Executive Summary

Randopt is a Python package for machine learning experiment management, hyper-parameter optimization, and results visualization. Some of its features include:

- result logging and management,
- human-readable format,
- support for parallelism / distributed / asynchronous experiments,
- command-line and programmatic API,
- shareable, flexible Web visualization,
- automatic hyper-parameter search, and
- pure Python - no dependencies !

Simple Demo

```python
import randopt as ro

def loss(x):
    return x**2

e = ro.Experiment('myexp', {
        'alpha': ro.Gaussian(mean=0.0, std=1.0, dtype='float'),
    })

# Sampling parameters
for i in xrange(100):
    e.sample('alpha')
    res = loss(e.alpha)
    print('Result: ', res)
    e.add_result(res)

# Manually setting parameters
e.alpha = 0.00001
res = loss(e.alpha)
e.add_result(result=res,
             data={
                 'myfield': 'content ready for analysis',
                 'convergence': [1.0, 0.5, 0.4, 0.3, 0.2, 0.1]
             },
             attachment={
                'description': 'Unlike data, attachments are not for analysis.',
                'examples': [model_weights,
                             'specific pre-processed data',
                             'anything pickle-serializable, really']
             })

# Search over all experiments results, including ones from previous runs
opt = e.minimum()
print('Best result: ', opt.result, ' with params: ', opt.params)
```

Define a loss.

Instantiate experiment and parameter samplers.

Sample parameters of the experiment according to the predefined samplers.

Manually set parameter without sampling.

Register results / data / attachments as JSON Summary.

Fetch and explore saved results via programmatic API.

Plugins

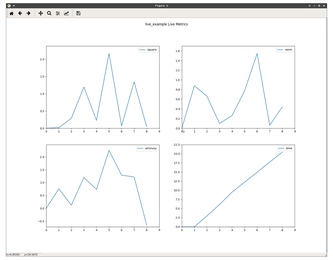
Randopt's *Experiment* class is easily extensible.

For example, plugins have been developed for

- Bayesian Hyperparameter Optimization,
- Hyperparameter Search Monitoring,
- Matplotlib-based Live Metric Tracking,
- Visdom-based (web) Live Metric Tracking, and
- Custom Analysis Templates.

Some of these plugins are freely available at

www.github.com/seba-1511/randopt_plugins



Contact

/seba-1511/randopt

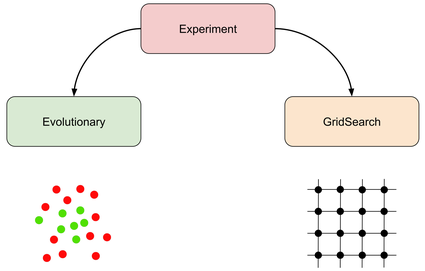
/seba1511

seba1511.com/randopt

randopt

Streamlined machine learning experiment management.

Hyperparameter Optimization



Problem
Finding good hyperparameter is almost black magic.

Solution

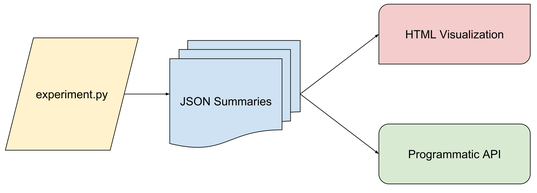
Randopt comes equiped with algorithms for automatic hyperparameter optimization. These algorithms are accessible from the programmatic API or the command-line interface, and they currently include:

- Random Search, where parameters are chosen according to a given prior distribution.
- Grid Search, where parameters are chosen according to a pre-defined grid.
- Evolutionary Search, where the previously best-performing parameters are slightly perturbed according to a given distribution.

```python
import randopt as ro

# Open existing experiment
exp = ro.Experiment('myexp', params={
    'alpha': ro.Uniform(-0.5, 0.5),
    'beta': ro.Choice([0.1, 0.2, 0.3])
})

# Instanciate experiment with search wrapper
evo = ro.Evolutionary(exp, params={
    'alpha': ro.Gaussian(0.0, 0.01),
    'beta': ro.Gaussian(0.0, 0.1)
})

# Sample parameters based on past results
evo.sample_all_params()
result = loss(evo.alpha, evo.beta)

# Add search result as a JSON Summary
evo.add_result(result)
```

```
ROPT_EXP='Evolutionary' ROPT_NAME='myexp' ropt.py python myscript.py --lr='Gaussian()'
```

Managing Experiments



Problem
Keeping track of experimental data is a hassle.

Solution

With 2 additional lines of code, randopt makes it easy to save your parameters and results (including analysis data and attachments) in the form of JSON Summaries.

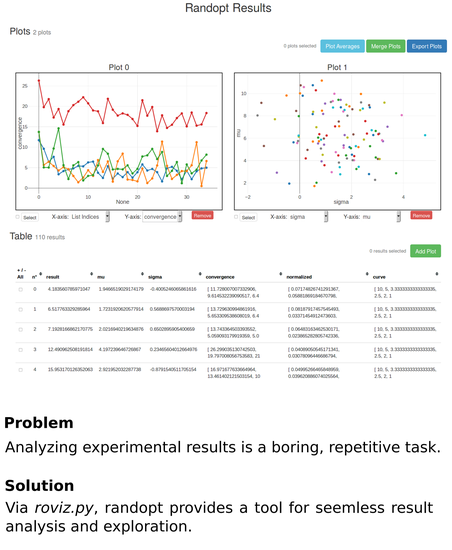
Once those JSON Summaries are saved, they can easily be analyzed using the Web visualization tool. Moreover, randopt offers a powerful programmatic API to explore and filter saved results so as to allow you to build your own visualizations.

By saving results in individual JSON files, you will be able to combine outputs from different machines by copying those files into the experiment's folder. This is ideal when working with multiple machines or and git submodules.

```python
import randopt as ro

exp = ro.Experiment(name='myexp', directory='mydir')

# Results Statistics
exp.count()
exp.all()  # Iterator over JSON Summaries

# Result Management
mini = exp.minimum()
maxi = exp.maximum()
assert mini.result < maxi.result

exp.top(10)
exp.top(10, lambda x, y: x.alpha**2 <= y.alpha**2)

# Random State Management
exp.seed(1234)
state = exp.get_state()
exp.set_state(state)
```

Example usage of the programmatic API.

Visualizing Results



Problem

Analyzing experimental results is a boring, repetitive task.

Solution

Via *roviz.py*, randopt provides a tool for seemless result analysis and exploration.

Upon running *roviz.py*, randopt will fetch all existing results from an experiment's directory, concatenate them into a single data structure, and create a dynamic local Web page.

```
roviz.py path/to/my/experiment
```

Running *roviz.py* from the command line.

From this Web page you will be able to access summaries of your results as well as create and compare interactive plots. Even better, you can easily share your insightful reports with colleagues via the *Export* function.

By making the analysis process exploratory, randopt allows you to quickly try comparisons of experimental data which you wouldn't have tried otherwise !