supplied as positional arguments.
- `**run_kwargs` : kwargs to pass to `session.run` .

Last updated November 2, 2017.