Parallelizing of ML Hyperparameter Tuning with Ad-hoc SSH Clusters

Saturday, May 30, 2020

The development of an AI DNN model requires specifying numerous parameters and options, known as hyperparameters, involved in compiling and fitting the DNN model. The AI DNN is based on hyperparameters which defines the NN model and training. A typical set of hyperparameters for NN include the number and size of the hidden layers, weight initialization scheme, learning rate and its decay, dropout and gradient clipping threshold, etc. A further distinction occurs when considering data and how the model is trained. The DNN model learns through the training process (e.g., based on the NN graph it defines the weights) and although in theory could be set beforehand, it is still guesswork*.*

The quality of that DNN model depends on selecting the best or "optimal" hyperparameters. Thus, tuning the hyperparameters to achieve the best DNN model is an important as well as time consuming element in ML software development. Since there are numerous combinations of hyperparameters and of DNN modeling strategies, using a computational strategy is very appealing. For example, the scikit learn Python software package has a GridSearchCV that does an optimization on function using a grid of potential hypervariable selections. GridSearchCV does an exhaustive search over specified grid of hypervariable for an estimator, in our case a machine learning model. If you assume 10 different hyperparameter, each with 10 different potential variables, then you have 100 different combinations of hyperparameters. If each ML train/test estimation evaluation takes 1 hour, the total evaluation time on one computer would take 100 hours or over 4 days. If you could use a cluster of 100 different PC computer and parallelize the search operation it could take 1 hour, plus the time to coordinate and synchronize the parallelization operation. Thus, in the simple example reducing the train/test/evaluate time from 4 days to an hour illustrates the appeal of parallelizing the hyperparameter grid search.

Since hyperparameter tuning is computationally intensive, the need to find ways to scale the computations across multiple machines is useful. As the need to perform computations across multiple machines is not inherently supported in most programming paradigms, this introduces many new programming concerns. First, the computers must communicate across a network. Scheduling of parallel execution, and synchronization of results must be available for the programmer. Specifying the scheduling and cluster of computers to communicate with and perform computation must be available. Other features like handling errors, cloud deployment and supporting a variety of existing parallel HPC tools are important considerations in parallel computing.

This tutorial covers hyperparameter tuning in a parallel ad-hoc network of computers using a variety of Python tools, some that work on a single computer, or a cluster of computers using SSH, or an HPC found at NIST (Enki) using SLURM described [here](https://www.nist.gov/programs-projects/computation-platform-aiml). GridSearchCV uses joblib as a parallelization mechanism on a single board CPU but which can be generatlized to a CPU cluster. More on joblib parallel mechanism can be found [here](https://joblib.readthedocs.io/en/latest/parallel.html).

# Installation

This sections covers general installation of the Python essential components in the AI/ML ad-hoc cluster.

## Keras Python ML Environment

A clean installation on a Linux machine will be presented. Bash shell scrips were written to load an Anaconda Python environment, create a Conda environment, and then load all the ML Python packages required in the (non-GPU) deployment of TensorFlow and Keras. This script is

#!/bin/bash

#Install anaconda3

> cd /tmp

> curl -O https://repo.anaconda.com/archive/Anaconda3-2019.03-Linux-x86\_64.sh

> bash Anaconda3-2019.03-Linux-x86\_64.sh

\*\* INSTALL INTO /usr/local/anaconda3 on every Linux Box\*\*

# Sometimes this doesn't work, and you need to start a new terminal

> source ~/.bashrc

# Create a new anaconda3 python environment

> conda create --name py37 python=3.7.6

# Activate the new anaconda3 python environment

> source activate py37

# install the required ML packages:

> conda install -y pandas

> conda install -y numpy

> conda install -y matplotlib

> conda install -y scikit-learn

> conda install -y tensorflow

> conda install -y pyyaml

> conda install -y lxml

Of note, while you are installing anaconda3, it is best to **INSTALL INTO /usr/local/anaconda3** on every Linux Box that you want in the cluster. By default, anaconda will install in your home directory ~/anaconda3, which for NIST domain has a division folder under the home directory. IN general, Ubuntu has /home/user as the home directory. So, you can have competing home directories (e.g., /home/isd/michalos versus /home/michalos) to contend with. To alleviate this home folder problem, all anaconda installations and virtual environments were under the /usr/local/anaconda3 or /usr/local/user folders across all cluster Linux boxes. However, using /usr/local as a directory root can cause administrative privileges issues when creating a directory. To mitigate this circumstance, sudo was used to run the Anaconda installation bash shell, and then sudo was used to recursively change the owner of all the files to myself. Yes, a real headache.

sudo chown -R michalos

To install Anaconda we first use curl to download the Anaconda installation shell into the /tmp directory, and then use the anaconda 3 install shell as described [here](https://www.digitalocean.com/community/tutorials/how-to-install-anaconda-on-ubuntu-18-04-quickstart):

#Install anaconda3

> cd /tmp

> curl -O https://repo.anaconda.com/archive/Anaconda3-2019.03-Linux-x86\_64.sh

> bash Anaconda3-2019.03-Linux-x86\_64.sh

Anaconda will ask for license agreement and then for installation folder, which /usr/local/anaconda3 was used, with the rationale provided previously. During the anaconda3 installation the .bashrc terminal script is modified to add conda bin directory to the path. Upon future terminal deployment, the conda binary will be found, however, to do this in the script we need to source the .bashrc:

# This doesn't seem to work?

> source ~/.bashrc

(Not absolutely positive this works, may need to export the modified PATH environment variable after its modified.) Assuming conda binary is in the path, we create an Python environment with a 3.6 Python interpreter, that will be used to program Keras ML models:

# Create a new anaconda3 python environment

> conda create --name py37 python=3.7.7

We then activate this anaconda environment:

# Activate the new anaconda3 python environment

> source activate py37

And load the required Python packages are loaded using conda install as the installation. Not the -y is used to silently accept the installation.

# install the required ML packages:

> conda install -y pandas

> conda install -y numpy

> conda install -y matplotlib

> conda install -y scikit-learn

> conda install -y tensorflow

> conda install -y pyyaml

> conda install -y lxml

> conda install -y mathutil

Of note, the lxml and the pyyaml Python packages are for the ROS URDFDom python parsing package, which is used to load the kinematic model of the robot for which ML models are to be deployed.

The Python install of mathutil is for a math package containing support for quaternion, matrix, and vector from Blender. It can be tricky finding the package with conda but installs and works.

## Installing SSH

The initial deployment intended goal was an ad-hoc network of PCs based on SSH implementation described [here](https://www.openssh.com/). SSH itself is a software package to enable secure system over insecure networks. SSH is based on a client-server architecture where the host system the user is working on is the client and the remote system being managed is the server. SSH itself has several communication layers but of interest will only be the user authentication layer which handles the authentication and communication of a user between a SSH client and a SSH server. SSH user authentication is based on a traditional password authentication as well as public-key or host-based authentication mechanisms. It is common for ad-hoc SSH clusters to use SSH keys to automate access to parallel workers (i.e., SSH servers).

For the Linux deployment, OpenSSH sponsored by the IETF is the SSH tool set used. More documentation on OpenSSH can be found [here](https://www.openssh.com/). We will discuss deployment on Ubuntu a Debian OS. Further a command line interface is used to execute the SSH command sequence. Assuming the Linux user has sudo rights, apt is used to install OpenSSH, and then the Ubuntu firewall port 22 is opened so that SSH can communicate through it, shown below.

sudo apt install openssh-server

# open firewall port 22 for ssh

sudo ufw allow ssh

Next, SSH keys are created and used to enable authentication and communication to a remote (and local) host. The local host is used as the head or master scheduler, so SSH need to be implemented so that the user can seamlessly authenticate with the localhost. First, we create a public and private set of RSA keys for use in authentication. This should be done on the client.

Code shown below is based on documentation found [here](https://help.ubuntu.com/community/SSH/OpenSSH/Keys). To create your public and private SSH keys on the command-line:

mkdir ~/.ssh

chmod 700 ~/.ssh

ssh-keygen -t rsa

Then the public key is copied to all the remote SSH hosts so that the client SSH can seamlessly authenticate and then communicate. Each remote SSH host is a "worker". To enable seamless authentication, the OpenSSH command ssh-copy-id is used to copy the public SSH key to the remote host, then an SSH login is done which will be required authentication with the SSH key passphrase. First, the localhost SSH communication is authenticated for user "michalos":

# copy public key to SSH server listening on port 22  
> ssh-copy-id michalos@localhost  
# Make sure password accepted

> ssh michalos@localhost

# This time it should not require password authenication  
> ssh michalos@localhost

Next the remote host "onyx" is SSH enabled:

# Copy public key to SSH server on onyx  
ssh-copy-id michalos@onyx  
ssh michalos@onyx

Remember, initially the SSH client asks for a password to authenticate, which corresponds to the passphrase, not the logon password.

Note, the id\_rsa private key must have r/w permission ONLY for me the owner. Changed to see pop up dialog in Linux asking for password would go away, assumed it did not have permission to read. DONT WONT WORK RUINS SSH.

Further note, Windows supports OpenSSH in beta (providing an SSH service as well as an SSH client) but does not support SSH-copy-id.

# Deployments

Parallel processing is a mode of operation where the task is executed concurrently in multiple processors in one or more computers. Parallelization is meant to reduce the overall processing time. However, there is communication and synchronization overhead when coordinating parallel processes which can increase the overall execution time taken for small tasks rather than decreasing it. An excellent online tutorial explaining parallel computation can be found [here](https://computing.llnl.gov/tutorials/parallel_comp/).

Traditionally, software has been written for serial computation where the programming is broken into a discrete series of instructions, the program instructions are executed sequentially one after another and the execution is on a single processor. Parallel computing is the simultaneous use of multiple computing resources to solve a computational problem where programming problem is broken into discrete parts that can be solved concurrently, and each part is further broken down to a series of instructions that execute simultaneously on different processors and requires an overall control/coordination mechanism to synchronize the parallel operation.

Some definitions should help clarify the parallel concepts. Task is a logically discrete section of computational work. A task is typically a program or program-like set of instructions that is executed by a processor. A parallel program consists of multiple tasks running on multiple processors. Throughput is a timing measure to evaluate the parallel operation performance. It measures the amount of completed work against time consumed. Pipeline is an approach to breaking a task into steps performed by different processor units, with inputs streaming through, much like an assembly line; a type of parallel computing. Synchronization applies to either shared data or parallel task is the coordination of processes or threads, either to avoid conflicts of shared resources or to coordinate execution.

Parallel programs can be executed across cores, processes, threads, multiple computer on a shared network. Concurrent programming concerns operations that appear to be parallel but are sharing a computational resource (one processor) so that multiple tasks share the CPU one-at-a-time during execution. Core is an independent computer processing unit (CPU) that read and execute computer program instructions. A multi-core processor is physically a single processor with two or more cores. Cores can help parallelize computer programs to increase throughput. A process is the instance of a computer program that is being executed by one or many threads on one or more cores. A thread is the smallest unit of concurrent task programming that executes within a process. A network connects a group of computer systems and other computing hardware devices using a standard communication channel (International Organization for Standardization, 1989).

Modern computers are parallel in architecture with multiple processors/cores. Parallel software is specifically intended for parallel hardware with multiple cores, threads, etc. In most cases, serial programs that run on modern computers "waste" potential computing power.

The goal of the hyperparameter optimization task is to use as much computing power as possible in a parallel operation. The hyperparameter optimization problem is computing intensive but can be easily parallelized into distributed hyperparameter evaluation operations. Computing power includes CPU cores, GPU on the host execution computer as well as remote computing resources accessible via a network or cloud.

## Python multiprocessing module

Varying success was found with different parallelization Python packages. The most success was achieved with Ray, so it's use will be covered. Dask and ipyparallel on PowerPC HPC were also attempted but with limited success.

## RAY

Ray is a Python module that has Read the docs documentation which can be found [here](https://ray.readthedocs.io/en/latest/autoscaling.html). The Berkeley rise lab also has many blog entries on Ray found [here](https://rise.cs.berkeley.edu/blog/). The blog entry "Programming in Ray: Tips for first-time users" was especially useful found [here](https://rise.cs.berkeley.edu/blog/ray-tips-for-first-time-users/).

### Install

Because the ML model used Keras (really TensorFlow 2 ), you should install the required Python packages as described earlier. Plus, you must make the Python interpreter you use match the Ray version you install. This can be a problem if for example, you installed TensorFlow for Python 3.6 and Ray distribution is now Python 3.7. You must watch for mismatch of versions. Installing older Python interpreter versions is explained in read the docs.

The latest version of Ray is geared for Python 3.7 and is installed by:

pip install -U ray

pip install -U ray [debug]

pip install -U ray [dashboard]

If you are not using the latest Ray python version, you must download a wheel to install, for example, pip install of the Python 3.6 shown below.

pip install -U ray -0.6.2-cp36-cp36m-manylinux1\_x86\_64.whl

**Be careful**. Python interpreter sub-versions must match across platforms, so 3.7.3 DOES NOT MATCH 3.7.7. You will need to include the correct subversion on the anaconda virtual environment creations.

### Configuration

The initial Ray cluster involved Linux PC two machines with the following hardware characteristics:

|  |  |  |
| --- | --- | --- |
| PC: | Dell Precision 7720 (07B1) | Dell Latitude E6530 |
| Architecture: | x86\_64 | x86\_64 |
| CPU(s): | 8 | 4 |
| Thread(s) per core: | 2 | 2 |
| Core(s) per socket: | 4 | 2 |
| Model name: | Intel(R) Core(TM) i7-7820HQ CPU @ 2.90GHz | Intel(R) Core(TM) i5-3360M CPU @ 2.80GHz |
| CPU MHz: | 1692.115 | 1196.138 |
| OS | Ubuntu 18.04 | Ubuntu 16.04 |

Then a YAML configuration file was configured for ad-hoc SSH cluster containing the two Linux CPU nodes that creates many workers. The file was modified from the sample provided by Ray's read the docs. Had intermittent success and failure with the YAML configuration file, so your results may vary.

# An unique identifier for the head node and workers of this cluster.

cluster\_name: wtfray

## NOTE: Typically for local clusters, min\_workers == initial\_workers == max\_workers.

# The minimum number of workers nodes to launch in addition to the head # node. This number should be >= 0.

# Typically, min\_workers == initial\_workers == max\_workers.

min\_workers: 10

# The initial number of worker nodes to launch in addition to the head node.

# Typically, min\_workers == initial\_workers == max\_workers.

initial\_workers: 1

# The maximum number of workers nodes to launch in addition to the head node.

# This takes precedence over min\_workers.

# Typically, min\_workers == initial\_workers == max\_workers.

max\_workers: 100

# Autoscaling parameters.

# Ignore this if min\_workers == initial\_workers == max\_workers.

autoscaling\_mode: default

target\_utilization\_fraction: 0.8

idle\_timeout\_minutes: 120

# This executes all commands on all nodes in the docker container, # and opens all the necessary ports to support the Ray cluster.

# Empty string means disabled. Assumes Docker is installed.

docker:

image: "" # e.g., tensorflow / tensorflow:1.5.0-py3

container\_name: "" # e.g. ray\_docker

# If true, pulls latest version of image. Otherwise, `docker run` will only pull the image

# if no cached version is present.

pull\_before\_run: True

run\_options: [] # Extra options to pass into "docker run"

# Local specific configuration.

provider:

type: local

head\_ip: lightning

worker\_ips: [192.168.1.9]

# How Ray will authenticate with newly launched nodes.

auth:

ssh\_user: michalos

ssh\_private\_key: ~/.ssh/id\_rsa

# Leave this empty.

head\_node: {

KeyName: ~/.ssh/id\_rsa

}

# Leave this empty.

worker\_nodes: {

KeyName: ~/.ssh/id\_rsa

}

# Files or directories to copy to the head and worker nodes. The format is a # dictionary from REMOTE\_PATH: LOCAL\_PATH, e.g.

file\_mounts: {

# "/path1/on/remote/machine": "/path1/on/local/machine", # "/path2/on/remote/machine": "/path2/on/local/machine", }

# List of commands that will be run before `setup\_commands`. If docker is # enabled, these commands will run outside the container and before docker # is setup.

initialization\_commands: []

# List of shell commands to run to set up each nodes.

setup\_commands:

- echo 'export PATH="/usr/local/anaconda3/envs/py37/bin:$PATH"' >> ~/.bashrc

- source /usr/local/anaconda3/etc/profile.d/conda.sh; conda activate py37

- pip install -U ray

# Custom commands that will be run on the head node after common setup.

head\_setup\_commands: []

# Custom commands that will be run on worker nodes after common setup.

worker\_setup\_commands: []

# Command to start Ray on the head node. You don't need to change this.

head\_start\_ ray \_commands:

- ray stop

- ulimit -c unlimited && ray start --head --redis-port=6379 --autoscaling-config=~/ray\_bootstrap\_config.yaml

# Command to start Ray on worker nodes. You don't need to change this.

worker\_start\_ray\_commands:

- ray stop

- ray start --address=$RAY\_HEAD\_IP:6379

One change was to add the SSH private key to the head\_node and the remote\_node based on a template I found on the internet. This added onyx (the worker node) to the cluster. You must have a space after the ":" delimiter or the Key and Value are combined(!) into one key. You can check what is happening after reading and interpreting your YAML configuration file, by examining the file ~/ray\_bootstrap\_config.yaml. That's how I found this feature. It's unclear if this even works or helps, but someone on the Internet did it, and it seemed like a good idea.

# Leave this empty. DIDN'T!

head\_node: {

KeyName: ~/.ssh/id\_rsa

}

# Leave this empty. DIDN'T!

worker\_nodes: {

KeyName: ~/.ssh/id\_rsa

}

When the diagnostic messages did spew forth on the command "Ray up" during the connecting to the worker onyx, it appeared that it wasn't using the correct Anaconda virtual environment (py37). So various attempts at different YAML configurations were taken, with no clear success.

setup\_commands:

    - echo 'export PATH="/usr/local/anaconda3/envs/py37/bin:$PATH"' >> ~/.bashrc

    - source /usr/local/anaconda3/etc/profile.d/conda.sh; conda activate py37

    - pip install -U ray

However, manual connecting of the onyx worker to the head Ray seemed to always work, once the header was underway.

ray start --address='192.168.1.3:6379' --redis-password='5241590000000000'

### Starting the Ray cluster

The "ray up" command uses a YAML configuration file to start cluster. In our case the cluster is an ad-hoc SSH network of PCs.

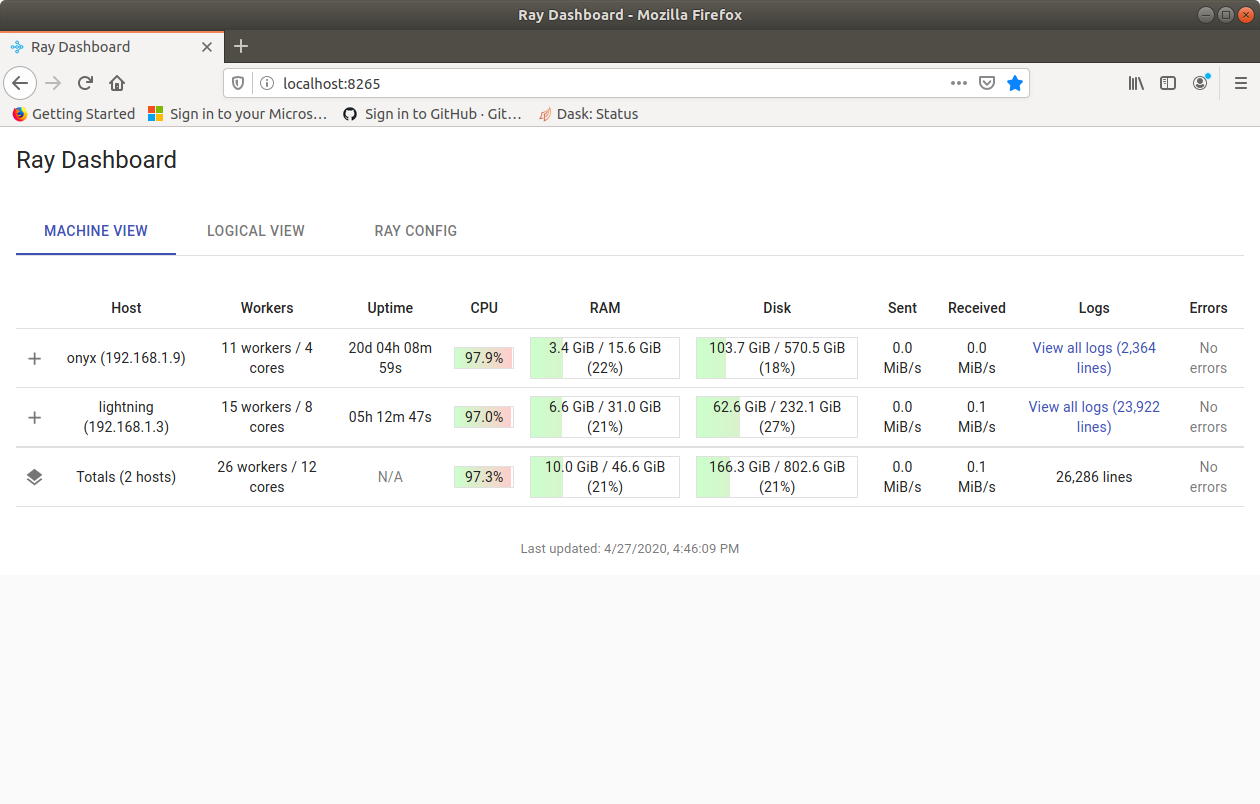
(py37) michalos@lightning:/usr/local/michalos/Py37$ ray up ./raytwoconfig.yaml

There is a lot of messy diagnostics spewing forth that follows. For example: quoting [here](https://ray.readthedocs.io/en/latest/autoscaling.html): "You may see a message like:

bash: cannot set terminal process group (-1): Inappropriate ioctl for device bash: no job control in this shell This is a harmless error. If the cluster launcher fails, it is most likely due to some other factor"

Unfortunately, sometimes a worker Linux PC wouldn't join the Ray cluster, so it was done manually as instructed by the Ray startup diagnostics.

To see if the Ray deployment launch is working, it's best to use the dashboard by putting the URL localhost:8265 (or port 8266) into a Web browser. This action will pop up the Ray dashboard and you should see two PC's in the cluster, which was the goal, shown below. It may not be eye candy, but it is very serviceable and understandable.



### Example Python Parallel Program

A test program from [here](https://medium.com/distributed-computing-with-ray/easy-distributed-scikit-learn-training-with-ray-54ff8b643b33) was used to validate that all the nodes in the cluster were being used. There are lots of Ray test source code found [here](https://github.com/ray-project/ray/blob/master/python/ray/tests/test_joblib.py) under the github site. Cluster parallelism was done by visually monitoring the web browser dashboard.

A brief overview of the simplicity of using the Scikit learn joblib backend to parallelize computation across the cluster will be discussed. The Ray package joblib parallelism approach involved three elements:

1. starting the head and workers in the cluster on the command line (using ray up),
2. importing the ray module into the Python program, and

import ray

from ray.util.joblib import register\_ray

register\_ray()

1. using the Ray parallel backend for scikit optimize joblib.

ray.init(address="192.168.1.3:6379")

with parallel\_backend('ray'):

search.fit(x, y)

where 192.168.1.3:6379 was the localhost IP. The details are not readily apparent, but the Ray cluster worked, given the following versions of Python interpreter and modules.

|  |  |
| --- | --- |
| Module | Version |
| Python interpreter | 3.7.7 |
| conda | 4.6.11 |
| ray | 0.8.4 |
| scikit-learn | 0.22.1 |

Once the two PC cluster was established as working, the Python interpreters' versions were matched up, life was much smoother, and this test Ray cluster ran. Again, it was not trouble free for me, as among problem encountered, Ray complained of a version mismatch again between the head and the remote Python interpreter. Rebooting a Linux PC caused its IP to change on the local home network (due to the pandemic quarantine).

## Sklearn GridSearchCV applied to the Optimal Hyperparameters for the AI/ML Problem

Once it was established that Ray would parallelize on an ad-hoc SSH cluster properly, it was applied to the AI/ML problem being researched. In general, ML models are based on numerous hyperparameters that can affect the accuracy of the learned model. For a ML model, there is no one "best" setting for these for all datasets. To get the optimal accuracy, the hyperparameters need to be tuned for each application and each dataset. Except for trivial cases, the assignment and evaluation of different hyperparameter options precludes anything but an automated approach. Yet even with an automated ML approach, hyperparameter search can be computationally expensive, especially if you are searching over a large hyperparameter space and dealing with numerous hyperparameters.

Scikit-learn is a Python module integrating a wide range of machine learning algorithms (Pedregosa, 2011). The intention of Scikit-learn package is to bring machine learning to non-specialists (scikit-learn.org, 2020). The automated machine learning packages used from sci-kit optimize were: RandomizedSearchCV, GridSearchCV, and Bayesian Search. Examples of various application of sklearn optimization can be found [here](https://www.programcreek.com/python/example/91146/sklearn.model_selection.RandomizedSearchCV). RandomizedSearchCV and GridSearchCV use the same basic template to evaluate the hyperparameters to find the best set. On the other hand, Scikit-learn Bayesian Search uses a different template for evaluation. The Scikit-learn user guide for "Tuning the hyper-parameters of an estimator" can be found [here](https://scikit-learn.org/stable/modules/grid_search.html#grid-search).

Scikit-learn does leverage performance opportunities on the current machine to minimize the computation required of hyperparameter searching. Scikit-learn exploits concurrency on the current machine by using all the cores through parallelism using the Python joblib package. Joblib instantiates jobs that run on multiple CPU cores. The parallelism of these jobs is limited by the number of CPU cores available on that node.

However, Scikit-Learn parallelism is adaptable by just changing the joblib's backend. Ray's implementation of a joblib's backend parallelism allows a GridSearchCV program to train and evaluate all the ML model using all the cores of a cluster. Ray handles all the difficult parallelism program allowing a GridSearchCV distributed application to run on multiple nodes by handling the scheduling of ML evaluation tasks across multiple machines. To use GridSearchCV joblib parallelism, we use an informally managed networks of machines using SSH as configured for Ray earlier.

Below is the primary of GridSearchCV application to the KerasRegressor ML model hyperparameters.

def main(argv):

global config

x\_train, y\_train, x\_test, y\_test = datagen.readDataFile(config["model\_dir"] + '/kindata.csv', verbose=True)

params = dict(activation=['relu', 'linear'],

optimizer=['adam', 'adagrad', 'nadam'],

batch\_size=[1, 128, 1024, datagen.numrows],

numneurons=[10, 64, 1024],

numlayers=[5, 20, 100],

learningrate=[0.01, 0.001],

epochs=[5, 10, 50, 100],

)

regressor = tf.keras.wrappers.scikit\_learn.KerasRegressor \

(build\_fn=baseline\_model,

batch\_size=numrows,

epochs=epochs,

verbose=1

)

# https://medium.com/swlh/hyper-parameter-tuning-for-keras-models-with-scikit-learn-library-dba47cf41551

rscv = GridSearchCV(regressor,

param\_grid=params,

refit=True,

cv=3,

verbose=2)

# Train model using randomized hyperparameter values

start = time.time()

ray.init(address="192.168.1.3:6379")

with parallel\_backend('Ray'): # added line.

rscv\_results = rscv.fit(x\_train, y\_train)

end = time.time()

elapsed = end - start

hours, rem = divmod(end - start, 3600)

minutes, seconds = divmod(rem, 60)

print(

"optimization time" + "Prediction time = {:0>2}:{:0>2}:{:07.4f}".format(int(hours), int(minutes), seconds))

print ('Best score ' + str(rscv\_results.best\_score\_))

print ('best option' + str(rscv\_results.best\_params\_))

print ('All options ' + str(params))

Of note, typical metrics useful in classification ML model, i.e., accuracy, precision, etc., are not useful in regression ML models. More in depth discussion on metrics as applied to different ML models can be found [here](https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce).

Overall embedded inside the KerasRegressor model is the compilation for the metrics as:

model.compile(optimizer=optfcn, loss=lossfcn, metrics=[r\_square, rmse])

Below is code that corresponds to the custom metric functions r\_square, rmse, and mse:

# root mean squared error (rmse) for regression (only for Keras tensors)

def rmse(y\_true, y\_pred):

return K.sqrt(K.mean(K.square(y\_pred - y\_true), axis=-1))

# mean squared error (mse) for regression (only for Keras tensors)

def mse(y\_true, y\_pred):

return K.mean(K.square(y\_pred - y\_true), axis=-1)

# coefficient of determination (R^2) for regression (only for Keras tensors)

def r\_square(y\_true, y\_pred):

SS\_res = K.sum(K.square(y\_true - y\_pred))

SS\_tot = K.sum(K.square(y\_true - K.mean(y\_true)))

return (1 - SS\_res / (SS\_tot + K.epsilon()))

## Observations

In the test local home network (during pandemic quarantine), the Linux worker IP changed! This will wreak havoc and wasted time.

For a novice, one of the major annoyances using sklearn Grid and Random Hyperparameter search in general is being clueless on the time until completion. This oblivious is especially annoying as there is no checkpoint mechanism to stop/resume a sklearn hyperparameter search so once the search is underway you must wait until done or abort the search. An example of lengthily duration required of exhaustive ML hyperparameter even for a mundane evaluation is given in the FAQ section.

### Addition Ray Packages

Ray provides other Python libraries to scale applications, including Tune, a scalable hyperparameter tuning library and RLlib, a scalable reinforcement learning library.

Ray Tune-sklearn is a package that integrates Ray Tune's hyperparameter tuning and scikit-learn's models, allowing users to optimize hyerparameter searching for sklearn using Tune's schedulers (more details found [here](https://github.com/ray-project/tune-sklearn)). Tune-sklearn provides additional benefits if specifying a scheduler with an estimator that **supports early stopping**. The list of estimators that can be supported from scikit-learn can be found in scikit-learn's documentation found [here](https://scikit-learn.org/stable/modules/computing.html#strategies-to-scale-computationally-bigger-data)

Using Tune, Ray supports distributed TensorFlow for ML modeling, which can be found [here](https://docs.ray.io/en/latest/raysgd/raysgd_tensorflow.html). However, at this time no application was attempted. In addition, Ray also has module support for PyTorch, also uninvestigated currently.

# FAQ

Q: Why GridSearchCV and why not hyperopt?

Based on the Python parallelization packages available (ray and dask especially), GridSearchCV joblib was used as a first step in parallelizing the hyperparameter optimization for clusters locally and on the cloud.

A hyperopt parallelization module with examples is available on github and found [here](https://github.com/hyperopt/hyperopt). This library uses Apache spark to parallelize. Although intriguing, this has not been attempted at this point in time.

Q: My remote host has anaconda3 in wrong pat, how do I fix this?

You might as get always load anaconda into /usr/local/anaconda3 on every Linux PC. This brings a few challenges as /usr/local is administrative privileged, but first, you should just remove existing anaconda and conda packages that are installed under you home directory, and reinstall under /usr/local. To remove the existing anaconda installation, do the following.

conda install anaconda-clean

Remove all Anaconda-related files and directories without being prompted to delete each one:

anaconda-clean –yes

Open a terminal window, and then remove your entire Anaconda directory, which has a name such as anaconda3

rm -rf ~/anaconda3.

Removing Anaconda path from .bash\_profile

export PATH="/Users/xxx/anaconda3/bin:$PATH"

Replace /Users/xxx/anaconda3/ with your actual path.

Q: What is the easiest way to load anaconda into /usr/local folder?

I suggest the following, depending on your .

1. You can install the anaconda distribution but load it into /usr/local/anaconda3 and then change the ownership of all the files and folders under anaconda3 to you.

chown -R ownername:groupname foldername

For more details see [here.](https://www.gnu.org/software/coreutils/manual/html_node/chown-invocation.html)

1. If you can, you could change the ownership of /usr/local to all with a

chmod 777 /usr/local

Q: How do I downgrade Ray installation to support Python 3.6 ?

Combination of uninstall, download and scp copy of Ray for python 3.6 wheel to onyx,

cd /usr/local

sudo chown -R michalos anaconda3

(gzrcsplugin\_env) michalos@lightning:~/Downloads$ scp  
./ray-0.6.2-cp36-cp36m-manylinux1\_x86\_64.whl michalos@onyx:/home/michalos

On remote machine (onyx) bring up terminal, run

(gzrcsplugin\_env): source activate gzrcsplugin\_env

(gzrcsplugin\_env): pip uninstall ray  
(gzrcsplugin\_env): pip uninstall ray [debug]  
(gzrcsplugin\_env): pip uninstall ray [dashboard]  
(gzrcsplugin\_env): pip install -U ray -0.6.2-cp36-cp36m-manylinux1\_x86\_64.whl

Q: How do you change the python version in Anaconda virtual environment?

Ray complained that a local and a remote host had different subversions of the Python interpreter, i.e.,

RuntimeError: Version mismatch: The cluster was started with:

Ray: 0.8.4

Python: 3.7.3

This process on node 192.168.1.13 was started with:

Ray: 0.8.4

Python: 3.7.7

A quick change is to set the anaconda virtual environment and then install the desired python version. Anaconda will do the removal of the old python interpreter and the install of the new version:

(py37) $ source activate py37

(py37) $ conda install python=3.7.3

. . .

The following packages will be DOWNGRADED:

python 3.7.7-hcf32534\_0\_cpython --> 3.7.3-h0371630\_0

readline 8.0-h7b6447c\_0 --> 7.0-h7b6447c\_5

You can verify with the following terminal code:

(py37) $ `which python` --version

Python 3.7.3

Q: Verified firewall not running on either cluster machine

Running the Ray parallelization library, you may see this warning.

If you have trouble connecting from a different machine, check that your firewall is configured properly.

Assuming you have ufw (uncomplicated firewall) installed on Ubuntu, you can check if it is running:

(py37) $ sudo ufw status  
[sudo] password for michalos:  
Status: inactive

If its active you would need to open the appropriate ports for host, dashboard, workers, etc. This is application dependent.

Q: What version of Python is the default interpreter when logging in?

Because ROS is installed and need Python 2.7 it is the default Python interpreter. This is a pain. Here is how to remotely tell what version of Python is the default interpreter when logging in (using ssh):

> ssh michalos@onyx python –version

2.7

> ssh michalos@onyx source activate py37; python –version

3.7

Q: How many grid combinations are there for a list or params?

Assume your Hyperparameter grid contains the following options.

params = dict(activation=[**'relu'**, **'linear'**],

optimizer=[**'adam'**, **'adagrad'**, **'nadam'**],

batch\_size=[1, 128, 1024, numrows],

numneurons=[10, 64, 1024],

numlayers=[5, 20, 100],

learningrate=[0.001],

epochs=[10, 50, 100],

lossfcn=[**'mean\_squared\_error'**],

)

For each dictionary list element, the number of possible combinations is the number of elements N. So for two dictionary list elements, it is an AND of the two elements, which corresponds to a multiplication, e.g., N1xN2

The number of combinations corresponds to the multiplication of the number of elements in each dictionary list element, i.e., 2x3x4x3x3x1x3x1=648 alternative hyperparameter options. You can double check this number of combinations using scikit learn (sklearn) ParameterGrid class which constructs a list of alternatives from the given param dictionary list.

grid = ParameterGrid(params)

n = len(grid)

Assuming 648 models using all the hyperparameter fits, and assuming each ML model train takes 10 minutes, this correspond to 648x10/1440 = 4.5 days of evaluation on a single processor. If you add just 2 cross validation evaluations, that is fitting with different combination of the training data, you know would take 9 days.

Q: How do I monitor progress of a GridSearchCV?

The exists a GridSearchCV progress bar found [here](https://pactools.github.io/auto_examples/plot_grid_search.html?highlight=gridsearchcvprogressbar). It was not tested using a Parallel Cluster. Various parallelization Python packages incorporate progress as part of their

During the KerasRegressor fit, a series of a callbacks are invoked whenever a state has occurred (start train, start epoch, end train, end epoch, etc.) A class can inherit and override functions of the Keras callback class to monitor progress. A simple

class MLCallbacks(tf.keras.callbacks.Callback):

counter = 0

def \_\_init\_\_(self, \*args, \*\*kwargs):

super(tf.keras.callbacks.Callback, self).\_\_init\_\_(\*args, \*\*kwargs)

self.increment()

self.start = time.time()

self.stats = []

@staticmethod

def increment():

MLCallbacks.counter += 1

@staticmethod

def count():

return MLCallbacks.counter

def on\_train\_end(self, logs={}):

self.stats.append(str(self.count()))

end = time.time()

elapsed = end - self.start

hours, rem = divmod(end - self.start, 3600)

minutes, seconds = divmod(rem, 60)

self.stats.append(**"{:0>2}:{:0>2}:{:07.4f}"**.format(int(hours), int(minutes), seconds))

self.stats.append(self.model.optimizer.\_\_class\_\_.\_\_name\_\_)

# self.stats.append(str(self.params['activation']))

self.stats.append(str(self.params[**'batch\_size'**]))

self.stats.append(str(self.model.layers[1].units))

self.stats.append(str(len(self.model.trainable\_weights) - 2))

# self.stats.append(str(self.params['learningrate']))

self.stats.append(str(**' '**.join(self.params[**'metrics'**])))

self.stats.append(str(self.params[**'epochs'**]))

self.stats.append(str(self.model.loss))

self.stats.append(str(statistics.mean(self.model.history.history[**'loss'**])))

print(**','**.join(self.stats) + **'**\n**'**)

Later the callback is inserted into the argument list for the fit function:

rscv\_results = rscv.fit(x\_train, y\_train, callbacks=[MLCallbacks()])

Note, a threading lock was added to the custom callback but resulted in Ray cluster penalization reporting this error:

TypeError: can't pickle \_thread.lock objects Remove threading lock from Callback.

One omission of hyperparameter is the Activation function, e.g., "relu". However, this hyperparameters is not easily accessible to novices (although we tried) and you should start [here](https://github.com/philipperemy/keract).

Further, indexing of the callback using a cluster involves multi-cloning the GridSearchCV process which renders the callback class to be "unique" on each running process so the index is only incremented on the local process. Index generation by matching the current model against the total grid search space was considered, but not undertaken.

Q: What does the GridSearchCV refit parameter mean?

From [here](https://datascience.stackexchange.com/questions/45810/what-is-gridsearchcv-doing-after-it-finishes-evaluating-the-performance-of-param) : The answer is that by default GridSearchCV's last act is to expose the API of the estimator object you passed so that you can directly call things like predict() or score() on the GridSearchCV object itself. It does this by retraining the estimator against the best parameters it found during cross validation. If you want to skip this step (because, for example, you're going to go on to do more development or cross-validation afterwards) then you can pass refit=False to prevent that from happening.

Q: How do I do Scikit Learn GridSearchCV without cross validation (e.g., unsupervised learning)

Cross-validation (CV) is a technique for evaluating ML models by training several ML models on subsets of the available input data and evaluating them on the complementary subset of the data. The main reason CV is used is to prevent overfitting. Timing can be a consideration when using CV. For instance, if every training fit takes 10 minutes and you have a 6 fold cross-validation, it will take an hour to evaluate one ML model. As overfitting does not appear to be a concern and hyperparameter fitting takes 4.5 days per train, CV equal to one was desired to reduce the computational complexity.

From [here](https://stackoverflow.com/questions/44636370/scikit-learn-gridsearchcv-without-cross-validation-unsupervised-learning) : It appears that you can get rid of cross validation in GridSearchCV if you use:

cv=[(slice(None), slice(None))]

I have tested this against my own coded version of grid search without cross validation and I get the same results from both methods. I am posting this answer to my own question in case others have the same issue.

cv = [(slice(None), slice(None))]

gs = GridSearchCV(estimator=sklearn.cluster.MeanShift(), param\_grid=param\_dict,

scoring=cv\_silhouette\_scorer, cv=cv, n\_jobs=-1)

gs.fit(df[cols\_of\_interest])

Cross-validation is a technique for evaluating ML models by training several ML models on subsets of the available input data and evaluating them on the complementary subset of the data. Now there will be only one fold:

Fitting 1 folds for each of 432 candidates, totalling 432 fits

The default is 5 cross validation ML model runs, which can be quite time intensive (if each run is 10 minutes for example.) You can just make cv=2. But this seems to work.

Q: How do I find learning rate in an optimizer using callback in Keras?

From [here](https://www.tensorflow.org/guide/keras/custom_callback):

    if not hasattr(self.model.optimizer, 'lr'):

      raise ValueError('Optimizer must have a "lr" attribute.')

    # Get the current learning rate from model's optimizer.

    lr = float(tf.keras.backend.get\_value(self.model.optimizer.lr))

How can I monitor progress using Tensorboard

Make sure installed.

From [here](https://www.tensorflow.org/tensorboard/get_started) :

Set up the code to create a tensorboard callback during the model fit.

# Tensorboard logging setup – empty directory if existsd

if(os.path.exists(config["model\_dir"]+"logs/fit/")):

shutil.rmtree(checkpoint\_dir)

os.makedirs(checkpoint\_dir)

log\_dir = config["model\_dir"]+"logs/fit/" + datetime.now().strftime("%Y%m%d-%H%M%S")

tensorboard\_callback = tf.keras.callbacks.TensorBoard(log\_dir=log\_dir, histogram\_freq=1)

. . .

rscv\_results = rscv.fit(x\_train, y\_train,

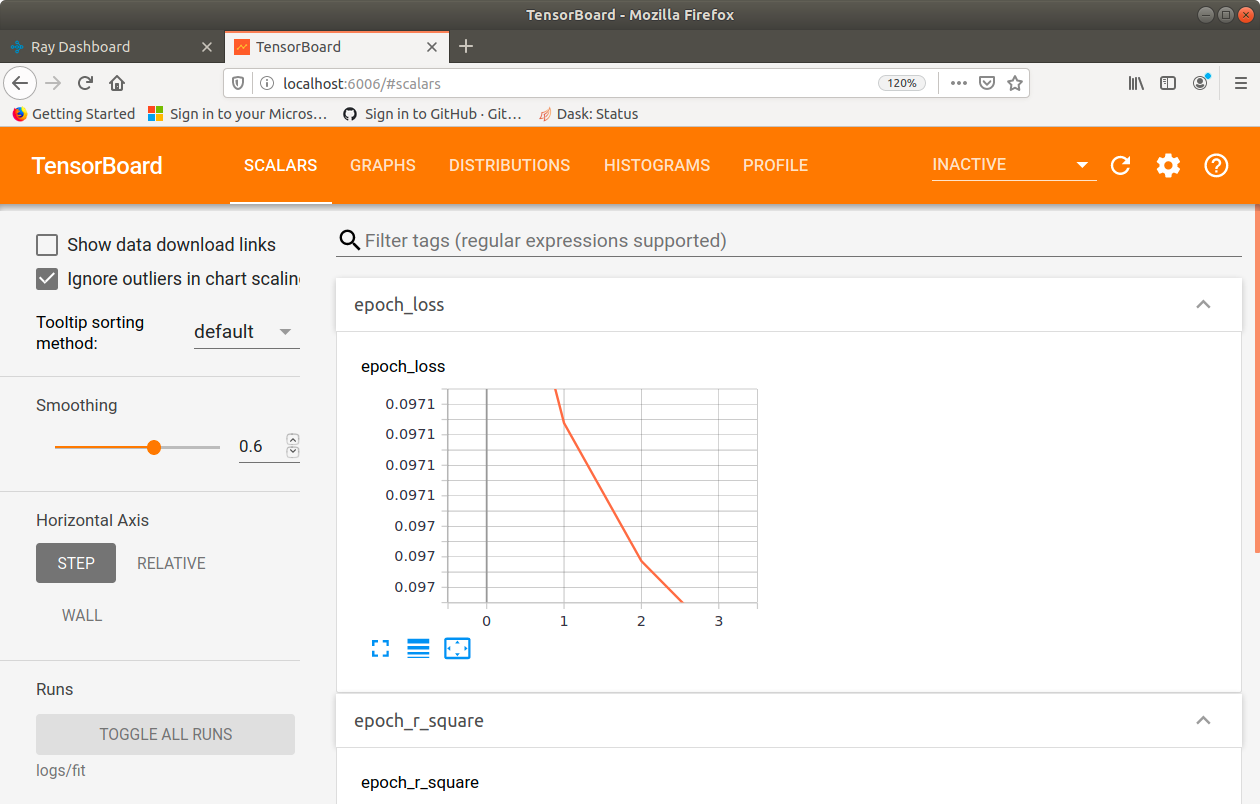
callbacks=[MLCallbacks(),tensorboard\_callback])

Then from a terminal command line start the tensorboard web server:

$ tensorboard --logdir logs/fit

Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind\_all TensorBoard 2.1.0 at <http://localhost:6006/> (Press CTRL+C to quit)

And bring up a web browser using the URL provided



Q: Are there any other ML regression functions that are useful:

Keras has the following loss functions:

* mean\_squared\_error , mean\_absolute\_error ,  mean\_squared\_logarithmic\_error , squared\_hinge , hinge , logcosh , categorical\_crossentropy , poisson , cosine\_proximity

The tutorial [here](https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0) states that the following loss functions are useful for regression ML:

* mean squared error, mean absolute error, huber loss, logcosh loss, quantile loss

You can wrap TensorFlow 's tf.losses.huber\_loss in a custom Keras loss function and then pass it to your model.

def get\_huber\_loss\_fn(\*\*huber\_loss\_kwargs):

def custom\_huber\_loss(y\_true, y\_pred):

return tf.losses.huber\_loss(y\_true, y\_pred, \*\*huber\_loss\_kwargs)

return custom\_huber\_loss

# Later...

model.compile(

loss=get\_huber\_loss\_fn(delta=0.1)

...

)

Quantile loss is explained [here](https://www.kdnuggets.com/2018/07/deep-quantile-regression.html).

Q: Find the Activation function in callback from Keras?

Most other hyperparameters are explicitly part of the model creation, however, the activation function is related to weights in the TensorFlow graph. You will need the keract Python package from Keras creator to decipher the activation functions for each layer in the DNN.

(py37) michalos@lightning:/usr/local/michalos/AI/Py37$ pip install keract Collecting keract

  Downloading keract-3.1.0-py3-none-any.whl (9.7 kB) Requirement already satisfied: numpy>=1.16.2 in /usr/local/anaconda3/envs/py37/lib/python3.7/site-packages (from keract)

(1.18.1)

Installing collected packages: keract

Successfully installed keract-3.1.0

Notes on running on Enki

<https://stackoverflow.com/questions/38601026/easy-way-to-use-parallel-options-of-scikit-learn-functions-on-hpc>