DASK
Distributed Summit 2021

Hyperparameter Optimization

using DASK

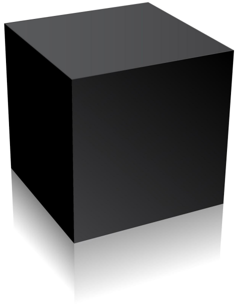
with Oríon

Xavier Bouthillier Mila

Yonggang Hu IBM

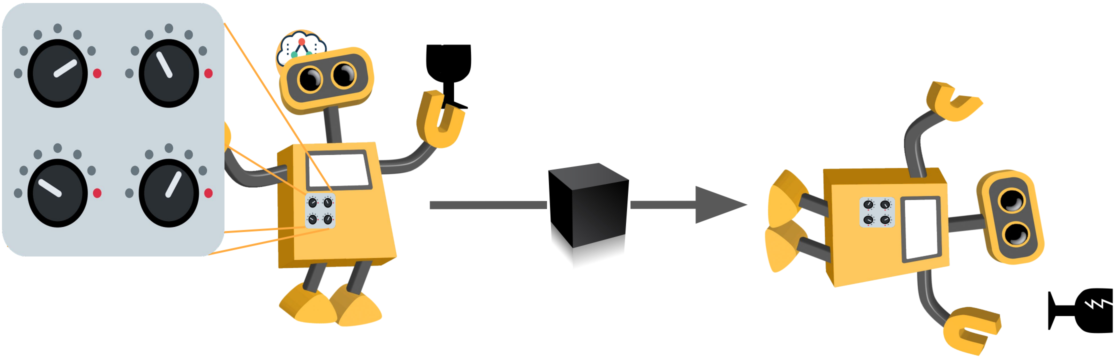
Agenda

- Introduction to black-box optimization
- Introduction to Oríon
- Integration with DASK



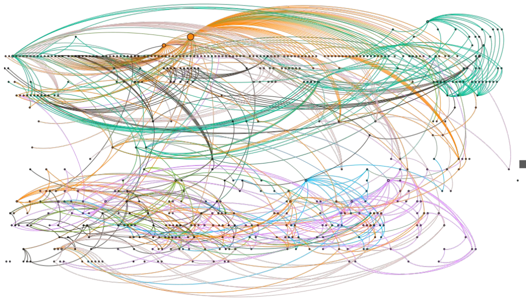
Optimization

Optimization

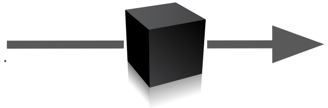


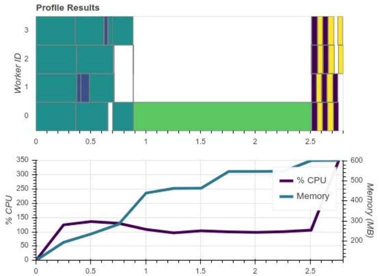
Learning well?

Optimization



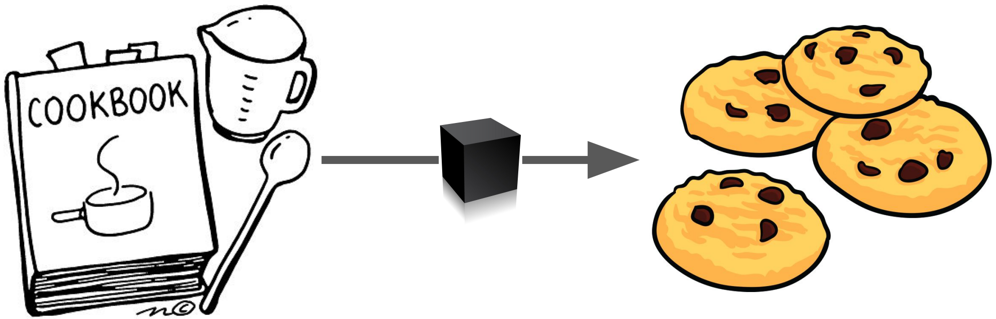
dask.org





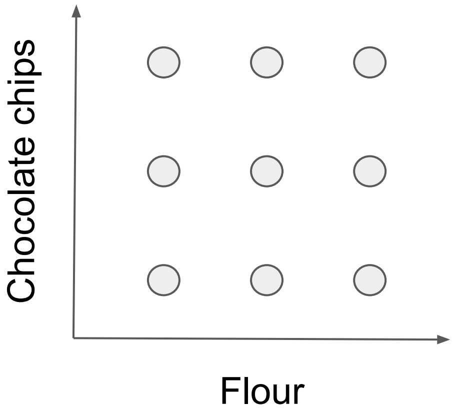
Faster?

Optimization



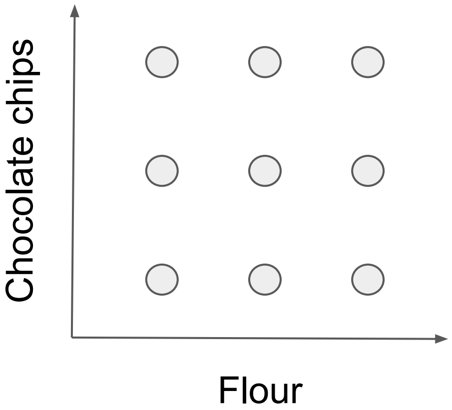
Yummy?

Optimization Search Spaces



Optimization Methods

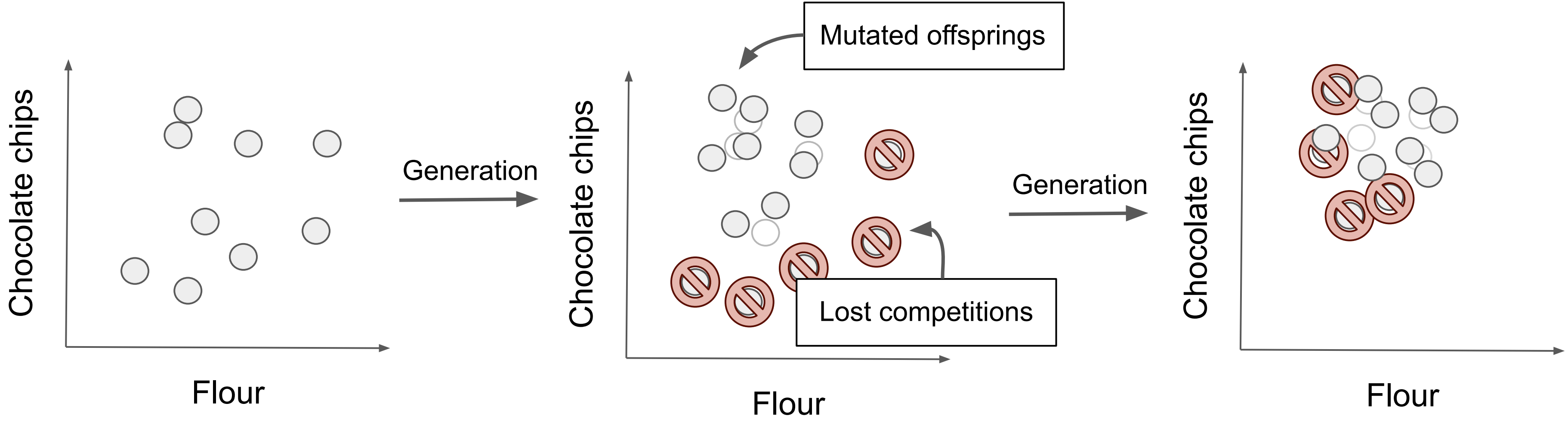
Grid Search



Random Search

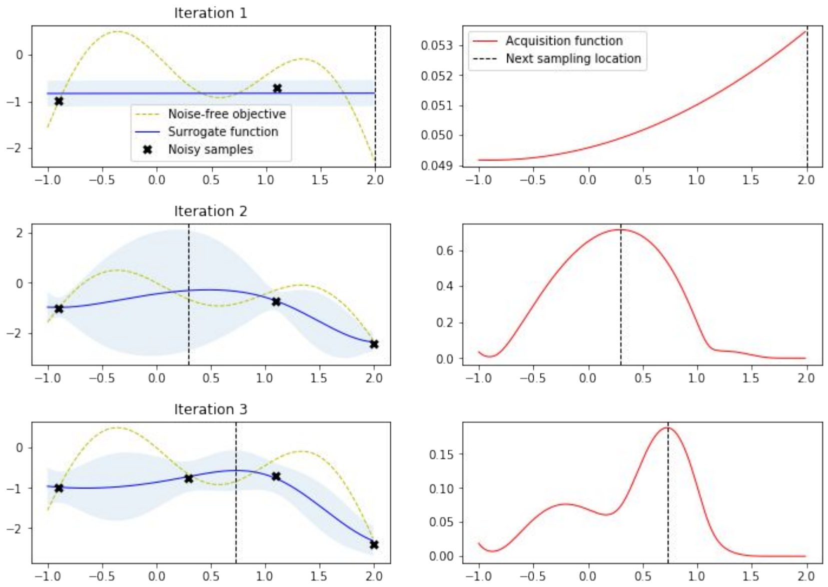
Optimization Methods

Evolutionary Algorithms



Optimization Methods

Model-Based Algorithms



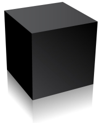
krasserm.github.io/2018/03/21/bayesian-optimization/

Design Principles of Oríon

The trigger: Finicky HPC

- No nodes for persistent processes.
- No internet connection on compute nodes.
- Unstable, you would often end up on nodes with faulty GPUs.
- Very slow file-system. Maybe due to bad usage, but whatever the reason, it was terribly slow. ¯_(ツ)_/¯

Design Principles of Orion

- Resiliency: Master-less and resilient to worker failures.
- Interoperability: Agnostic to programming language of the  .
- Modularity: Most components are ‘pluginable’.

Installing Oríon

From PyPI

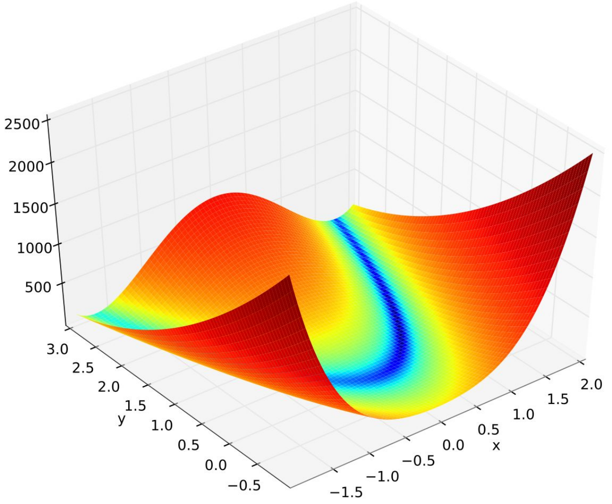
```
$ pip install orion
```

From Anaconda

```
$ conda install -c epistimio orion
```

Running Oríon

```python
def rosenbrock(x, y, a=1, b=100):
    z = (a - x)**2 + b * (y - x**2)**2

    return [
        {"name": "objective",
         "type": "objective",
         "value": z
        }
    ]
```

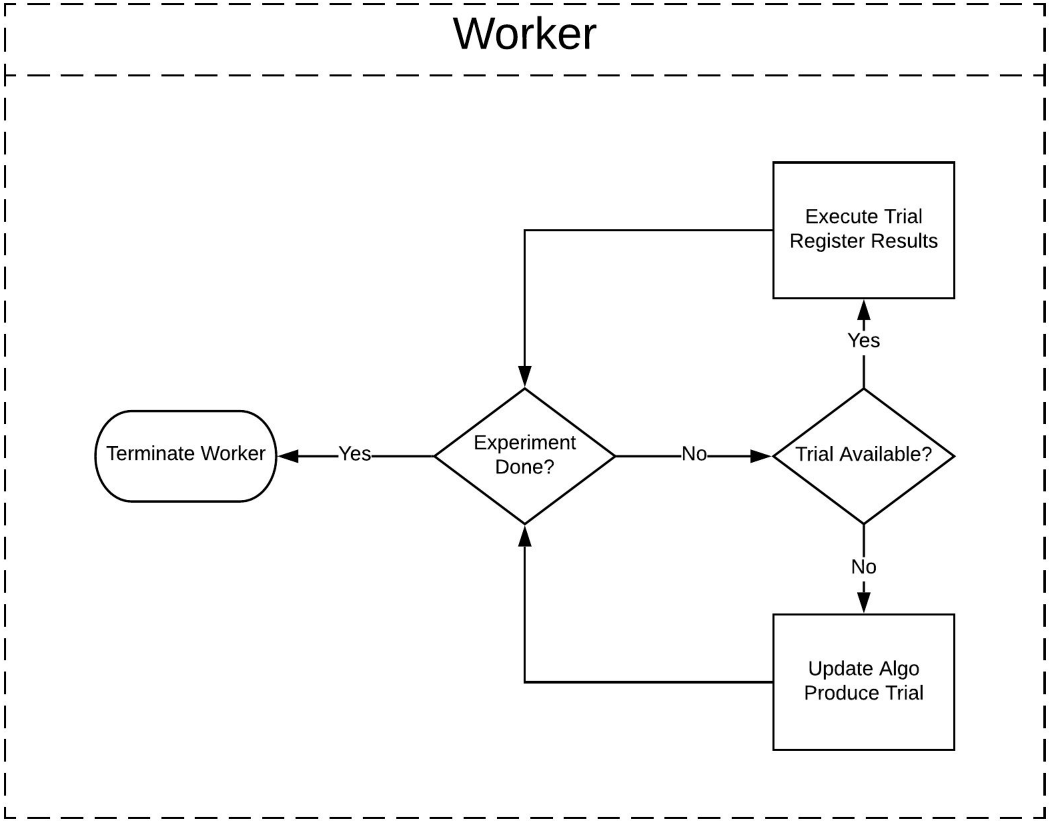


Running Oríon

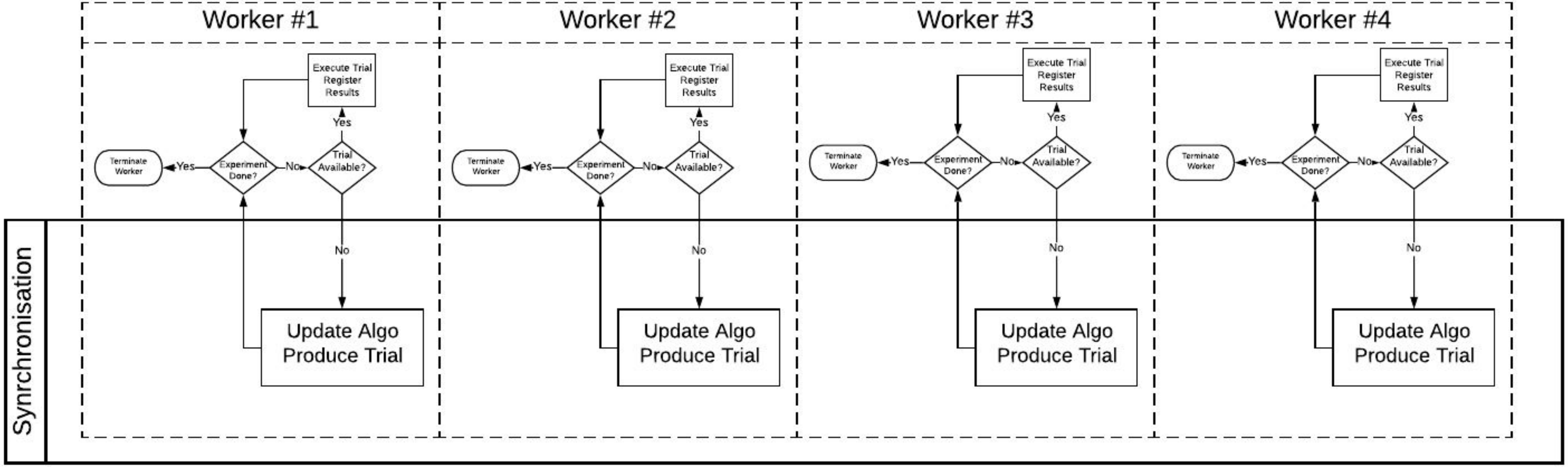
```python
def rosenbrock(x, y, a=1, b=100):
    z = (a - x)**2 + b * (y - x**2)**2

    return [
        {"name": "objective",
         "type": "objective",
         "value": z
        }
    ]
```

```python
from orion.client import build_experiment

experiment = build_experiment(
    "random-rosenbrock",
     space={
         "x": "uniform(-5, 5)",
         "y": "uniform(-5, 5)",
    },
)

experiment.workon(rosenbrock, max_trials=20)
```

Workflow


Worker
Execute Trial
Register Results
Yes
Terminate Worker
Yes
Experiment Done?
No
Trial Available?
No
Update Algo
Produce Trial

Distributed Workflow



Running Orion

```python
def rosenbrock(x, y, a=1, b=100):
    z = (a - x)**2 + b * (y - x**2)**2

    return [
        {"name": "objective",
         "type": "objective",
         "value": z
        }
    ]
```

```python
from orion.client import build_experiment

experiment = build_experiment(
    "random-rosenbrock",
     space={
         "x": "uniform(-5, 5)",
         "y": "uniform(-5, 5)",
    },
)

experiment.workon(rosenbrock, max_trials=20)
```

Running Oríon with DASK

```python
def rosenbrock(x, y, a=1, b=100):
    z = (a - x)**2 + b * (y - x**2)**2

    return [
        {"name": "objective",
         "type": "objective",
         "value": z
        }
    ]
```

```python
from orion.client import build_experiment

experiment = build_experiment(
    "random-rosenbrock",
    space={
        "x": "uniform(-5, 5)",
        "y": "uniform(-5, 5)",
    },
)

with experiment.tmp_executor("dask"):
    experiment.workon(rosenbrock, max_trials=20)
```

Running Orion with DASK

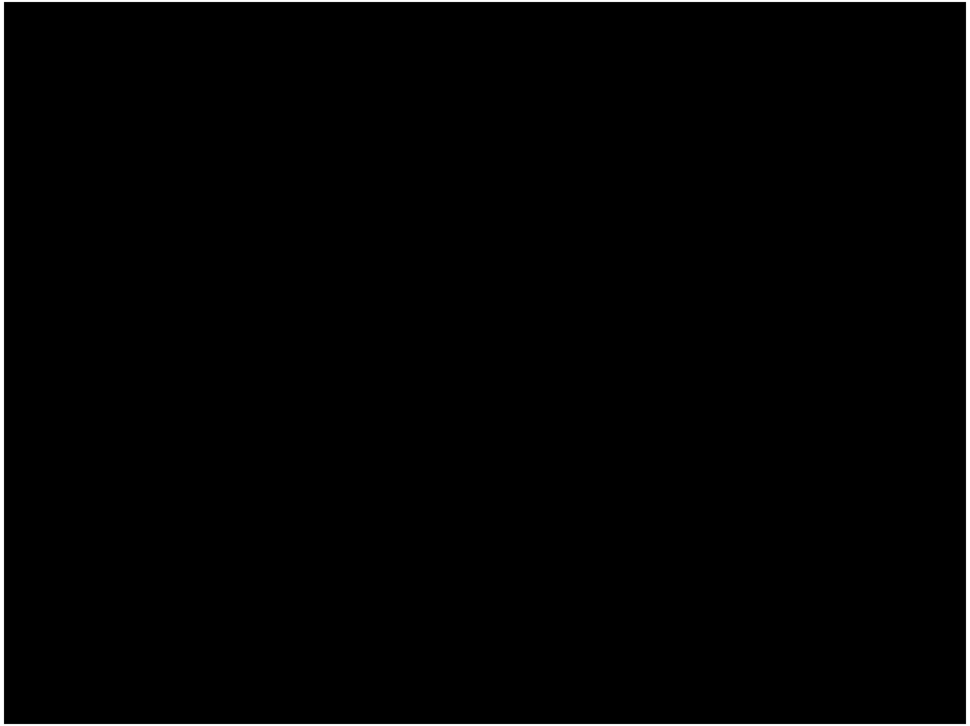
```python
def main(C, gamma, tol, class_weight):
    diabetes = datasets.load_diabetes()

    X = diabetes.data
    y = diabetes.target

    model = SVC(kernel="rbf", C=C, gamma=gamma,
                tol=tol, class_weight=class_weight)

    with joblib.parallel_backend("dask"):
        cv_results = cross_validate(model, X, y, cv=5)

    accuracy = numpy.mean(cv_results["test_score"])
    error_rate = 1 - accuracy

    return [{"name": "test_error_rate",
             "type": "objective",
             "value": error_rate}]
```

Running Oríon with DASK

```python
def main(C, gamma, tol, class_weight):
    diabetes = datasets.load_diabetes()

    X = diabetes.data
    y = diabetes.target

    model = SVC(kernel="rbf", C=C, gamma=gamma,
                tol=tol, class_weight=class_weight)

    with joblib.parallel_backend("dask"):
        cv_results = cross_validate(model, X, y, cv=5)

    accuracy = numpy.mean(cv_results["test_score"])
    error_rate = 1 - accuracy

    return [{"name": "test_error_rate",
             "type": "objective",
             "value": error_rate}]
```

```python
def hpo(n_workers=16):

    experiment = create_experiment(
        name="dask",
        max_trials=200,
        space={
            "C": "loguniform(1e-6, 1e6, precision=None)",
            "gamma": "loguniform(1e-8, 1e8, precision=None)",
            "tol": "loguniform(1e-4, 1e-1, precision=None)",
            "class_weight": "choices([None, 'balanced'])",
        },
    )

    with experiment.tmp_executor("dask", n_workers=n_workers):
        experiment.workon(main, n_workers=n_workers // 2)

    experiment.plot.regret().show()
    experiment.plot.partial_dependencies(
        params=["C", "gamma", "tol"]).show()


if __name__ == "__main__":
    hpo()
```

Running Orion with DASK



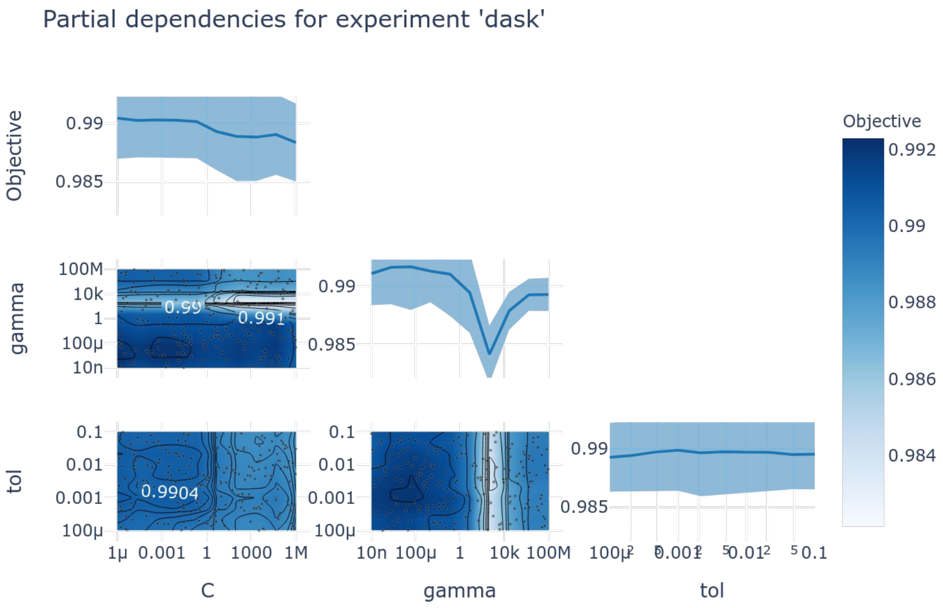
Running Orion with DASK

```
experiment.plot.regret().show()
```

Conveys how quickly the algorithm found the best hyperparameters.

Running Oríon with DASK

```
experiment.plot.partial_dependencies(
        params=["C", "gamma", "tol"]).show()
```



Conveys overview of the search space, what has been explored.

Helps identifying best optimal regions of the space.

Coming up next

Issues of master-less design

- Complexifies the implementation of algorithms. Lots of corner-cases!
- Requires more I/O with the database and scaling is hampered.
- Heavy reliance on atomicity of storage operations to handle race conditions.

Transition to Queue-based workflow

- Supports resiliency for the workers.
- Simpler to operate. Less asynchronicity to handle.
- More efficient. Less database I/O.
- Master becomes single point of failure.
- Needs persistent process on HPC.

Transition to Queue-based workflow

- Supports resiliency for the workers.
- Simpler to operate. Less asynchronicity to handle.
- More efficient. Less database I/O.
- Master becomes single point of failure.
- Needs persistent process on HPC.

We need to have the master bouncing around workers!

Documentation

orion.readthedocs.io

Source code

github.com/Epistimio/orion

Packages

pypi.org/project/orion

anaconda.org/epistimio/orion

We are looking for interns and full time developers!

Send your CV to jobs@mila.quebec if you are interested.

References

Agnihotri, Apoorv, and Nipun Batra. "Exploring bayesian optimization." Distill 5, no. 5 (2020): e26.

Bergstra, James, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. "Algorithms for hyper-parameter optimization." In 25th annual conference on neural information processing systems (NIPS 2011), vol. 24. Neural Information Processing Systems Foundation, 2011.

Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." Journal of Machine Learning Research 13, no. Feb (2012): 281-305.

Biedenkapp, André, Joshua Marben, Marius Lindauer, and Frank Hutter. "Cave: Configuration assessment, visualization and evaluation." In International Conference on Learning and Intelligent Optimization, pp. 115-130. Springer, Cham, 2018.

Bouthillier, Xavier, and Gaël Varoquaux. "Survey of machine-learning experimental methods at NeurIPS2019 and ICLR2020." PhD diss., Inria Saclay Ile de France, 2020.

Eggensperger, Katharina, Marius Lindauer, and Frank Hutter. "Pitfalls and best practices in algorithm configuration." Journal of Artificial Intelligence Research 64 (2019): 861-893.

Henderson, P., R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. "Deep Reinforcement Learning that Matters. arXiv, cs." (2017).

References

Hoffer, Elad, Ron Banner, Itay Golan, and Daniel Soudry. "Norm matters: efficient and accurate normalization schemes in deep networks." arXiv preprint arXiv:1803.01814 (2018).

Hutter, Frank, Lars Kotthoff, and Joaquin Vanschoren. Automated machine learning: methods, systems, challenges. Springer Nature, 2019.

Jastrzębski, Stanislaw, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. "Width of minima reached by stochastic gradient descent is influenced by learning rate to batch size ratio." In International Conference on Artificial Neural Networks, pp. 392-402. Springer, Cham, 2018.

Van Laarhoven, Twan. "L2 regularization versus batch and weight normalization." arXiv preprint arXiv:1706.05350 (2017).

Li, Lisha, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. "Hyperband: A novel bandit-based approach to hyperparameter optimization." The Journal of Machine Learning Research 18, no. 1 (2017): 6765-6816.

Li, Liam, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. "Massively parallel hyperparameter tuning." arXiv preprint arXiv:1810.05934 (2018).

Li, Liam, and Ameet Talwalkar. "Random search and reproducibility for neural architecture search." In Uncertainty in Artificial Intelligence, pp. 367-377. PMLR, 2020.

References

Lindauer, Marius, and Frank Hutter. "Best practices for scientific research on neural architecture search." Journal of Machine Learning Research 21, no. 243 (2020): 1-18.

Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams. "Practical bayesian optimization of machine learning algorithms." arXiv preprint arXiv:1206.2944 (2012).

So, David, Quoc Le, and Chen Liang. "The evolved transformer." In International Conference on Machine Learning, pp. 5877-5886. PMLR, 2019.

Wilson, Ashia C., Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. "The marginal value of adaptive gradient methods in machine learning." arXiv preprint arXiv:1705.08292 (2017).