maggy Documentation

Release 0.1

Logical Clocks AB

May 21, 2019

Contents:

Maggy is a framework for efficient asynchronous optimization of expensive black-box functions on top of Apache Spark. Compared to existing frameworks, maggy is not bound to stage based optimization algorithms and therefore it is able to make extensive use of early stopping in order to achieve efficient resource utilization.

Right now, maggy supports asynchronous hyperparameter tuning of machine learning and deep learning models, but other use cases include ablation studies and asynchronous distributed training.

Moreover, it provides a developer API that allows advanced usage by implementing custom optimization algorithms and early stopping criteria.

In order to make decisions on early stopping, the Spark executors are sending heart beats with the current performance of the model they are training to the maggy experiment driver which is running on the Spark driver. We call the process of training a model with a certain hyperparameter combination a *trial*. The experiment driver then uses all information of finished trials and the currently running ones to check in a specified interval, which of the trials should be stopped early. Subsequently, the experiment driver provides a new trial to the Spark executor.

CHAPTER 1

Quick Start

To Install:

```
>>> pip install maggy
```

The programming model is that you wrap the code containing the model training inside a wrapper function. Inside that wrapper function provide all imports and parts that make up your experiment.

There are three requirements for this wrapper function:

1. The function should take the hyperparameters as arguments, plus one additional parameter reporter which is needed for reporting the current metric to the experiment driver.
2. The function should return the metric that you want to optimize for. This should coincide with the metric being reported in the Keras callback (see next point).
3. In order to leverage on the early stopping capabilities of maggy, you need to make use of the maggy reporter API. By including the reporter in your training loop, you are telling maggy which metric to report back to the experiment driver for optimization and to check for global stopping. It is as easy as adding reporter.broadcast(metric=YOUR_METRIC) for example at the end of your epoch or batch training step and adding a reporter argument to your function signature. If you are not writing your own training loop you can use the pre-written Keras callbacks in the *maggy.callbacks* module.

Sample usage:

```
>>> # Define Searchspace
>>> from maggy import Searchspace
>>> # The searchspace can be instantiated with parameters
>>> sp = Searchspace(kernel=('INTEGER', [2, 8]), pool=('INTEGER', [2, 8]))
>>> # Or additional parameters can be added one by one
>>> sp.add('dropout', ('DOUBLE', [0.01, 0.99]))
```

```
>>> # Define training wrapper function:
>>> def mnist(kernel, pool, dropout, reporter):
>>>     # This is your training iteration loop
>>>     for i in range(number_iterations):
```

(continues on next page)

(continued from previous page)

```
>>>         ...
>>>         # add the maggy reporter to report the metric to be optimized
>>>         reporter.broadcast(metric=accuracy)
>>>         ...
>>>     # Return the same final metric
>>>     return accuracy
```

```
>>> # Launch maggy experiment
>>> from maggy import experiment
>>> result = experiment.lagom(map_fun=mnist,
>>>                           searchspace=sp,
>>>                           optimizer='randomsearch',
>>>                           direction='max',
>>>                           num_trials=15,
>>>                           name='MNIST'
>>>                          )
```

lagom is a Swedish word meaning "just the right amount". This is how maggy uses your resources.

CHAPTER 2

MNIST Example

For a full MNIST example with random search using Keras, see the Jupyter Notebook in the *examples* folder.

CHAPTER 3

Documentation

API documentation is available here.

3.1 Maggy User API

3.1.1 maggy.experiment module

Experiment module used for running asynchronous optimization tasks.

The programming model is that you wrap the code containing the model training inside a wrapper function. Inside that wrapper function provide all imports and parts that make up your experiment, see examples below. Whenever a function to run an experiment is invoked it is also registered in the Experiments service along with the provided information.

maggy.experiment.**lagom**(*map_fun*, *searchspace*, *optimizer*, *direction*, *num_trials*, *name*, *hb_interval=1*, *es_policy='median'*, *es_interval=300*, *es_min=10*, *description=''*)

Launches a maggy experiment for hyperparameter optimization.

Given a search space, objective and a model training procedure *map_fun* (black-box function), an experiment is the whole process of finding the best hyperparameter combination in the search space, optimizing the black-box function. Currently maggy supports random search and a median stopping rule.

lagom is a Swedish word meaning "just the right amount".

Parameters

- **map_fun** (*function*) – User defined experiment containing the model training.
- **searchspace** (Searchspace) – A maggy Searchspace object from which samples are drawn.
- **optimizer** (*str, AbstractOptimizer*) – The optimizer is the part generating new trials.

- **direction** (*str*) – If set to 'max' the highest value returned will correspond to the best solution, if set to 'min' the opposite is true.
- **num_trials** (*int*) – the number of trials to evaluate given the search space, each containing a different hyperparameter combination
- **name** (*str*) – A user defined experiment identifier.
- **hb_interval** (*int, optional*) – The heartbeat interval in secondss from trial executor to experiment driver, defaults to 1
- **es_policy** (*str, optional*) – The earlystopping policy, defaults to 'median'
- **es_interval** (*int, optional*) – Frequency interval in seconds to check currently running trials for early stopping, defaults to 300
- **es_min** (*int, optional*) – Minimum number of trials finalized before checking for early stopping, defaults to 10
- **description** (*str, optional*) – A longer description of the experiment.

Raises **RuntimeError** – An experiment is currently running.

Returns A dictionary indicating the best trial and best hyperparameter combination with it's performance metric

Return type dict

3.1.2 maggy.searchspace module

class maggy.**Searchspace**(***kwargs*)

Create an instance of *Searchspace* from keyword arguments.

The keyword arguments specify name-values pairs for the hyperparameters, where values are tuples of the form (type, list). Type is a string with one of the following values:

- DOUBLE
- INTEGER
- DISCRETE
- CATEGORICAL

And the list in the tuple specifies either two values only, the start and end point of of the feasible interval for DOUBLE and INTEGER, or the discrete possible values for the types DISCRETE and CATEGORICAL.

Sample usage:

```
>>> # Define Searchspace
>>> from maggy import Searchspace
>>> # The searchspace can be instantiated with parameters
>>> sp = Searchspace(kernel=('INTEGER', [2, 8]), pool=('INTEGER', [2, 8]))
>>> # Or additional parameters can be added one by one
>>> sp.add('dropout', ('DOUBLE', [0.01, 0.99]))
```

The *Searchspace* object can also be initialized from a python dictionary:

```
>>> sp_dict = sp.to_dict()
>>> sp_new = Searchspace(**sp_dict)
```

The parameter names are added as attributes of *Searchspace* object, so they can be accessed directly with the dot notation *searchspace._name_*.

add(*name*, *value*)
Adds {name, value} pair to hyperparameters.

Parameters

- **name** (*str*) – Name of the hyperparameter
- **value** (*tuple*) – A tuple of the parameter type and its feasible region

Raises

- **ValueError** – Hyperparameter name is reserved
- **ValueError** – Hyperparameter feasible region in wrong format

get(*name*, *default=None*)
Returns the value of *name* if it exists, else *default*.

get_random_parameter_values(*num*)
Generate random parameter dictionaries, e.g. to be used for initializing an optimizer.

Parameters **num** (*int*) – number of random parameter dictionaries to be generated.

Raises **ValueError** – *num* is not an int.

Returns a list containing parameter dictionaries

Return type list

names()
Returns the dictionary with the names and types of all hyperparameters.

Returns Dictionary of hyperparameter names, with types as value

Return type dict

to_dict()
Return the hyperparameters as a Python dictionary.

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

Return type dict

3.1.3 maggy.callbacks module

class maggy.callbacks.**KerasBatchEnd**(*reporter*, *metric='loss'*)
A Keras callback reporting a specified *metric* at the end of the batch to the maggy experiment driver.

loss is always available as a metric, and optionally *acc* (if accuracy monitoring is enabled, that is, accuracy is added to keras model metrics). Validation metrics are not available for the BatchEnd callback. Validation after every batch would be too expensive. Default is training loss (*loss*).

Example usage:

```
>>> from maggy.callbacks import KerasBatchEnd
>>> callbacks = [KerasBatchEnd(reporter, metric='acc')]
```

class maggy.callbacks.**KerasEpochEnd**(*reporter*, *metric='val_loss'*)
A Keras callback reporting a specified *metric* at the end of an epoch to the maggy experiment driver.

val_loss is always available as a metric, and optionally *val_acc* (if accuracy monitoring is enabled, that is, accuracy is added to keras model metrics). Training metrics are available under the names *loss* and *acc*. Default is validation loss (*val_loss*).

Example usage:

```
>>> from maggy.callbacks import KerasBatchEnd
>>> callbacks = [KerasBatchEnd(reporter, metric='val_acc')]
```

3.2 Maggy Developer API

3.3 Release 0.1

3.4 License

GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007. See LICENSE.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index