Asynchronous Hyperparameter Tuning and Ablation Studies with Apache Spark

Sina Sheikholeslami
Distributed Computing Group, KTH Royal Institute of Technology

October 16 2019
The Machine Learning System

Problem Definition
Data Preparation
Model Selection

Dataset

Machine Learning Model

Optimizer

Repeat if needed

Model Training

Evaluate
Artificial Neural Networks

Input Layer

Hidden Layer

Output Layer
How We Study the Brain

• Early 19th Century, ablative brain surgeries by Jean Pierre Flourens (1794 - 1867)
Ablation for Machine Learning?

- Dataset
- Machine Learning
- Model
- Optimizer

Problem Definition
Data Preparation
Model Selection

Repeat if needed

Evaluation

floors area rooms price

Model Training

Optimizer
“Too frequently, authors propose many tweaks absent proper ablation studies … Sometimes just one of the changes is actually responsible for the improved results … this practice misleads readers to believe that all of the proposed changes are necessary.”

(Lipton & Steinhardt, “Troubling Trends in Machine Learning Scholarship”)
Example: Layer Ablation (1/6)

The Base Model

Accuracy: 78%
Example: Layer Ablation (2/6)

Accuracy: 73%
Example: Layer Ablation (3/6)
Example: Layer Ablation (4/6)

Accuracy: 67%
Example: Layer Ablation (5/6)

The Base Model
Example: Layer Ablation (6/6)

Accuracy: 63%
Ablation Study

Evaluate

Machine Learning System

Ablation

New Dataset / Model Configuration
Hyperparameter Tuning

Evaluate

New Hyperparameter Values

Machine Learning System

Hyperparameter Tuner
System Experimentation (Search)

Machine Learning System

Evaluate

Global Experiment Controller

New Trial
Better Parallel

• Ability to train better models, faster
• Ability to modify and inspect, easier

(“Parallel Training” - by Maxim Melnikov)
Parallelization in Practice

Machine Learning
Deep Learning

Parallel Processing

TensorFlow
Keras

(TensorFlow, the TensorFlow logo and any related marks are trademarks of Google Inc.)
Hopworks

Open-source Platform for Data-intensive AI

[Diagram of Hopworks ecosystem with various components and integrations]
Hopworks

Open-source Platform for Data-intensive AI

What is Hopworks?
https://tinyurl.com/y4ze79d4
ML/DL in Hopsworks

Bottleneck, due to
• iterative nature
• human interaction
Spark and Bulk Synchronous Parallel Model

[Diagram of Spark and Bulk Synchronous Parallel Model]

- Task 11
- Task 12
- Task 13
- ... (multiple tasks)
- Task 1N

- Task 21
- Task 22
- Task 23
- ... (multiple tasks)
- Task 2N

- Task 31
- Task 32
- Task 33
- ... (multiple tasks)
- Task 3N

HDFS

Metrics 1

Metrics 2

Metrics 3

Barrier

Driver
Example: Synchronous Hyperparameter Search

```
<table>
<thead>
<tr>
<th>Task_11</th>
<th>Task_21</th>
<th>Task_31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task_12</td>
<td>Task_22</td>
<td>Task_32</td>
</tr>
<tr>
<td>Task_13</td>
<td>Task_23</td>
<td>Task_33</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Task_1N</td>
<td>Task_2N</td>
<td>Task_3N</td>
</tr>
</tbody>
</table>

HDFS

Metrics_1

Metrics_2

Metrics_3

Driver

Wasted Compute

Spark

11

21

31

22

32

23

33

...
Critical Requirements

• Parallel execution of trials
• Support for early stopping of trials
• Support for global control of the experiment
• Resilience to stragglers
• Simple, “Unified” User & Developer API
Maggy

An Open-source Framework for Asynchronous Computation on top of Apache Spark
Key Idea: Long Running Tasks
Maggy Core Architecture
Back to Ablation
LOCO: Leave One Component Out

• A simple, “natural” ablation policy: an implementation of an ablator

• Currently supports Feature Ablation + Layer Ablation
Feature Ablation

• Uses the Feature Store to access the dataset metadata
• Generates Python callables that once called, will return modified datasets
  • Removes one-feature-at-a-time

<table>
<thead>
<tr>
<th>floors</th>
<th>area</th>
<th>rooms</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>floors</th>
<th>rooms</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Layer Ablation

• Uses a base model function
• Generates Python *callables* that once called, will return modified models
  • Uses the model configuration to find and remove layer(s)
  • Removes one-layer-at-a-time (or one-layer-group-at-a-time)
(Example Notebook Available!)

Ablation User & Developer API
User API: Initialize the Study and Add Features

```python
import tensorflow as tf
import maggy
from maggy.ablation import AblationStudy

ablation_study = AblationStudy('titanic_train_dataset',
                               training_dataset_version=1,
                               label_name='survived')

ablation_study.features.include('pclass', 'fare')
```
def base_model_generator():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Dense(64, activation='relu'))
    model.add(tf.keras.layers.Dense(64, name='my_dense_two', activation='relu'))
    model.add(tf.keras.layers.Dense(32, name='my_dense_three', activation='relu'))
    model.add(tf.keras.layers.Dense(32, name='my_dense_four', activation='relu'))
    model.add(tf.keras.layers.Dense(2, name='my_dense_sigmoid', activation='sigmoid'))
    # output layer
    model.add(tf.keras.layers.Dense(1, activation='linear'))
    return model
User API: Setup Model Ablation

```python
# set base model generator
ablation_study.model.set_base_model_generator(base_model_generator)

# include layers
ablation_study.model.layers.include('my_dense_two', 'my_dense_three',
                                     'my_dense_four', 'my_dense_sigmoid')

# add a layer group using a list
ablation_study.model.layers.include_groups(['my_dense_two', 'my_dense_four'])

# add a layer group using a prefix
ablation_study.model.layers.include_groups(prefix='my_dense')
```
def training_fn(dataset_function, model_function):
    import tensorflow as tf
    epochs = 5
    batch_size = 10
    tf_dataset = dataset_function(epochs, batch_size)
    model = model_function()
    model.compile(optimizer=tf.train.AdamOptimizer(0.001),
                   loss='binary_crossentropy',
                   metrics=["accuracy"])

    history = model.fit(tf_dataset, epochs=5, steps_per_epoch=30, verbose=0)
    return float(history.history["acc"][0])
User API: Lagom!

```python
result = experiment.lagom(map_fun=training_fn, experiment_type='ablation',
                          ablation_study=ablation_study,
                          ablator='loco',
                          name='Titanic-LOCO')
```

------ LOCO Results ------
BEST Config Excludes {"ablated_feature": "fare", "ablated_layer": "None"} -- metric 0.6766666730244955
WORST Config Excludes {"ablated_feature": "None", "ablated_layer": "Layers prefixed my_dense"} -- metric 0.3533333403 368791
AVERAGE metric -- 0.5800000042275146
Total Job Time 43 seconds
Developer API: Policy Implementation (1/2)

class AbstractAblator(ABC):

    def __init__(self, ablation_study, final_store):
        self.ablation_study = ablation_study
        self.final_store = final_store
        self.trial_buffer = []

    @abstractmethod
    def get_number_of_trials(self):
        pass

    @abstractmethod
    def get_dataset_generator(self, ablated_feature=None, dataset_type='tfrecord'):
        pass

    @abstractmethod
    def get_model_generator(self, ablated_layer):
        pass
@abstractmethod
def initialize(self):
    pass

@abstractmethod
def get_trial(self, ablation_trial=None):
    pass

@abstractmethod
def finalize_experiment(self, trials):
    pass
Hyperparameter Tuning: User API

```python
from maggy import Searchspace
from maggy import experiment

# 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]))

def train_fn(kernel, pool, dropout, reporter):
    # This is your training iteration loop
    For i in range(nr_iterations):
        ...
        # add maggy reporter to heartbeat the metric
        reporter.broadcast(metric=accuracy)
        reporter.log('Current acc: {}'.format(accuracy))
        ...
    # Return the final metric
    return accuracy

# Lagom maggy experiment
result = experiment.lagom(train_fn,
                           searchspace=sp,
                           optimizer='randomsearch',
                           num_trials=5,
                           name='demo',
                           direction='max')
```
Hyperparameter Tuning: Developer API

```python
# Developers implement abstract class

class CustomOptimizer(AbstractOptimizer):
    def __init__(self):
        super().__init__()

    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
```
Maggy is Open-source

• Code Repository: https://github.com/logicalclocks/maggy

• API Documentation: https://maggy.readthedocs.io/en/latest/
Next Steps

• More Ablators
• More Tuners
• Support for More Frameworks
Thank you! 😊

(Example Notebook Available!)

Thanks to the entire Logical Clocks Team 😊

Specially:

Moritz Meister
Jim Dowling
Robin Andersson
Kim Hammar
Alex Ormenisan

@morimeister
@jim_dowling
@robzor92
@KimHammar1
@alex_ormenisan

@logicalclocks
@hopsworks

GitHub

https://github.com/hopshadoop/maggy
https://maggy.readthedocs.io/en/latest/
https://logicalclocks.com/whitepapers/