deephyper Documentation

Release alpha v0.0.5

ArgonneMCS

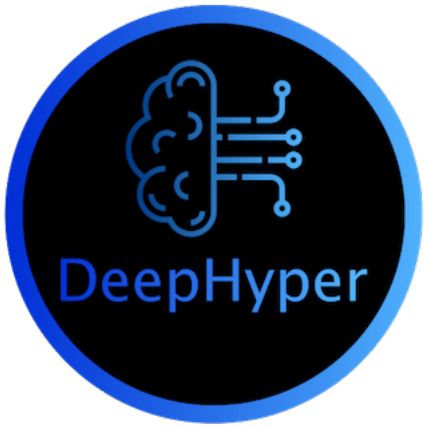
Feb 26, 2019

QUICK START

CHAPTER

ONE

SCALABLE ASYNCHRONOUS NEURAL ARCHITECTURE AND HYPERPARAMETER SEARCH FOR DEEP NEURAL NETWORKS



Deephyper is a Python package which provides a common interface for the implementation and study of scalable hyperparamet

- `benchmark`: a set of problems for hyperparameter or neural architecture search which the user can use to compare our different search algorithms or as examples to build their own problems.
- `evaluator`: a set of objects which help to run search on different systems and for different cases such as quick and light experiments or long and heavy runs.
- `search`: a set of algorithms for hyperparameter and neural architecture search. You will also find a modular way to define new search algorithms and specific sub modules for hyperparameter or neural architecture search.

1.1 Installation

1.1.1 Local

You can run the following commands if you want to install deephyper on your local machine.

```
# Cloning repos en installing with pip
git clone https://github.com/deephyper/deephyper.git
pip install -e deephyper/
```

1.1.2 Theta

Deephyper can be directly installed as a module on Theta.

```
module load deephyper
```

1.1.3 Cooley

Todo: installation on Cooley

1.1.4 Contribute to documentation

Installation

```
source activate ENV_NAME
pip install -U Sphinx
pip install sphinx_bootstrap_theme
```

Build

To build the documentation you just need to be in the `deephyper/doc` folder and run `make html` assuming you have MakeFile installed on your computer. Then you can see the build documentation inside the `doc/_build` folder just by opening the `index.html` file with your web browser.

Useful informations

The documentation is made with Sphinx and the following extensions are used :

Extensions	
Name	Description
autodoc	automatically insert docstrings from modules
napoleon	inline code documentation
doctest	automatically test code snippets in doctest blocks
intersphinx	link between Sphinx documentation of different projects
todo	write “todo” entries that can be shown or hidden on build
coverage	checks for documentation coverage
mathjax	include math, rendered in the browser by MathJax
ifconfig	conditional inclusion of content based on config values
viewcode	include links to the source code of documented Python objects
githubpages	create .nojekyll file to publish the document on GitHub pages

Sphinx uses reStructuredText files, click on this link if you want to have an overview of the corresponding syntax and mechanism.

<aside class=”notice”> Our documentation try to take part of the inline documentation in the code to auto-generate documentation from it. For that reason we highly recommend you to follow specific rules when writing inline documentation : https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html. </aside>

1.2 Running locally

This section will show you how to run Hyperparameter or neural architecture on your local machine. All search can be run throw command line or using python.

1.2.1 Hyperparameter search

Command Line

Assuming you have installed deephyper on your local environment we will show you how to run an asynchronous model-based search (AMBS) on a benchmark included within deephyper. To print the arguments of a search like AMBS just run:

```
python -m deephyper.search.hps.ambs --help
```

Now you can run AMBS with custom arguments:

```
python -m deephyper.search.hps.ambs --problem deephyper.benchmark.hps.polynome2.
↪Problem --run deephyper.benchmark.hps.polynome2.run
```

Python

Todo: use hps inside python

1.2.2 Neural Architecture Search

Command Line

```
python -m deephyper.search.nas.ppo_a3c_sync --problem deephyper.benchmark.nas.mnist1D.
↪problem.Problem --run deephyper.search.nas.model.run.alpha.run
```

Python

Todo: use nas inside python

1.3 Running on Theta

1.3.1 Hyperparameter Search

First we are going to run a search on b2 benchmark.

```
# Load deephyper on theta
module load deephyper

# Create a new postgress database in current directory
balsam init testdb

# Start or Link to the database
source balsamactivate testdb

# Create a new application and Print applications referenced
balsam app --name AMBS --exec $DH_AMBS
balsam ls apps

# Create a new job and Print jobs referenced
balsam job --name test --application AMBS --workflow TEST --args '--evaluator balsam -
↪-problem deephyper.benchmark.hps.polynome2.Problem --run deephyper.benchmark.hps.
↪polynome2.run'
balsam ls jobs

# Submit a Theta job that will run the balsam job corresponding to the search
balsam submit-launch -n 128 -q default -t 180 -A PROJECT_NAME --job-mode serial --wf-
↪filter TEST
```

Now if you want to look at the logs, go to `testdb/data/TEST`. You'll see one directory prefixed with `test`. Inside this directory you will find the logs of you search. All the other directories prefixed with `task` correspond to the logs of your `--run` function, here the run function is corresponding to the training of a neural network.

1.3.2 Neural Architecture Search

1.4 Introduction

Benchmarks are here for you to test the performance of different search algorithm and reproduce our results. They can also help you to test your installation of deephyper or discover the many parameters of a search. In deephyper we have two different kind of benchmark. The first type is *hyper parameters search* benchmark and the second type is *neural architecture search* benchmark. To see a full explanation about the different kind of search please refer to the following pages: Hyperparameter Searchs & Neural Architecture Search.

1.5 Problem

This class describe the most generic aspect of a problem. Basically we are using a python `dict` and adding key-values. It is mostly used for neural architecture search problems, see *Create a new NAS problem* for more details.

class deephyper.benchmark.problem.**Problem**

Representation of a problem.

Attribute: space (OrderedDict): represents the search space of the problem.

add_dim(*p_name*, *p_space*)

Add a dimension to the search space.

Parameters

- **p_name** (*str*) – name of the parameter/dimension.

- **p_space** (*Object*) – space corresponding to the new dimension.

1.6 Hyperparameter Search Problem

Use this class to define a hyperparameter search problem, see *Create a new HPS problem* for more details.

class deephyper.benchmark.problem.**HpProblem**
Problem specification for Hyperparameter Optimization

add_dim(*p_name*, *p_space*, *default=None*)
Add a dimension to the search space.

Parameters

- **p_name** (*str*) – name of the parameter/dimension.
- **p_space** (*Object*) – space corresponding to the new dimension.
- **default** – default value of the new dimension, it must be compatible with the p_space given.

starting_point
Starting point of the search space.

1.7 Hyper Parameters Search (HPS)

1.7.1 Create a new HPS problem

For HPS a benchmark is defined by a problem definition and a function that runs the model.

```
problem_folder/
      __init__.py
      problem.py
      model_run.py
```

The problem contains the parameters you want to search over. They are defined by their name, their space and a default value for the starting point. Deephyper recognizes three types of parameters : - continuous - discrete ordinal (for instance integers) - discrete non-ordinal (for instance a list of tokens) For example if we want to create an hyper parameter search problem for Mnist dataset :

```
from deephyper.benchmark import HpProblem

Problem = HpProblem()
Problem.add_dim('epochs', (5, 500), 5)

# benchmark specific parameters
Problem.add_dim('nhidden', (1, 100), 1)
Problem.add_dim('nunits', (1, 1000), 1)

# network parameters
Problem.add_dim('activation', ['relu', 'elu', 'selu', 'tanh'], 'relu')
Problem.add_dim('batch_size', (8, 1024), 8)
Problem.add_dim('dropout', (0.0, 1.0), 0.0)
Problem.add_dim('optimizer', ['sgd', 'rmsprop', 'adagrad', 'adadelta', 'adam', 'adamax
↪', 'nadam'], 'sgd')
```

(continues on next page)

(continued from previous page)

```
# common optimizer parameters
#Problem.add_dim(['clipnorm'] = (1e-04, 1e01)
#Problem.add_dim(['clipvalue'] = (1e-04, 1e01)
# optimizer parameters
Problem.add_dim('learning_rate', (1e-04, 1e01), 1e-04)
#Problem.add_dim(['momentum'] =  (0, 1e01)
#Problem.add_dim(['decay'] =  (0, 1e01)
#Problem.add_dim(['nesterov'] = [False, True]
#Problem.add_dim(['rho'] = (1e-04, 1e01)
#Problem.add_dim(['epsilon'] = (1e-08, 1e01)
#Problem.add_dim(['beta1'] = (1e-04, 1e01)
#Problem.add_dim(['beta2'] = (1e-04, 1e01)


if __name__ == '__main__':
    print(Problem)
```

and that's it, we just defined a problem with one dimension 'num_n_l1' where we are going to search the best number of neurons for the first dense layer.

Now we need to define how to run hour mnist model while taking in account this 'num_n_l1' parameter chosen by the search. Let's take an basic example from Keras documentation with a small modification to use the 'num_n_l1' parameter :

```
from __future__ import print_function
import sys
from pprint import pprint
import os

here = os.path.dirname(os.path.abspath(__file__))
top = os.path.dirname(os.path.dirname(os.path.dirname(here)))
sys.path.append(top)
BNAME = os.path.splitext(os.path.basename(__file__))[0]

from deephyper.benchmark import util

timer = util.Timer()
timer.start('module loading')

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
import os


from keras.callbacks import EarlyStopping
from deephyper.benchmark.util import TerminateOnTimeOut


from keras import layers
from deephyper.benchmark import keras_cmdline
from keras.models import load_model
import hashlib
import pickle
from deephyper.benchmark.mnistmlp.load_data import load_data
```

(continues on next page)

(continued from previous page)

```
from numpy.random import seed
from tensorflow import set_random_seed
timer.end()

seed(1)
set_random_seed(2)

def run(param_dict):
    param_dict = keras_cmdline.fill_missing_defaults(augment_parser, param_dict)
    optimizer = keras_cmdline.return_optimizer(param_dict)
    pprint(param_dict)

    timer.start('stage in')
    if param_dict['data_source']:
        data_source = param_dict['data_source']
    else:
        data_source = os.path.dirname(os.path.abspath(__file__))
        data_source = os.path.join(data_source, 'data')

    (x_train, y_train), (x_test, y_test) = load_data(
        origin=os.path.join(data_source, 'mnist.npz'),
        dest=param_dict['stage_in_destination']
    )

    timer.end()

    num_classes = 10

    BATCH_SIZE = param_dict['batch_size']
    EPOCHS = param_dict['epochs']
    DROPOUT = param_dict['dropout']
    ACTIVATION = param_dict['activation']
    NHIDDEN = param_dict['nhidden']
    NUNITS = param_dict['nunits']
    TIMEOUT = param_dict['timeout']

    timer.start('preprocessing')

    x_train = x_train.reshape(60000, 784)
    x_test = x_test.reshape(10000, 784)
    x_train = x_train.astype('float32')
    x_test = x_test.astype('float32')
    x_train /= 255
    x_test /= 255
    print(x_train.shape[0], 'train samples')
    print(x_test.shape[0], 'test samples')

    # convert class vectors to binary class matrices
    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)

    model_path = param_dict['model_path']
    model_mda_path = None
    model = None
    initial_epoch = 0
```

(continues on next page)

(continued from previous page)

```
    if model_path:
        savedModel = util.resume_from_disk(BNAME, param_dict, data_dir=model_path)
        model_mda_path = savedModel.model_mda_path
        model_path = savedModel.model_path
        model = savedModel.model
        initial_epoch = savedModel.initial_epoch

    if model is None:
        model = Sequential()
        model.add(Dense(NUNITS, activation=ACTIVATION, input_shape=(784,)))
        model.add(Dropout(DROPOUT))
        for i in range(NHIDDEN):
            model.add(Dense(NUNITS, activation=ACTIVATION))
            model.add(Dropout(DROPOUT))
        model.add(Dense(num_classes, activation='softmax'))
        model.summary()
        model.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])

    timer.end()

    #earlystop = EarlyStopping(monitor='val_acc', min_delta=0.0001, patience=50,␣
↪verbose=1, mode='auto')
    timeout_monitor = TerminateOnTimeOut(TIMEOUT)
    callbacks_list = [timeout_monitor]

    timer.start('model training')
    history = model.fit(x_train, y_train,
                    batch_size=BATCH_SIZE,
                    initial_epoch=initial_epoch,
                    epochs=EPOCHS,
                    verbose=1,
                    callbacks=callbacks_list,
                    validation_split = 0.3)
                    #validation_data=(x_test, y_test))
    timer.end()
    score = model.evaluate(x_test, y_test, verbose=0)
    print('Test loss:', score[0])
    print('Test accuracy:', score[1])

    if model_path:
        timer.start('model save')
        model.save(model_path)
        util.save_meta_data(param_dict, model_mda_path)
        timer.end()

    print('OUTPUT:', -score[1])
    return -score[1]


def augment_parser(parser):

    parser.add_argument('--nunits', action='store', dest='nunits',
                        nargs='?', const=2, type=int, default='512',
                        help='number of units/layer in MLP')
```

(continues on next page)

(continued from previous page)

```
    parser.add_argument('--nhidden', action='store', dest='nhidden',
                        nargs='?', const=2, type=int, default='2',
                        help='number of hidden layers in MLP')

    return parser

if __name__ == "__main__":
    parser = keras_cmdline.create_parser()
    parser = augment_parser(parser)
    cmdline_args = parser.parse_args()
    param_dict = vars(cmdline_args)
    run(param_dict)
```

Warning: When designing a new optimization experiment, keep in mind *model_run.py* must be runnable from an arbitrary working directory. This means that Python modules simply located in the same directory as the *model_run.py* will not be part of the default Python import path, and importing them will cause an *ImportError*!

To ensure that modules located alongside the *model_run.py* script are always importable, a quick workaround is to explicitly add the problem folder to *sys.path* at the top of the script

```
import os
import sys
here = os.path.dirname(os.path.abspath(__file__))
sys.path.insert(0, here)
# import user modules below here
```

1.7.2 Available benchmarks

Hyper Parameters Search Benchmarks `deephyper.benchmark.hps`		
Name	Type	Description
b1		
b2		
b3		
capsule		
cifar10cnn	Classification	https://www.cs.toronto.edu/~kriz/cifar.html
dummy1		
dummy2		
gcn		
mnistcnn	Classification	http://yann.lecun.com/exdb/mnist/
mnistmlp	Classification	http://yann.lecun.com/exdb/mnist/
rosen2		
rosen10		
rosen30		

1.8 Neural Architecture Search (NAS)

1.8.1 Create a new NAS problem

Neural Architecture Search models are following a generic workflow. This is why you don't need to define the function which ru

- load data
- preprocess data
- build tensor graph of model (using Structure interface)
- train model
- return accuracy or mse

The basic generic function which is used in our package to run a model for NAS is

`deephyper.search.nas.model.run.alpha.`**`run`**(*config*)

All the items are linked by a problem definition.

Problem

Let's take the problem of our most simple benchmark as an example `deephyper.benchmark.nas.linearReg.problem`.

```
from deephyper.benchmark import Problem
from deephyper.benchmark.nas.linearReg.load_data import load_data
from deephyper.search.nas.model.baseline.anl_mlp_2 import create_structure
from deephyper.search.nas.model.preprocessing import minmaxstdscaler

# We create our Problem object with the Problem class, you don't have to name your␣
↪Problem object 'Problem' it can be any name you want. You can also define different␣
↪problems in the same module.
Problem = Problem()

# You define if your problem is a regression problem (the reward will be minus of the␣
↪mean squared error) or a classification problem (the reward will be the accuracy of␣
↪the network on the validation set).
Problem.add_dim('regression', True)

# You define how to load your data by giving a 'load_data' function. This function␣
↪will return your data set following this interface: (train_X, train_y), (valid_X,␣
↪valid_y). You can also add a 'kwargs' key with arguments for the load_data function.
Problem.add_dim('load_data', {
    'func': load_data,
})

# OPTIONAL : You define a preprocessing function which will be applied on your data␣
↪before training generated models. This preprocessing function use sklearn␣
↪preprocessors api.
Problem.add_dim('preprocessing', {
    'func': minmaxstdscaler
})

# You define the create structure function. This function will return an object␣
↪following the Structure interface. You can also have kwargs arguments such as 'num
↪cells' for this function.
```

(continues on next page)

(continued from previous page)

```
Problem.add_dim('create_structure', {
    'func': create_structure,
    'kwargs': {
        'num_cells': 5
    }
})

# You define the hyperparameters used to train your generated models during the␣
↪search.
Problem.add_dim('hyperparameters', {
    'batch_size': 100,
    'learning_rate': 0.01,
    'optimizer': 'adam',
    'num_epochs': 10, #50,
    'loss_metric': 'mean_squared_error',
    'metrics': ['mean_squared_error']
})

# Just to print your problem, to test its definition and imports in the current␣
↪python environment.
if __name__ == '__main__':
    print(Problem)
```

Load Data

A `load_data` function returns the data of your problem following the interface: `(train_X, train_Y), (valid_X, valid_Y)`.

```
import os
from tensorflow.examples.tutorials.mnist import input_data
import numpy as np
from deephyper.benchmark.benchmark_functions_wrappers import linear_

HERE = os.path.dirname(os.path.abspath(__file__))

np.random.seed(2018)

def load_data(dim=10):
    """
    Generate data for linear function -sum(x_i).

    Return:
        Tuple of Numpy arrays: ``(train_X, train_y), (valid_X, valid_y)``.
    """
    size = 100000
    prop = 0.80
    f, (a, b), _ = linear_()
    d = b - a
    x = np.array([a + np.random.random(dim) * d for i in range(size)])
    y = np.array([[f(v)] for v in x])

    sep_index = int(prop * size)
    train_X = x[:sep_index]
    train_y = y[:sep_index]
```

(continues on next page)

(continued from previous page)

```
    valid_X = x[sep_index:]
    valid_y = y[sep_index:]

    print(f'train_X shape: {np.shape(train_X)}')
    print(f'train_y shape: {np.shape(train_y)}')
    print(f'valid_X shape: {np.shape(valid_X)}')
    print(f'valid_y shape: {np.shape(valid_y)}')
    return (train_X, train_y), (valid_X, valid_y)

if __name__ == '__main__':
    load_data()
```

Preprocessing

A preprocessing function is returning an object folling the same interface as scikit-learn preprocessors.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler


def stdscaler():
    """
    Return:
        preprocessor:
    """
    preprocessor = Pipeline([
        ('stdscaler', StandardScaler()),
    ])
    return preprocessor
```

Structure

Here is the structure used for the `deephyper.benchmark.nas.linearReg benchmark.

```
from deephyper.search.nas.cell.mlp import create_dense_cell_type2
from deephyper.search.nas.cell.structure import create_seq_struct_full_skipco


def create_structure(input_tensor, num_cells):
    return create_seq_struct_full_skipco(input_tensor, create_dense_cell_type2, num_
↪cells)
```

See *What is a Structure ?* for more details

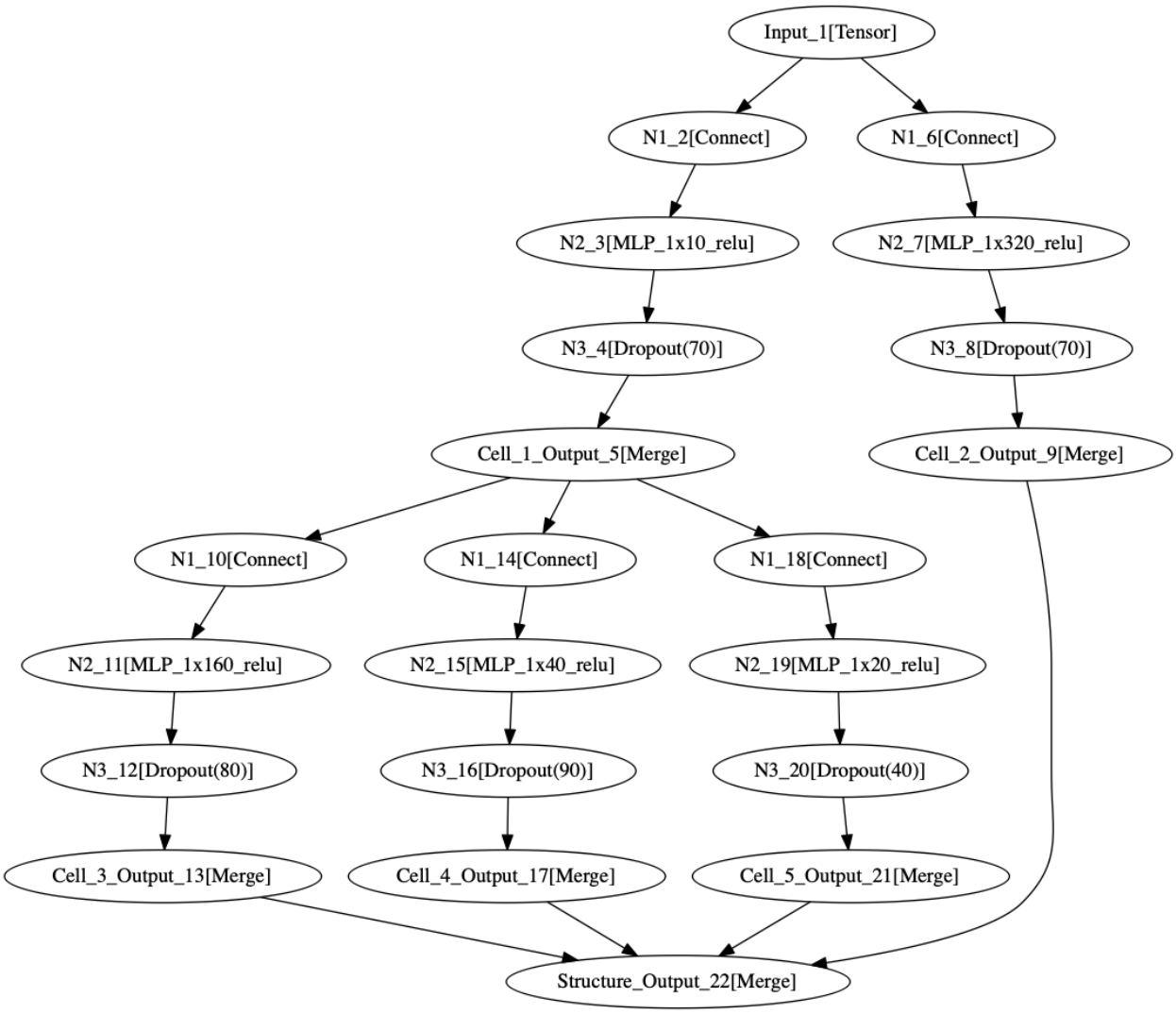


Fig. 1: A first example of graph generated with `anl_mlp_2.create_structure`.

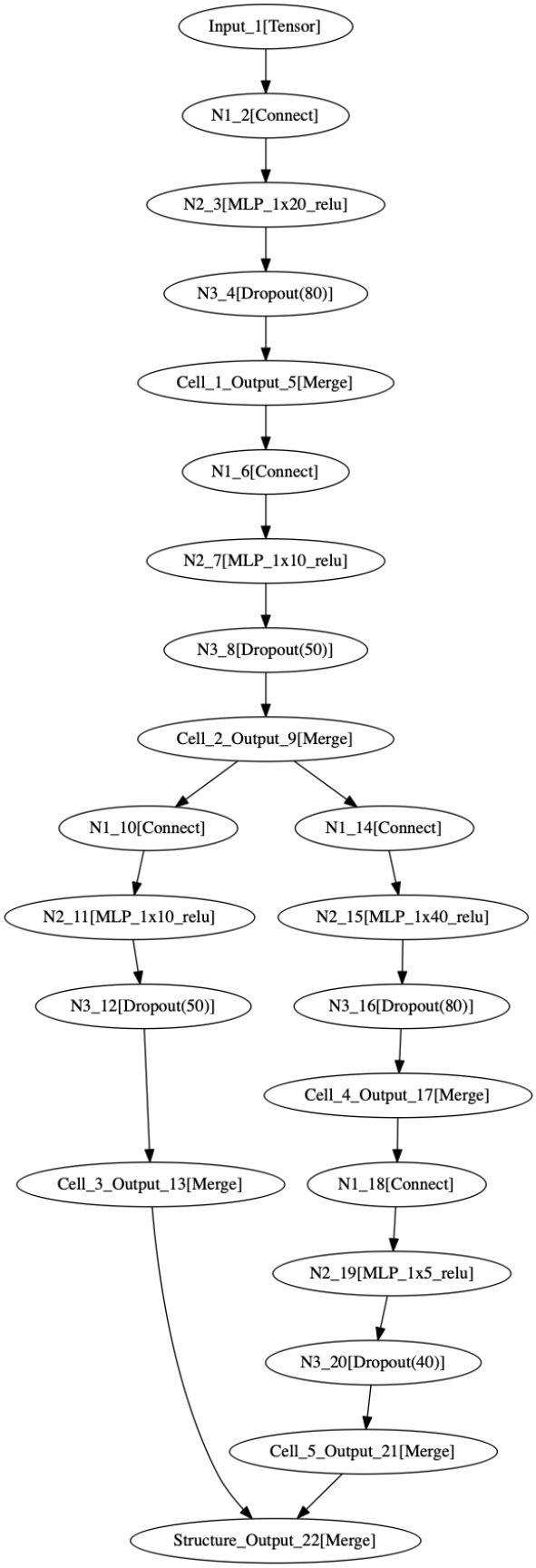


Fig. 2: A second example of graph generated with `anl_mlp_2.create_structure`.

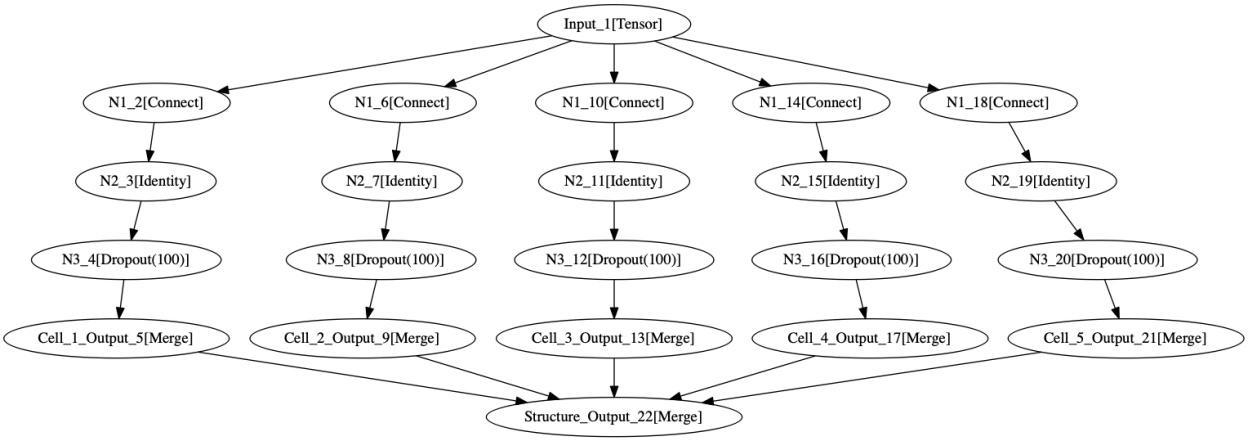


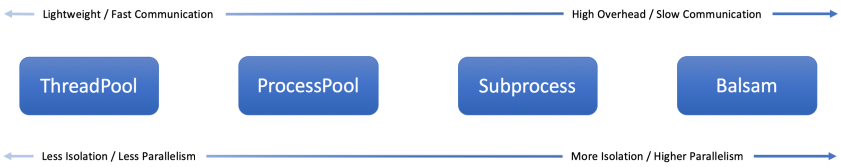
Fig. 3: A last example of graph generated with `anl_mlp_2.create_structure`.

1.8.2 Available benchmarks

Neural Architecture Search Benchmarks `deephyper.benchmark.nas`		
Name	Type	Description
ackleyReg	Regression	Generation of points in N dimensions corresponding to y=f(x) where f is https://www.sfu.ca/~ssurjano/ackley.html
cifar10	Classification	https://www.cs.toronto.edu/~kriz/cifar.html
dixonpriceReg	Regression	https://www.sfu.ca/~ssurjano/dixonpr.html
levyReg	Regression	Generation of points in N dimensions corresponding to y=f(x) where f is https://www.sfu.ca/~ssurjano/levy.html
linearReg	Regression	Generation of points in N dimensions corresponding to y=x
mnistNas	Classification	http://yann.lecun.com/exdb/mnist/
polynome2Reg	Regression	Generation of points in N dimensions corresponding to y=sum(x_i^2)
saddleReg	Regression	https://en.wikipedia.org/wiki/Saddle_point

1.9 Introduction

The goal off the evaluator module is to have a set of objects which can helps us to run our task on different environments and with different system settings/properties.



1.10 BalsamEvaluator

class `deephyper.evaluator._balsam.`**BalsamEvaluator**(*run_function*, *cache_key=None*)

Evaluator using balsam software.

Documentation to balsam : https://balsam.readthedocs.io This class helps us to run task on HPC systems with more flexibility and ease of use.

Parameters

- **run_function** (*func*) – takes one parameter of type dict and returns a scalar value.
- **cache_key** (*func*) – takes one parameter of type dict and returns a hashable type, used as the key for caching evaluations. Multiple inputs that map to the same hashable key will only be evaluated once. If `None`, then cache_key defaults to a lossless (identity) encoding of the input dict.

1.11 SubprocessEvaluator

class deephyper.evaluator._subprocess.**SubprocessEvaluator**(*run_function*, *cache_key=None*)

Evaluator using subprocess.

The `SubprocessEvaluator` use the `subprocess` package. The generated processes have a fresh memory independant from their parent process. All the imports are going to be repeated.

Parameters

- **run_function** (*func*) – takes one parameter of type dict and returns a scalar value.
- **cache_key** (*func*) – takes one parameter of type dict and returns a hashable type, used as the key for caching evaluations. Multiple inputs that map to the same hashable key will only be evaluated once. If `None`, then cache_key defaults to a lossless (identity) encoding of the input dict.

1.12 ProcessPoolEvaluator

class deephyper.evaluator._processPool.**ProcessPoolEvaluator**(*run_function*, *cache_key=None*)

Evaluator using ProcessPoolExecutor.

The ProcessPoolEvaluator use the `concurrent.futures.ProcessPoolExecutor` class. The processes doesn't share memory but they are forked from the mother process so imports done before are done repeated. Be carefull if your `run_function` is loading an package such as tensorflow it can hang.

Parameters

- **run_function** (*func*) – takes one parameter of type dict and returns a scalar value.
- **cache_key** (*func*) – takes one parameter of type dict and returns a hashable type, used as the key for caching evaluations. Multiple inputs that map to the same hashable key will only be evaluated once. If `None`, then cache_key defaults to a lossless (identity) encoding of the input dict.

1.13 ThreadPoolEvaluator

class deephyper.evaluator._threadPool.**ThreadPoolEvaluator**(*run_function*, *cache_key=None*)

Evaluator using ThreadPoolExecutor.

The `ThreadPoolEvaluator` use the `concurrent.futures.ThreadPoolExecutor` class. The processes share memory and they are forked from the mother process so imports done before are done repeated. Be carefull if your `run_function` is loading an package such as tensorflow it can hang. If your `run_function` is very fast this evaluator can be faster than `ProcessPoolEvaluator`.

Parameters

- **run_function** (*func*) – takes one parameter of type dict and returns a scalar value.
- **cache_key** (*func*) – takes one parameter of type dict and returns a hashable type, used as the key for caching evaluations. Multiple inputs that map to the same hashable key will only be evaluated once. If `None`, then cache_key defaults to a lossless (identity) encoding of the input dict.

Warning: For ThreadPoolEvaluator, note that this does not mean that they are executed on different CPUs. Python threads will NOT make your program faster if it already uses 100 % CPU time. Python threads are used in cases where the execution of a task involves some waiting. One example would be interaction with a service hosted on another computer, such as a webserver. Threading allows python to execute other code while waiting; this is easily simulated with the sleep function. (from: https://en.wikibooks.org/wiki/Python_Programming/Threading)

1.14 Search

The `search` module brings a modular way to implement new search algorithms and two sub modules. One is for hyperparameter search `deephyper.search.hps` and one is for neural architecture search `deephyper.search.nas`. The `Search` class is abstract and has different subclasses such as: `deephyper.search.ambs` and `deephyper.search.ga`.

class deephyper.search.search.**Search**(*problem*, *run*, *evaluator*, ***kwargs*)

Abstract representation of a black box optimization search.

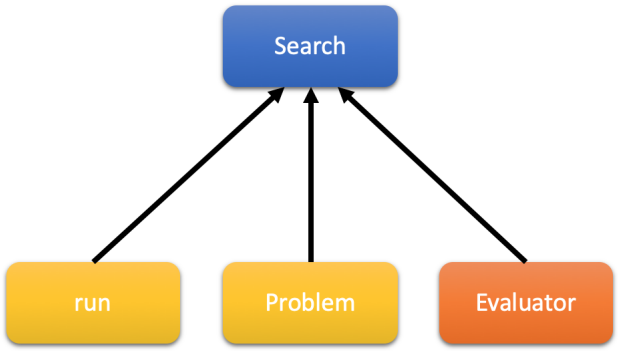
A search comprises 3 main objects: a problem, a run function and an evaluator: The *problem* class defines the optimization problem, providing details like the search domain. (You can find many kind of problems in *deephyper.benchmark*) The *run* function executes the black box function/model and returns the objective value which is to be optimized. The *evaluator* abstracts the run time environment (local, supercomputer. . . etc) in which run functions are executed.

Parameters

- **problem** (*str*) –
- **run** (*str*) –
- **evaluator** (*str*) – in [‘balsam’, ‘subprocess’, ‘processPool’, ‘threadPool’]

1.15 Hyperparameter Search (HPS)

1.15.1 Optimizer



An hyperparameter search use a problem definition which contain the characteristics of our search space, a model to evaluate and an evaluator to run the model. The model is defined inside a function (corresponding to parameter `run`). This function must have one argument which is a python `dict` object. Inside this dictionary you will have different keys corresponding to the definition of your problem. Let's see how to define a simple problem for hyperparameter search:

```
>>> from deephyper.benchmark import HpProblem
>>> Problem = HpProblem()
>>> Problem.add_dim('nunits', (10, 20), 10)
>>> print(Problem)
Problem
{'nunits': (10, 20)}

Starting Point
{'nunits': 10}
```

So the function which runs the model will receive a dictionary like `{'nunits': 10}` but the value of each key will change depending on the choices of the search. Let's see how to define a simple run function for a multi-layer Perceptron model training on mnist data.

```
'''Trains a simple deep NN on the MNIST dataset.
Gets to 98.40% test accuracy after 20 epochs
(there is *a lot* of margin for parameter tuning).
2 seconds per epoch on a K520 GPU.
'''

from __future__ import print_function

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop

def run(params):
    nunits = params['nunits]

    batch_size = 128
```

(continues on next page)

(continued from previous page)

```
    num_classes = 10
    epochs = 20

    # the data, split between train and test sets
    (x_train, y_train), (x_test, y_test) = mnist.load_data()

    x_train = x_train.reshape(60000, 784)
    x_test = x_test.reshape(10000, 784)
    x_train = x_train.astype('float32')
    x_test = x_test.astype('float32')
    x_train /= 255
    x_test /= 255
    print(x_train.shape[0], 'train samples')
    print(x_test.shape[0], 'test samples')

    # convert class vectors to binary class matrices
    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)

    model = Sequential()
    model.add(Dense(nunits, activation='relu', input_shape=(784,)))
    model.add(Dropout(0.2))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(num_classes, activation='softmax'))

    model.summary()

    model.compile(loss='categorical_crossentropy',
                  optimizer=RMSprop(),
                  metrics=['accuracy'])

    history = model.fit(x_train, y_train,
                        batch_size=batch_size,
                        epochs=epochs,
                        verbose=1,
                        validation_data=(x_test, y_test))
    score = model.evaluate(x_test, y_test, verbose=0)
    print('Test loss:', score[0])
    print('Test accuracy:', score[1])

    return -score[1]
```

Now if you want a search with the previous problem and model. Let's suppose that the problem is define inside `package_name/problem.py` and the model is define inside `package_name/mnist_mlp.py`. If you run a search like AMBS with the command line:

```
python ambs.py --problem package_name.problem.Problem --run package_name.mnist_mlp.run
```

All search can be used directly with the command line or inside an other python file. To print the parameters of a search just run `python search_script.py --help`. For example with AMBS run `python ambs.py --help`.

1.15.2 Asynchronous Model-Base Search (AMBS)

You can download the deephyper paper `here`

Environment variable to access the search on Theta: `DH_AMBS`

Asynchronous Model-Based Search.

Arguments of AMBS : * `learner`

- `RF` : Random Forest (default)
- `ET` : Extra Trees
- `GBRT` : Gradient Boosting Regression Trees
- `DUMMY` :
- `GP` : Gaussian process
- `liar-strategy`
  - `cl_max` : (default)
  - `cl_min` :
  - `cl_mean` :
- `acq-func` : Acquisition function
  - `LCB` :
  - `EI` :
  - `PI` :
  - `gp_hedge` : (default)

class deephyper.search.hps.ambs.**AMBS**(*problem*, *run*, *evaluator*, ***kwargs*)

1.15.3 Genetic Algorithm (GA)

class deephyper.search.hps.ga.**GA**(*problem*, *run*, *evaluator*, ***kwargs*)

1.16 Neural Architecture Search (NAS)

1.16.1 Agent

1.16.2 Environment

1.16.3 Model

Baseline

Run

Space

Operations

Node

class deephyper.search.nas.model.space.node.**Node**(*name=''*, *ops=[]*, *index=None*)
This class represents a node of a graph.

Parameters

- **name** (*str*) – node name.
- **ops** (*list*) – possible operations of node.
- **index** (*int*) – index corresponding to the operation choosen for this node among the possible operations

Block

class deephyper.search.nas.model.space.block.**Block**
This class represent a basic group of Nodes.

Cell

class deephyper.search.nas.model.space.cell.**Cell**(*inputs=None*)
Create a new Cell object.

Parameters **inputs** (*list(Node)*) – possible inputs of the cell

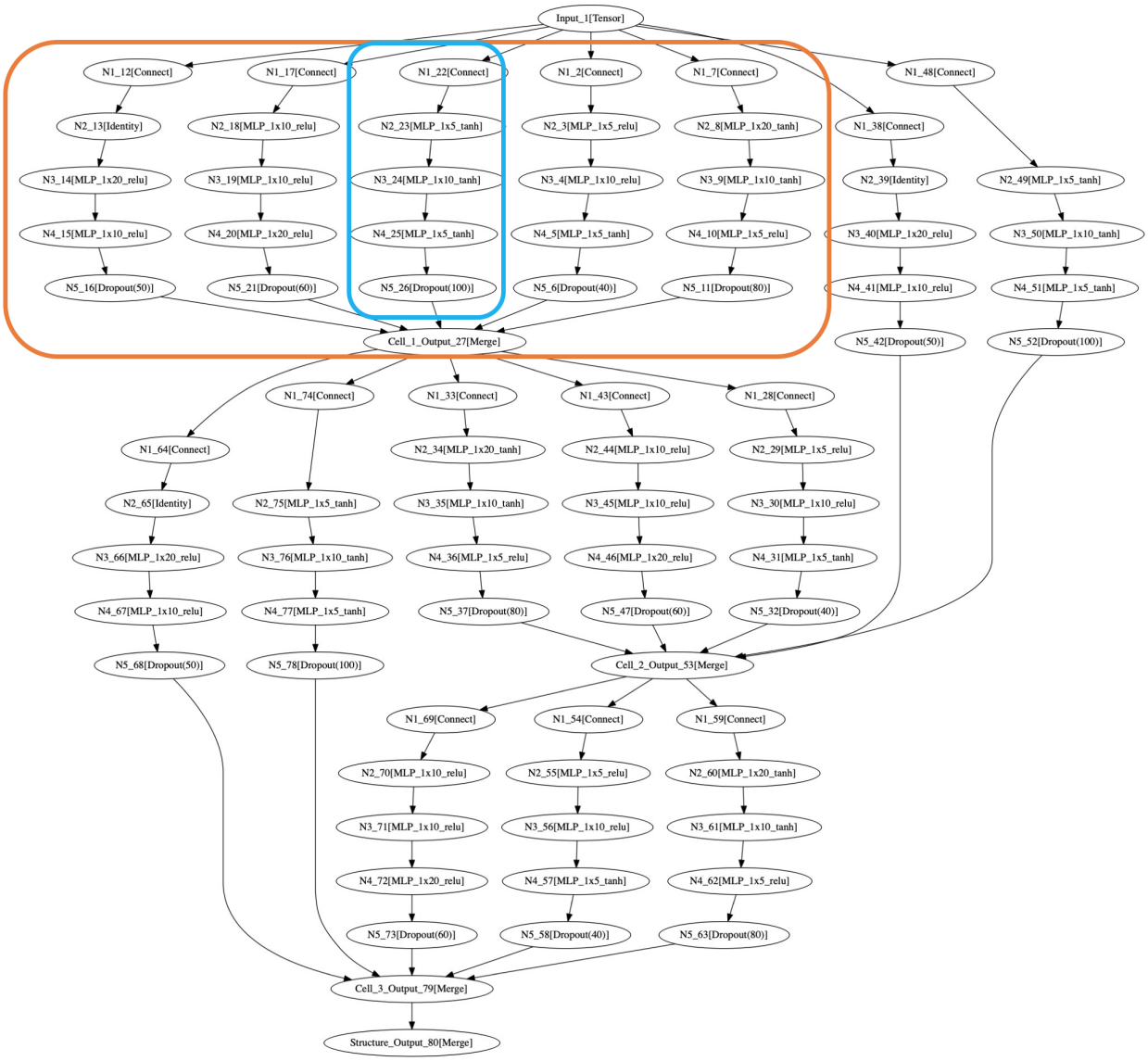
Structure

class deephyper.search.nas.model.space.structure.**KerasStructure**(*input_shape*, *output_shape*)

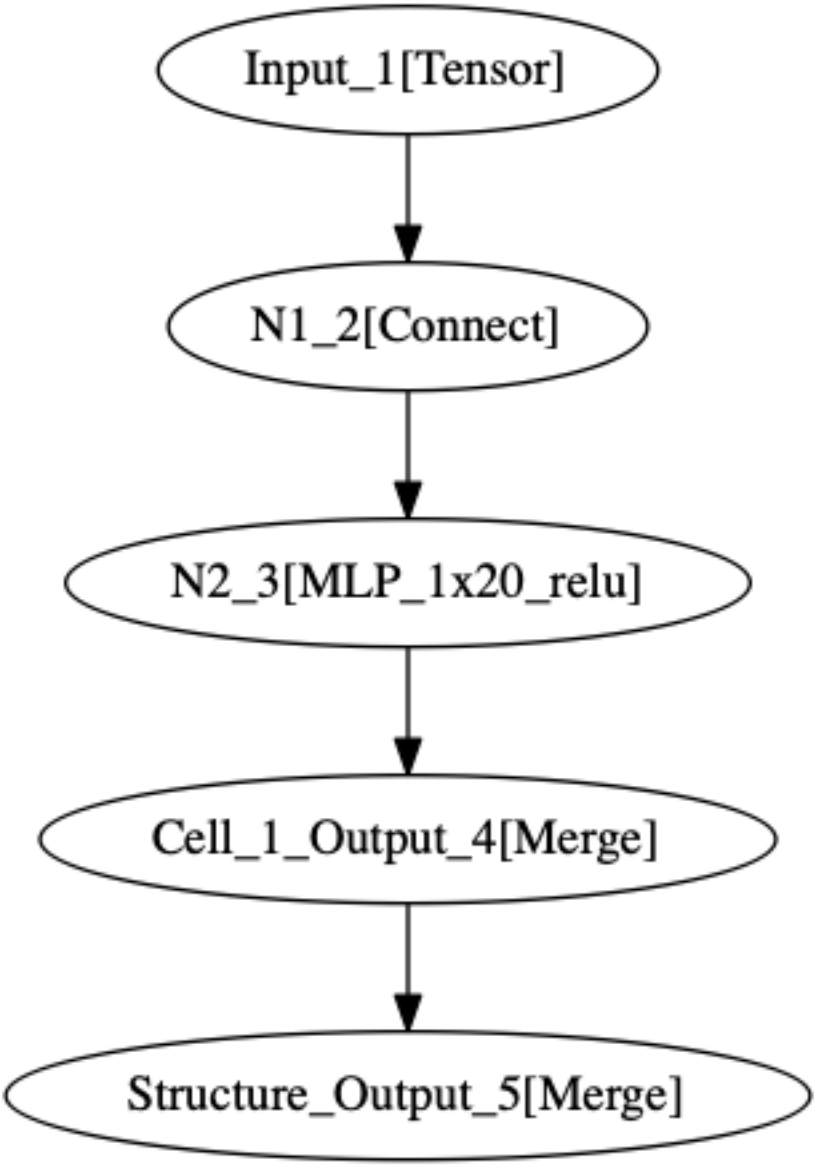
What is a Structure ?

Warning: If you want to output the dot files of graphs that you are creating with the nas api please install pygraphviz: `pip install pygrapviz`

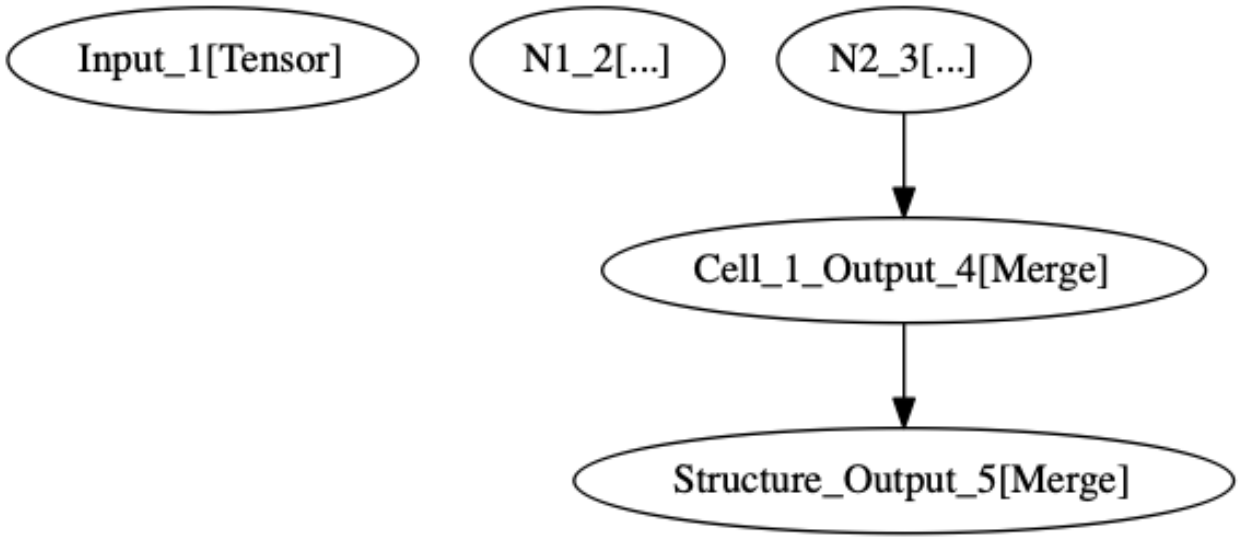
In neural architecture search we have an agent who is producing a sequence of actions to build an architecture. In deephyper we are bringing a specific API to define the action space of this agent which is located in two modules: `deephyper.search.nas.cell` and `deephyper.search.nas.operation`. Let's start with an example of a structure with the following figure:



The previous image represents a specific choice of operations in a `SequentialStructure` with 3 cells (orange), each cell contains 5 blocks (blue), and each block contains 5 nodes. Let's take a more simple structure to understand its purpose:



In the previous figure we have a choice of operations for a very simple structure called `anl_mlp_toy` which can be represented with one cell by the following figure:



In this structure we have only 1 cell which contains only 1 block. This block contains 2 nodes, the first one represents the creation of a connection (N1_2), the second one represent the creation of a multi layer Perceptron (N2_3).

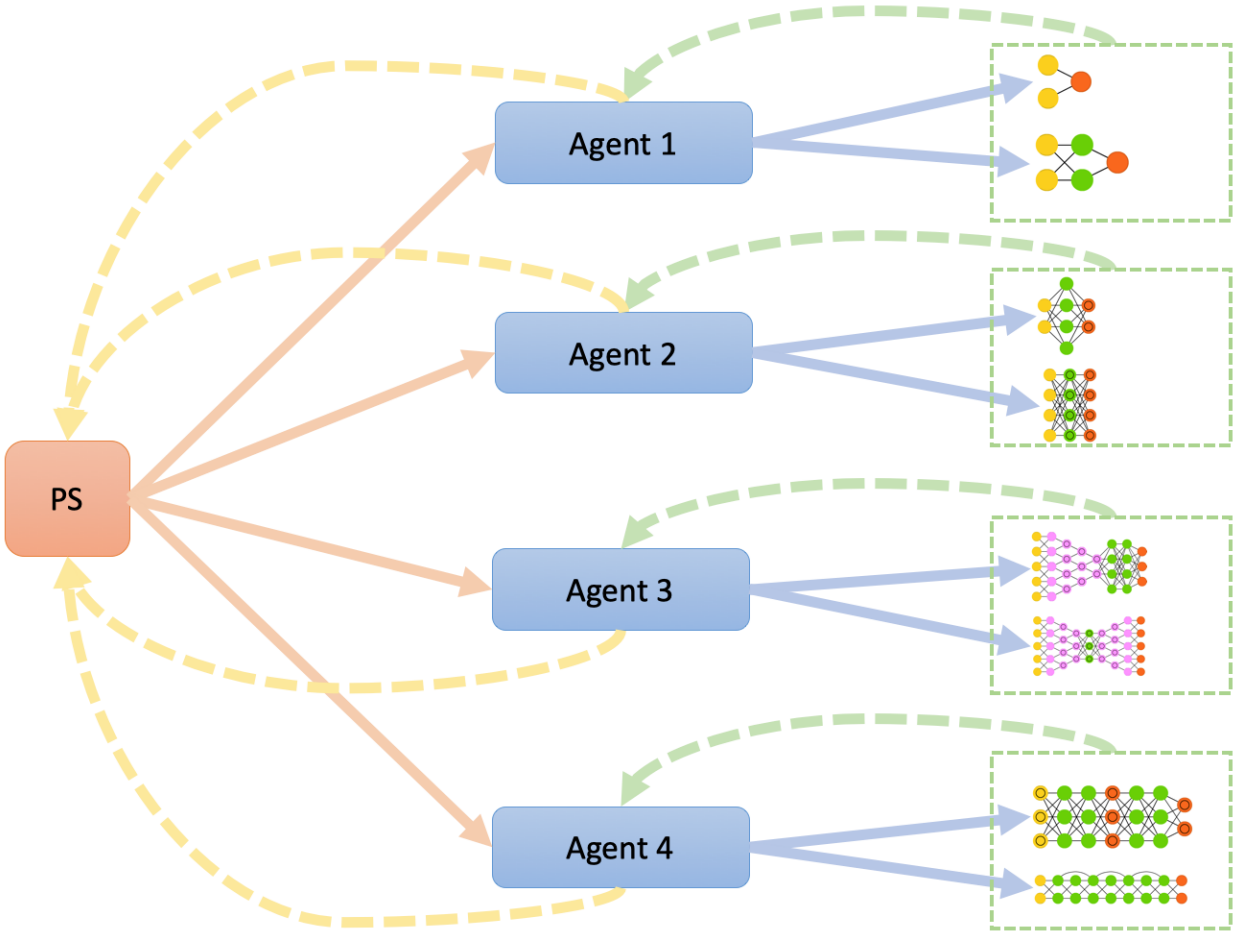
Todo: end what is a structure

Create a new Structure

Todo: how to create a new structure

Trainer

1.16.4 NAS A3C (PPO) Asynchronous



class deephyper.search.nas.ppo_a3c_async.**NasPPOAsyncA3C**(*problem*, *run*, *evaluator*, ***kwargs*)

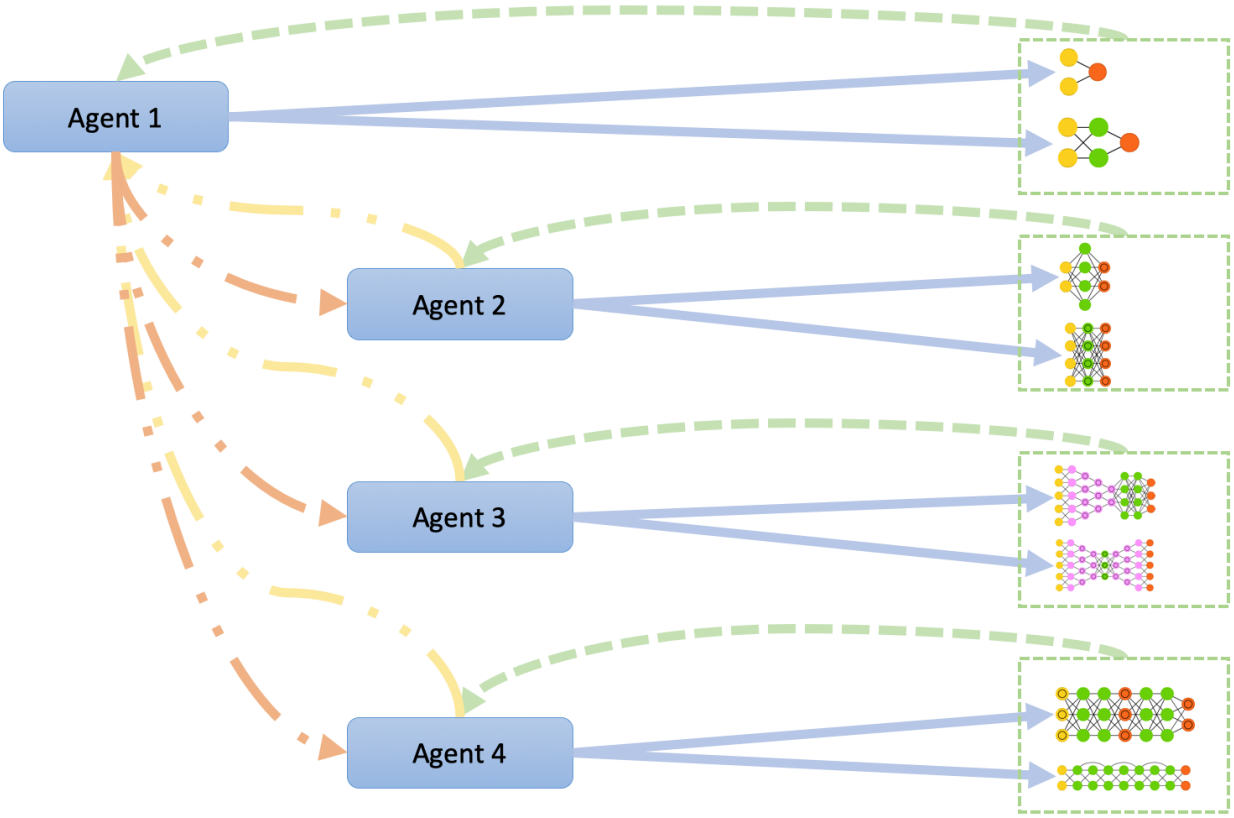
Neural Architecture search using proximal policy gradient with asynchronous optimization.

Run locally

With n agent where n = np - 1, because 1 mpi process is used for the parameter server.

```
mpirun -np 2 python ppo_a3c_async.py --problem deephyper.benchmark.nas.mnist1D.
↪problem.Problem --run deephyper.search.nas.model.run.alpha.run --evaluator␣
↪subprocess
```

1.16.5 NAS A3C (PPO) Synchronous



class deephyper.search.nas.ppo_a3c_sync.**NasPPOSyncA3C**(*problem*, *run*, *evaluator*, ***kwargs*)

Neural Architecture search using proximal policy gradient with synchronous optimization.

```
python -m deephyper.search.nas.ppo_a3c_sync --evaluator subprocess --problem
↪'deephyper.benchmark.nas.linearReg.problem.Problem' --run 'deephyper.search.nas.
↪model.run.alpha.run'
```

or with MPI to launch n agents, where n = np because all the agent are going to be synchronized with the first agent:

```
mpirun -np 2 python ppo_a3c_async.py --problem deephyper.benchmark.nas.mnist1D.
↪problem.Problem --run deephyper.search.nas.model.run.alpha.run --evaluator␣
↪subprocess
```

It is important to use the subprocess evaluator.

1.16.6 NAS Random

class deephyper.search.nas.random.**NasRandom**(*problem*, *run*, *evaluator*, ***kwargs*)

Neural Architecture search using random search.

1.17 Workflow

1.17.1 On local computer

TODO (see Quick Start for now)

1.17.2 On super computer (Theta/Cooley)

General Workflow

Load the deephyper module

```
# You can add this line in your ~/.bashrc so that
# deephyper will be automatically loaded at your login
module load deephyper
```

Check your available balsam databases with

```
balsam which --list
```

If you didn't create any database yet, let's create one

```
balsam create DB_NAME
```

Now you can start or connect to the balsam database with

```
source balsamactivate DB_NAME
```

Before running a search you need to create an balsam application corresponding to the executable. Inside deephyper we provide different applications. For example, you can access AMBS search through the environment variable `DH_AMBS`.

```
balsam app --name APPLICATION_NAME --exec EXECUTABLE_PATH
```

To see your available balsam applications do

```
balsam ls apps
```

Now you need to add a balsam job that is going to run your application with a specific configuration, for example with a set of arguments.

```
balsam job --name JOB_NAME --workflow WF_NAME --application APPLICATION_NAME --args '-
↪-problem foo.problem.Problem --run foo.run.run'
```

to see the available arguments of a specific search just run

```
python $EXECUTABLE_PATH_OF_SEARCH
# for example : python $DH_AMBS
```

You can finally run your search on Theta with

```
balsam submit-launch -q QUEUE_NAME -n NUMBER_OF_NODES -t TIME_IN_MINUTES -A PROJECT_
↪NAME --job-mode serial --wf-filter WORKFLOW_NAME
```

This last command submits the theta job to the queue for you. The workflow is always an optional parameter. If you don't give a workflow filter the balsam launcher will start all available jobs inside the database.

1.18 Tests

For automatic tests in deephyper we choosed to use the pytest framework: pytest official website.

1.18.1 Install Pytest

```
pip install -U pytest
```

1.18.2 Run Tests

```
cd deephyper/tests/
pytest
```

1.19 Indices and tables

- genindex
- modindex
- search

1.20 Doc todo list

Todo: installation on Cooley

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/deephyper/checkouts/latest/docs/quickstart/installatio
line 29.)

Todo: use hps inside python

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/deephyper/checkouts/latest/docs/quickstart/local.rst,
line 27.)

Todo: use nas inside python

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/deephyper/checkouts/latest/docs/quickstart/local.rst,
line 42.)

Todo: end what is a structure

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/deephyper/checkouts/latest/docs/search/nas/model/sp
line 56.)

Todo: how to create a new structure

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/deephyper/checkouts/latest/docs/search/nas/model/sp
line 63.)

PYTHON MODULE INDEX

d

INDEX