Maggy

Open-Source Asynchronous Distributed Hyperparameter Optimization Based on Apache Spark
"Methods that scale with computation are the future of AI"*

"The two (general purpose) methods that seem to scale ... ... are search and learning."*

* [http://www.incompleteideas.net/InclIdeas/BitterLesson.html](http://www.incompleteideas.net/InclIdeas/BitterLesson.html)

** [https://www.youtube.com/watch?v=EeMCEQa85tw](https://www.youtube.com/watch?v=EeMCEQa85tw)

Rich Sutton
(Father of Reinforcement Learning)
Spark scales with available compute!

Spark is the answer!

$1 + 1 = 3$
Distribution and Deep Learning

Generalization Error

Better Regularization Methods

Better Optimization Algorithms

Larger Training Datasets

Hyperparameter Optimization

Design Better Models

Distribution
Inner and Outer Loop of Deep Learning

Outer Loop

Search Method

Trial

HParams, Architecture, etc.

Inner Loop

Training Data

worker_1

worker_2

worker_N

\[ \Delta_1 \]

\[ \Delta_2 \]

\[ \Delta_N \]

Synchronization

Metric

http://tiny.cc/51yjdz
Inner and Outer Loop of Deep Learning

**Outer Loop**

- **Search Method**
  - Trial
  - HParams, Architecture, etc.

  **SEARCH**

**Inner Loop**

- Training Data
  - worker$_1$
  - worker$_2$
  - worker$_N$

  **LEARNING**

  - $\nabla_1$
  - $\nabla_2$
  - $\nabla_N$

  **Synchronization**

  **Metric**

http://tiny.cc/51yjdz
In Reality This Means Rewriting Training Code

Data Pipelines
Ingest & Prep

Feature Store

Explore/Design Model

Machine Learning Experiments
Hyperparameter Optimization
Ablation Studies

Data Parallel Training

Model Serving
Towards Distribution Transparency

The distribution oblivious training function (pseudo-code):

```python
def train(model_gen, hparams, ...):
    with distr_stratgy():
        model = model_gen(hparams)
        model.compile(hparams)
        data = data_gen(hparams)
        result_dict = model.fit(data)
    return result_dict
```
Towards Distribution Transparency

- Trial and Error is slow
- Iterative approach is greedy
- Search spaces are usually large
- Sensitivity and interaction of hyperparameters
Sequential Black Box Optimization

Search space

Meta-level learning & optimization

Learning Black Box

Metric
Sequential Search

- Global Controller
- Learning Black Box
- Outer Loop

Flow:
- Global Controller → Learning Black Box
- Metric → Global Controller
- Metric → Learning Black Box
Parallel Search

- Which algorithm to use for search?
- How to monitor progress?
- How to aggregate results?
- Fault Tolerance?

Search space or Ablation Study

Global Controller

Outer Loop

Learning Black Box

Parallel Workers

Queue

Metric

Trial

Trial
Parallel Search

Which algorithm to use for search?
How to monitor progress?
This should be managed with platform support!
How to aggregate results?
Fault Tolerance?
A flexible framework for asynchronous parallel execution of trials for ML experiments on Hopsworks:

ASHA, Random Search, Grid Search, LOCO-Ablation, Bayesian Optimization and more to come...
Synchronous Search
Add Early Stopping and Asynchronous Algorithms
Performance Enhancement

Early Stopping:
- Median Stopping Rule
- Performance curve prediction

Multi-fidelity Methods:
- Successive Halving Algorithm
- Hyperband
Asynchronous Successive Halving Algorithm

Animation: https://blog.ml.cmu.edu/2018/12/12/massively-parallel-hyperparameter-optimization/
Ablation Studies

Replacing the Maggy Optimizer with an Ablator:

• Feature Ablation using the Feature Store

• Leave-One-Layer-Out Ablation

• Leave-One-Component-Out (LOCO)
Challenge

How can we fit this into the bulk synchronous execution model of Spark?

**Mismatch:** Spark Tasks and Stages vs. Trials

Databricks’ approach: Project Hydrogen (barrier execution mode) & SparkTrials in Hyperopt
The Solution

Long running tasks and communication:

HyperOpt: One Job/Trial, requiring many Threads on Driver
Enter Maggy
User API

```python
sp = Searchspace(kernel=('INTEGER', [2, 8]),
                 pool=('INTEGER', [2, 8]))

def train_fn(kernel, pool):
    from maggy import KerasBatchEnd
    ...
    model.compile()
    model.fit(..., callbacks=[KerasBatchEnd(metric='acc'), tb_callback])
    ...
    return accuracy

result = experiment.lagom(train_fn, searchspace=sp,
                           optimizer='randomsearch',
                           num_trials=5, name='demo',
                           direction='max')
```
class CustomOptimizer(AbstractOptimizer):
    def initialize(self):
        pass
    def get_suggestion(self, trial=None):
        # Return trial, return None if experiment finished
        pass
    def finalize_experiment(self, trials):
        pass

class CustomEarlyStop(AbstractEarlyStop):
    def earlystop_check(self, to_check, finalized_trials, direction):
        pass
Ablation API

```python
ablation_study = AblationStudy('titanic_train_dataset',
                             label_name='survived')

ablation_study.features.include('pclass', 'fare')

ablation_study.model.layers.include('my_dense_two',
                                     'my_dense_three')

ablation_study.model.layers.include_groups([['my_dense_two',
                                             'my_dense_four']])

ablation_study.model.layers.include_groups(prefix='my_dense')
```
Ablation API

```python
ablation_study.model.set_base_model_generator(base_model_generator)

def training_function(dataset_function, model_function, reporter):
    tf_dataset = dataset_function(epochs, batch_size)
    model = model_function()
    model.compile(...)

    cb = [KerasBatchEnd(reporter, metric='acc')]
    history = model.fit(tf_dataset, callbacks=cb, epochs=5, steps_per_epoch=30)

    return float(history.history['acc'][-1])
```
## Results

### Hyperparameter Optimization Task

<table>
<thead>
<tr>
<th>Method</th>
<th>Best Accuracy (Std)</th>
<th>Trials (Std)</th>
<th>Trials Stopped (Std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASHA</td>
<td>0.8136 (0.02)</td>
<td>442 (0.0)</td>
<td>0 (0.0)</td>
</tr>
<tr>
<td>RS-ES</td>
<td>0.7958 (0.01)</td>
<td>120 (60.7)</td>
<td>90 (65.3)</td>
</tr>
<tr>
<td>RS-NS</td>
<td>0.7747 (0.04)</td>
<td>36 (0.0)</td>
<td>0 (0.0)</td>
</tr>
</tbody>
</table>

### ASHA Validation Task

<table>
<thead>
<tr>
<th>Method</th>
<th>Best Accuracy (Std)</th>
<th>Trials (Std)</th>
<th>Trials Stopped (Std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASHA</td>
<td>0.7004 (0.03)</td>
<td>422 (38.69)</td>
<td>0 (0.0)</td>
</tr>
<tr>
<td>RS-ES</td>
<td>0.5438 (0.12)</td>
<td>112 (7.53)</td>
<td>63 (6.43)</td>
</tr>
<tr>
<td>RS-NS</td>
<td>0.5306 (0.28)</td>
<td>40 (4.51)</td>
<td>0 (0.0)</td>
</tr>
</tbody>
</table>

---

Cuda Conv Net

ASHA Experiment
Conclusions

- **Avoid** iterative Hyperparameter Optimization
- Black box optimization is **hard**
- State-of-the-art algorithms can be deployed **asynchronously**
- **Maggy**: platform support for automated hyperparameter optimization and ablation studies
- **Save** resources with asynchronism
- **Early stopping** for sensible models
What’s next?

- More algorithms
- Distribution Transparency
- Comparability/reproducibility of experiments
- Implicit Provenance
- Support for PyTorch
Acknowledgements

Thanks to the entire Logical Clocks Team 😊

Contributions from colleagues:
Robin Andersson  @robzor92
Sina Sheikholeslami  @cutlash
Kim Hammar  @KimHammar1
Alex Ormenisan  @alex_ormenisan

- **Maggy**
  https://github.com/logicalclocks/maggy
  https://maggy.readthedocs.io/en/latest/

- **Hopsworks**
  https://github.com/logicalclocks/hopsworks
  https://www.logicalclocks.com/whitepapers/hopsworks

- **Feature Store**: the missing data layer in ML pipelines?
  https://www.logicalclocks.com/feature-store/