tf.contrib.training.HParams

Contents

Class `HParams`

Defined in `tensorflow/contrib/training/python/training/hparam.py`.

Class to hold a set of hyperparameters as name-value pairs.

A `HParams` object holds hyperparameters used to build and train a model, such as the number of hidden units in a neural net layer or the learning rate to use when training.

You first create a `HParams` object by specifying the names and values of the hyperparameters.

To make them easily accessible the parameter names are added as direct attributes of the class. A typical usage is as follows:

```
# Create a HParams object specifying names and values of the model
# hyperparameters:
hparams = HParams(learning_rate=0.1, num_hidden_units=100)

# The hyperparameter are available as attributes of the HParams object:
hparams.learning_rate ==> 0.1
hparams.num_hidden_units ==> 100
```

Hyperparameters have type, which is inferred from the type of their value passed at construction type. The currently supported types are: integer, float, string, and list of integer, float, or string.

You can override hyperparameter values by calling the `parse()` method, passing a string of comma separated `name=value` pairs. This is intended to make it possible to override any hyperparameter values from a single command-line flag to which the user passes 'hyper-param=value' pairs. It avoids having to define one flag for each hyperparameter.

The syntax expected for each value depends on the type of the parameter. See `parse()` for a description of the syntax.

Example:

```
# Define a command line flag to pass name=value pairs.
# For example using argparse:
import argparse
parser = argparse.ArgumentParser(description='Train my model.')
parser.add_argument('--hparams', type=str,
                    help='Comma separated list of "name=value" pairs.')
args = parser.parse_args()
...
def my_program():
  # Create a HParams object specifying the names and values of the
  # model hyperparameters:
  hparams = tf.HParams(learning_rate=0.1, num_hidden_units=100,
                       activations=['relu', 'tanh'])

  # Override hyperparameters values by parsing the command line
  hparams.parse(args.hparams)

  # If the user passed `--hparams=learning_rate=0.3` on the command line
  # then 'hparams' has the following attributes:
  hparams.learning_rate ==> 0.3
  hparams.num_hidden_units ==> 100
  hparams.activations ==> ['relu', 'tanh']

  # If the hyperparameters are in json format use parse_json:
  hparams.parse_json('{"learning_rate": 0.3, "activations": "relu"}')
```

Methods

`__init__`

```
__init__(
    hparam_def=None,
    model_structure=None,
    **kwargs
)
```

Create an instance of `HParams` from keyword arguments.

The keyword arguments specify name-values pairs for the hyperparameters. The parameter types are inferred from the type of the values passed.

The parameter names are added as attributes of `HParams` object, so they can be accessed directly with the dot notation `hparams._name_`.

Example:

```
# Define 3 hyperparameters: 'learning_rate' is a float parameter,
# 'num_hidden_units' an integer parameter, and 'activation' a string
# parameter.
hparams = tf.HParams(
    learning_rate=0.1, num_hidden_units=100, activation='relu')

hparams.activation ==> 'relu'
```

Note that a few names are reserved and cannot be used as hyperparameter names. If you use one of the reserved name the constructor raises a `ValueError`.

Args:

- `hparam_def`: Serialized hyperparameters, encoded as a hparam_pb2.HParamDef protocol buffer. If provided, this object is initialized by deserializing hparam_def. Otherwise **kwargs is used.
- `model_structure`: An instance of ModelStructure, defining the feature crosses to be used in the Trial.
- `**kwargs`: Key-value pairs where the key is the hyperparameter name and the value is the value for the parameter.

Raises:

- `ValueError`: If both `hparam_def` and initialization values are provided, or if one of the arguments is invalid.

add_hparam

```
add_hparam(
    name,
    value
)
```

Adds {name, value} pair to hyperparameters.

Args:

- `name`: Name of the hyperparameter.
- `value`: Value of the hyperparameter. Can be one of the following types: int, float, string, int list, float list, or string list.

Raises:

- `ValueError`: if one of the arguments is invalid.

from_proto

```
@staticmethod
from_proto(
    hparam_def,
    import_scope=None
)
```

get_model_structure

```
get_model_structure()
```

parse

```
parse(values)
```

Override hyperparameter values, parsing new values from a string.

See parse_values for more detail on the allowed format for values.

Args:

- `values`: String. Comma separated list of `name=value` pairs where 'value' must follow the syntax described above.

Returns:

The `HParams` instance.

Raises:

- `ValueError`: If `values` cannot be parsed.

parse_json

```
parse_json(values_json)
```

Override hyperparameter values, parsing new values from a json object.

Args:

- `values_json`: String containing a json object of name:value pairs.

Returns:

The `HParams` instance.

Raises:

- `ValueError`: If `values_json` cannot be parsed.

set_from_map

```
set_from_map(values_map)
```

Override hyperparameter values, parsing new values from a dictionary.

Args:

- `values_map`: Dictionary of name:value pairs.

Returns:

The `HParams` instance.

Raises:

- `ValueError`: If `values_map` cannot be parsed.

set_hparam

```
set_hparam(
    name,
    value
)
```

Set the value of an existing hyperparameter.

This function verifies that the type of the value matches the type of the existing hyperparameter.

Args:

- `name`: Name of the hyperparameter.
- `value`: New value of the hyperparameter.

Raises:

- `ValueError`: If there is a type mismatch.

`set_model_structure`

```
set_model_structure(model_structure)
```

`to_json`

```
to_json()
```

Serializes the hyperparameters into JSON.

Returns:

A JSON string.

`to_proto`

```
to_proto(export_scope=None)
```

Converts a `HParams` object to a `HParamDef` protocol buffer.

Args:

- `export_scope`: Optional `string`. Name scope to remove.

Returns:

A `HParamDef` protocol buffer.

`values`

```
values()
```

Return the hyperparameter values as a Python dictionary.

Returns:

A dictionary with hyperparameter names as keys. The values are the hyperparameter values.

Last updated November 2, 2017.