tfdeploy

Release 0.4.0

Mar 30, 2017

Contents

This page contains only API docs. For more info, visit tfdeploy on GitHub.

Deploy tensorflow graphs for fast evaluation and export to tensorflow-less environments running numpy.

CHAPTER 1

Classes

Model

class `tfdeploy.`**`Model`**(*path=None*)

A trained model that contains one or more converted tensorflow graphs. When *path* is set, a previously saved model is loaded from that path. Usage:

```
import tensorflow as tf
import tfdeploy as td

# build your graph, use names for input and output tensors
sess = tf.Session()
x = tf.placeholder("float", shape=[None, 784], name="input")
W = tf.Variable(tf.truncated_normal([784, 100], stddev=0.05))
b = tf.Variable(tf.zeros([100]))
y = tf.nn.softmax(tf.matmul(x, W) + b, name="output")
sess.run(tf.initialize_all_variables())

# ... training ...

# create a model and save it to disk
model = td.Model()
model.add(y, sess)
model.save("/path/to/model.pkl")
```

And then in an other file:

```
import tfdeploy as td
import numpy as np

model = td.Model("/path/to/model.pkl")
inp, outp = model.get("input", "output")

batch = np.random.rand(10000, 784)
result = outp.eval({inp: batch})
```

roots
Contained root tensors in a dict mapped to a key.

get (**names*, *key=None*)
Returns one or more `Tensor` instances given by *names* using a deep lookup within the model. If *key* is not *None*, only the root tensor with that *key* is traversed. *None* is returned when no tensor was found. In case a tensor is passed, it's name is used for the lookup.

add (*tensor*, *tf_sess=None*, *key=None*, ***kwargs*)
Adds a new root *tensor* for a *key* which, if *None*, defaults to a consecutive number. When *tensor* is not an instance of `Tensor` but an instance of `tensorflow.Tensor`, it is converted first. In that case, *tf_sess* should be a valid tensorflow session and *kwargs* are forwarded to the `Tensor` constructor.

load (*path*)
Loads all tensors from a file defined by *path* and adds them to the root set.

save (*path*)
Saves all tensors of the root set to a file defined by *path*.

Tensor

class `tfdeploy.`**`Tensor`** (*tf_tensor*, *tf_sess*, *tf_feed_dict=None*)
Building block of a model. In *graph* terms, tensors represent connections between nodes (ops) of a graph. It contains information on the op it results from. The conversion uses the (tensorflow) instances *tf_tensor* and *tf_sess*, *tf_feed_dict* can be set to evaluate the tensor's current value.

name
The name of the tensor.

value_index
The integer value index of this tensor, i.e., the position in the op's output list.

op
The op instance that defines the value of this tensor. When created from a `tensorflow.Placeholder` or a `tensorflow.Variable/V2`, op will be *None*.

value
The value of this tensor. When created from a `tensorflow.Variable/V2`, this will be the value of that variable, or *None* otherwise until it is evaluated the first time.

get (**names*)
Returns one or more tensors given by *names* using a deep lookup within the inputs of the op. Note that *this* tensor is returned when the name matches. *None* is returned when no tensor was found.

eval (*feed_dict=None*)
Returns the value of this tensor based on the evaluation of all dependent ops and tensors. You can overwrite values of dependent tensors using *feed_dict*, a mapping of tensors to numpy arrays, which is passed down the evaluation chain.

Operation

class `tfdeploy.`**`Operation`** (*tf_op*, **args*, ***kwargs*)
Building block of a model. In *graph* terms, operations (ops) represent nodes that are connected via tensors. It contains information on its input tensors. The conversion uses the (tensorflow) instance *tf_op*, all *args* and

kwargs are forwarded to the `Tensor` constructor for this op's input tensors. Op instances can have multiple implementations, i.e., different methods that lead to equivalent results but might use additional third-party software such as *scipy*. To select a specific implementation, invoke `use_impl()`:

```
# tell SomeOp to use the scipy implementation of its op logic
SomeOp.use_impl(IMPL_SCIPY)
```

See `add_impl()` for more info about adding new implementations.

`types`
`classmember`
A tuple containing the types of tensorflow ops that this op can represent.

`unpack`
`classmember`
If *True* (default), the values of evaluated input tensors are forwarded to *func* as single arguments, or, otherwise, as a list.

`attrs`
`classmember`
Names of the configuration attributes of the original tensorflow op in a tuple.

`name`
The name of the op.

`inputs`
Tuple of tensors that are input to this op. Their order is important as they are forwarded to *func* for evaluation.

`kwargs`
Keyword arguments containing configuration values that will be passed to *func*.

classmethod `new` (*tf_op*, **args*, ***kwargs*)
Factory function that takes a tensorflow op *tf_op* and returns an instance of the appropriate op class. *args* and *kwargs* are forwarded to the op constructor. Raises an exception of type `UnknownOperationException` in case the requested op type is not known.

`set_attr` (*attr*, *value*)
Overwrites the value of an attribute *attr* with a new *value*.

`get` (**names*)
Returns one or more tensors given by *names* using a deep lookup within this op. *None* is returned when no tensor was found.

`eval` (*feed_dict=None*)
Returns the value of all output tensors in a tuple. See `Tensor.eval()` for more info.

classmethod `func` (**args*)
The actual op logic. By default, the method call is forwareded to the implementation-specific version which is determined using *impl*. Overwrite this method in inheriting classes to disable this feature. Must return a tuple.

static `func_numpy` (**args*)
Numpy implementation of the op logic. Returns a tuple.

static `func_scipy` (**args*)
Scipy implementation of the op logic. Returns a tuple.

classmethod `factory` (*func=None*, *impl=IMPL_NUMPY*, ***kwargs*)
Returns a new op class whose static function will be set to *func*. The name of *func* will also be the op class

name. *impl* is the default implementation type of the op. *kwargs* are used to update the class dict of the newly created op class.

classmethod **`use_impl`**(*impl*)
Switches the implementation type to *impl*. Returns the previous type.

classmethod **`add_impl`**(*impl*)
Decorator to add an additional implementation to this op. Example:

```
# initial implementation using factory, defaults to numpy
@Operation.factory
def MyOp(a, b):
    # use numpy only
    return ...

# also add a scipy implementation
@MyOp.add_impl(IMPL_SCIPY)
def MyOp(a, b):
    # also use scipy
    return ...
```

Ensemble

class `tfdeploy.`**`Ensemble`**(*paths=None*, *method=0*)
An ensemble is a wrapper around multiple models to compute ensemble values. It can initialized with a list of model paths and an ensembling method that decides how to compute the merged value.

```
# create the ensemble
ensemble = Ensemble(["model1.pkl", "model2.pkl", ...], METHOD_MEAN)

# get input and output tensors (which actually are TensorEnsemble instances)
input, output = ensemble.get("input", "output")

# evaluate the ensemble just like a normal model
batch = ...
value = output.eval({input: batch})
```

If you want to use another method than `METHOD_MEAN`, `METHOD_MAX` or `METHOD_MAX`, use `METHOD_CUSTOM` and overwrite the `func_custom` method of the *`TensorEnsemble`* instance.

`models`
A list that contains all read models.

`method`
The ensembling method.

`get`(**names*, *key=None*)
Returns one or more *`TensorEnsemble`* instances given by *names* using a deep lookup within all read models. Each returned tensor ensemble will have `len(models)` tensors. If a model does not contain a specific tensor defined by a specific *name*, the associated ensemble tensor will contain a *None* for that model in its tensors. If *key* is not *None*, only the root tensors with that *key* are traversed.

`load`(*paths*)
Loads models from a list of *paths*.

TensorEnsemble

class `tfdeploy.`**`TensorEnsemble`**(*tensors*, *method=0*)

A tensor ensemble basically contains a list of tensors that correspond to models of an *`Ensemble`* instance.

`eval`(*feed_dict=None*)

Evaluates all contained tensors using a *feed_dict* and returns the ensemble value. The keys of *feed_dict* must be tensor ensembles. Its values can be batches, i.e., numpy arrays, or lists or tuples of batches. In the latter case, these lists or tuples must have the same length as the list of stored tensors as they will be mapped.

`func`(*values*)

The actual ensembling logic that combines multiple *values*. The method call is forwareded tothe ensemble method-specific variant which is determined using *method*.

CHAPTER 2

Functions

setup

tfdeploy.**setup**(*tf*, *order=None*)

Sets up global variables (currently only the tensorflow version) to adapt to peculiarities of different tensorflow versions. This function should only be called before `Model` creation, not for evaluation. Therefore, the tensorflow module *tf* must be passed:

```
import tensorflow as tf
import tfdeploy as td

td.setup(tf)

# ...
```

Also, when *order* is not *None*, it is forwarded to `optimize()` for convenience.

reset

tfdeploy.**reset**()

Resets the instance caches of `TensorRegister` and `OperationRegister`.

optimize

tfdeploy.**optimize**(*impl*)

Tries to set the implementation type of all registered `Operation` classes to *impl*. This has no effect when an op does not implement that type.

The behavior is equivalent to:

```
for op in Operation.__subclasses__():
    if impl in op.impls:
        op.use_impl(impl)
```

impl can also be a list or tuple of valid implementation types representing a preferred order.

print_tensor

`tfdeploy.`**`print_tensor`**`(`*td_tensor*, *indent=" "*, *max_depth=-1*`)`

Prints the dependency graph of a `Tensor` *td_tensor*, where each new level is indented by *indent*. When *max_depth* is positive, the graph is truncated at that depth, where each tensor and each op count as a level.

print_op

`tfdeploy.`**`print_op`**`(`*td_op*, *indent=" "*, *max_depth=-1*`)`

Prints the dependency graph of a `Operation` *td_op*, where each new level is indented by *indent*. When *max_depth* is positive, the graph is truncated at that depth, where each tensor and each op count as a level.

print_tf_tensor

`tfdeploy.`**`print_tf_tensor`**`(`*tf_tensor*, *indent=" "*, *max_depth=-1*`)`

Prints the dependency graph of a tensorflow tensor *tf_tensor*, where each new level is indented by *indent*. When *max_depth* is positive, the graph is truncated at that depth, where each tensor and each op count as a level.

print_tf_op

`tfdeploy.`**`print_tf_op`**`(`*tf_tensor*, *indent=" "*, *max_depth=-1*`)`

Prints the dependency graph of a tensorflow operation *tf_op*, where each new level is indented by *indent*. When *max_depth* is positive, the graph is truncated at that depth, where each tensor and each op count as a level.

CHAPTER 3

Other Attributes

`tfdeploy.`**`IMPL_NUMPY`**
Implementation type for ops that use numpy (the default).

`tfdeploy.`**`IMPL_SCIPY`**
Implementation type for ops that use scipy.

`tfdeploy.`**`HAS_SCIPY`**
A flag that is *True* when scipy is available on your system.

CHAPTER 4

Exceptions

UnknownOperationException

exception `tfdeploy.`**`UnknownOperationException`**
An exception which is raised when trying to convert an unknown tensorflow.

OperationMismatchException

exception `tfdeploy.`**`OperationMismatchException`**
An exception which is raised during instantiation of an op whose type does not match the underlying tensorflow op.

InvalidImplementationException

exception `tfdeploy.`**`InvalidImplementationException`**
An exception which is raised when an implementation of an unknown type is registered for an *`Operation`* class.

UnknownImplementationException

exception `tfdeploy.`**`UnknownImplementationException`**
An exception which is raised when an *`Operation`* instance is requested to use an implementation type that was not yet added.

`UnknownEnsembleMethodException`

exception `tfdeploy.`**`UnknownEnsembleMethodException`**

An exception which is raised when an *`Ensemble`* instance is initialised with an unknown ensemle method.

`EnsembleMismatchException`

exception `tfdeploy.`**`EnsembleMismatchException`**

An exception which is raised when a *`TensorEnsemble`* instance is evaluated with a *feed_dict* whose keys, i.e. also *`TensorEnsemble`* instances, do not match the tensor to evaluate. An example would be that a tensor ensemble with *n* tensors is evaluated with a tensor ensemble it its *feed_dict* that contains *m* tensors.

`ScipyOperationException`

exception `tfdeploy.`**`ScipyOperationException`**(*attr*)

An exception which is raised when trying to evaluate an op that uses scipy internally and scipy is not available.

Python Module Index

Index