Design Space Exploration of Deep Neural Network in Resource Constrained Devices

Credits: (https://github.com/AmriHS/DNN-Inference-Optimization (https://github.com/AmriHS/DNN-Inference-Optimization)) by Yang Ren, Rui Xin, Hassan Alamri

This notebook demonstrates how to use our framework to produce the in the final report.

The environment set up as follow:

In [2]:

```
# imports and basic setup for SVR_prediction.py
from sklearn.datasets import make_regression
from sklearn.multioutput import MultiOutputRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
import numpy as np
import pandas as pd
from math import sqrt
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot
import matplotlib.pyplot as plt
from sklearn import cross_validation
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# imports and basic setup for baysian_opt.py
import paramiko
from matplotlib import pyplot as plt
import matplotlib.pyplot as plt
import gpflow
import gpflowopt
import numpy as np
import random
import time
import csv
from gpflowopt.acquisition import ExpectedImprovement
from random import randint

# imports and basic setup for regression_poly.py
from sklearn.datasets import make_regression
from sklearn.multioutput import MultiOutputRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
```

```
import numpy as np
import pandas as pd
from math import sqrt
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# imports and basic setup for run_bench_v2.py
import tensorflow as tf
import argparse
import keras
import json
import os
import cv2
import Keras_Resnet50 as res50
import subprocess
import time
import signal
```

Shell File Introduction

In our project, we has four shell files to change the GPU factors.

Use cpu_freq.sh file to change the cpu frequency

In [ ]:

```
num_cores=$1
cpu_freq=$2
cur_freq=$(cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq)

# Disable/enable CPU cores
# 3 is the total number of cores we are able to enable.
# We start from core 1 to num_cores requested to disable. Subsequently
, we enable cores that are not asked for.

for i in $(seq 1 1 $num_cores)
        do
                sudo bash -c 'echo 0 > /sys/devices/system/cpu/cpu['$i
']/online'
        done

num_cores=$((num_cores+1))
for i in $(seq $num_cores 1 3)
        do
                sudo bash -c 'echo 1 > /sys/devices/system/cpu/cpu['$i
']/online'
        done


#Change GPU Frequency
if [ ! -z $cpu_freq ];
then
        if [ $cpu_freq -gt $cur_freq ];
        then
                sudo bash -c 'echo '${cpu_freq}' > /sys/devices/system
/cpu/cpu0/cpufreq/scaling_max_freq'
                sudo bash -c 'echo '${cpu_freq}' > /sys/devices/system
/cpu/cpu0/cpufreq/scaling_min_freq'

        else
                sudo bash -c 'echo '${cpu_freq}' > /sys/devices/system
/cpu/cpu0/cpufreq/scaling_min_freq'
                sudo bash -c 'echo '${cpu_freq}' > /sys/devices/system
/cpu/cpu0/cpufreq/scaling_max_freq'
        fi
fi
```

Use emc_freq.sh file to change the emc frequency

In [ ]:

```
emc_freq=$1
cur_freq=$(cat /sys/kernel/debug/clk/override.emc/clk_rate)

#Change EMC Frequency
sudo bash -c 'echo '${emc_freq}' > /sys/kernel/debug/clk/override.emc/clk_update_rate'
sudo bash -c 'echo 1 > /sys/kernel/debug/clk/override.emc/clk_state'
```

Use gpu_freq.sh file to change the GPU frequency

In [ ]:

```
gpu_freq=$1
cur_freq=$(cat /sys/devices/57000000.gpu/devfreq/57000000.gpu/cur_freq)

#Change GPU Frequency
if [ $gpu_freq -gt $cur_freq ];
then
        sudo bash -c 'echo '${gpu_freq}' >  /sys/devices/57000000.gpu/devfreq/57000000.gpu/max_freq'
        sudo bash -c 'echo '${gpu_freq}' >  /sys/devices/57000000.gpu/devfreq/57000000.gpu/min_freq'
else
        sudo bash -c 'echo '${gpu_freq}' >  /sys/devices/57000000.gpu/devfreq/57000000.gpu/min_freq'
        sudo bash -c 'echo '${gpu_freq}' >  /sys/devices/57000000.gpu/devfreq/57000000.gpu/max_freq'
fi
```

Use script.sh file to set the configuration space

In [ ]:

```
# possible configuration space
$CPU_FREQ = $1
$CPU_DIS_CORES = $2
$GPU_FREQ = $3
$EMC_FREQ = $4

sh ./cpu_freq.sh $CPU_FREQ $CPU_DIS_CORES
sh ./gpu_freq.sh $GPU_FREQ
sh ./emc_freq.sh $EMC_FREQ

python run_benchmark.py --bsize 32 --all_growth 1 --mem_frac 0.25

# verify configuration
cur_gpu_freq=$(cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq)
cur_cpu_freq=$(cat /sys/devices/57000000.gpu/devfreq/57000000.gpu/cur_freq)
cur_emc_freq=$(cat /sys/kernel/debug/tegra_bwmgr/emc_rate)
dis_cpu_core_1=$(cat /sys/devices/system/cpu/cpu1/online)
dis_cpu_core_2=$(cat /sys/devices/system/cpu/cpu2/online)
dis_cpu_core_3=$(cat /sys/devices/system/cpu/cpu3/online)


echo "GPU Frequency: ${cur_gpu_freq}"
echo "CPU Frequency: ${cur_cpu_freq}"
echo "EMC Frequency: ${cur_emc_freq}"
echo "CPU 1 core Status: ${dis_cpu_core_1}"
echo "CPU 2 core Status: ${dis_cpu_core_2}"
echo "CPU 3 core Status: ${dis_cpu_core_3}"
#echo "$host, `date`, checkout,$Time_checkout" >> log.csv
```

Pre-Train Model Introduction

In this project, we use two pre-train model as the input model:

```
* Keras_Resnet50
* VGG 16
```

In [ ]:

```
# Keras_Resnet50.py
# An example using Keras Resnet50 pre-trained model to measure the inference time
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
```

```
from datetime import datetime
import time
import os
from apscheduler.schedulers.background import BackgroundScheduler
#import apscheduler.schedulers.blocking
import commands

from keras.applications.resnet50 import ResNet50
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input, decode_predictions
from timeit import default_timer as timer
from keras.datasets import cifar10
from multiprocessing import Process, Queue

import keras.backend.tensorflow_backend as ktf
import tensorflow as tf
import numpy as np
import cv2 #, os
import csv
import gc


os.environ["CUDA_VISIBLE_DEVICES"]="0"
import subprocess

import logging

#log = logging.getLogger('apscheduler.executors.default')
#log.setLevel(logging.INFO)  # DEBUG

#fmt = logging.Formatter('%(levelname)s:%(name)s:%(message)s')
#h = logging.StreamHandler()
#h.setFormatter(fmt)
#log.addHandler(h)


logger = logging.getLogger()  # this returns the root logger
logger.addHandler(logging.StreamHandler())
Time = []
power_cons = []

def tick():
    # Read current power consumptions
    input0 = open('/sys/devices/7000c400.i2c/i2c-1/1-0040/iio_device/in_power0_input', 'r')
    input1 = open('/sys/devices/7000c400.i2c/i2c-1/1-0040/iio_device/in_power1_input', 'r')
    input2 = open('/sys/devices/7000c400.i2c/i2c-1/1-0040/iio_device/in_power2_input', 'r')
    mod_power = input0.readline()
    gpu_power = input1.readline()
    cpu_power = input2.readline()
    power_cons.append([float(mod_power), float(gpu_power), float(cpu_p
ower)])
    CurrentTime = time.time()
    Time.append(int(CurrentTime))
    #print('Tick! The time is: %s' % datetime.now())

def RelationPlot(Time):
    #plt.plot(Time, PowerConsumption)
    #plt.xlabel('Time')
    #plt.ylabel('PowerConsumption')
    #plt.savefig("PowerConsumption_test_timer.jpg")
    len_power=len(power_cons)
    mod_power_sum = 0
    gpu_power_sum = 0
    cpu_power_sum = 0
    for i in range(len_power):
        #print("Current Power Consumption:"+repr(power_cons[i]))
        mod_power_sum+=power_cons[i][0]
        gpu_power_sum+=power_cons[i][1]
        cpu_power_sum+=power_cons[i][2]
    mod_power_sum/=len_power
    gpu_power_sum/=len_power
    cpu_power_sum/=len_power
    return [mod_power_sum, gpu_power_sum, cpu_power_sum]

def write_to_csv(data, filename):
    with open(filename,'w') as out:
        csv_out= csv.writer(out, lineterminator='\n')
        csv_out.writerow(['Class', 'Prob'])
        for row in data:
            csv_out.writerow(row[0][1:])

def resize(dataset):
    processed_data = []
    for i in range(len(dataset)):
        x = cv2.resize(dataset[i], (224,224))
        x = image.img_to_array(x)
        #x = np.expand_dims(x, axis=0)
        processed_data.append(preprocess_input(x))
    return processed_data


def make_predictions(dataset, batch_size, allow_growth, memory_frac):
    #print('getting into models!')
    # reset values of power consumption
    power_cons = []
    scheduler = BackgroundScheduler()
    #scheduler = apscheduler.schedulers.blocking.BackgroundScheduler('apscheduler.job_defaults.max_instances': '2')
    #print('BackgroundScheduler define')
    scheduler.add_job(tick, 'interval', seconds=0.5, misfire_grace_time=1)# execute every 0.5 second
    #print('job added!')
    config = tf.ConfigProto(log_device_placement=False, device_count = {'GPU' :1})
    if allow_growth:
        config.gpu_options.allow_growth = True
    else:
        config.gpu_options.per_process_gpu_memory_fraction = memory_frac

    session = tf.Session(config=config)
    ktf.set_session(session)

    model = ResNet50(weights='imagenet')
    # start time
    try:
        scheduler.start()# new seperate thread
        #print('Press Ctrl+{0} to exit'.format('Break' if os.name == 'nt' else 'C'))
        start = timer()
        preds = model.predict(np.array(dataset), batch_size=batch_size)
        # end time
        end = timer()
    except (KeyboardInterrupt, SystemExit):
        # Not strictly necessary if daemonic mode is enabled but should be done if possible
        scheduler.shutdown()
    scheduler.shutdown()
    #ktf.clear_session()
    session.close()
    del session
    gc.collect()
    power_cons = RelationPlot(Time)
    # calculate runtime
    runtime = end-start
    #print('Runtime: ' + "{0:.2f}".format(runtime) + 's')
    return preds, runtime, power_cons

def run_resnet50_benchmark(dataset, batch_size, all_growth=True, mem_frac=None):
    os.environ["CUDA_VISIBLE_DEVICES"]="0"
    preds, runtime, power_cons = make_predictions(dataset[:100], batch_size, all_growth, mem_frac)
    decoded =  decode_predictions(preds, top=1)
    write_to_csv(decoded, "Keras_result.csv")
    #data = [runtime, power_cons[0],power_cons[1], power_cons[2]]
    return runtime, power_cons
    #for i in range(len(data)):
    #   q.put(data[i])
```

Sampling Strategy

In our project, we use baysian optimization, the reson is:

- Black-box optimization
- Small number of function evaluations
- Exploit regions that yield good points
- And explore regions with high uncertainty
- With small number of evaluations, it builds an informative model
- Based on Gaussian Process (GP) ** Uses previously observed parameters to make an assumption about unobserved parameters.
- Acquisition Function used to intelligently suggest the next set of parameters

baysian_opt.py will execute the baysian optimization process, the file detail as follow:

In [ ]:

```
import paramiko
from matplotlib import pyplot as plt
import matplotlib.pyplot as plt
import gpflow
import gpflowopt
import numpy as np
import random
import time
import csv
from gpflowopt.acquisition import ExpectedImprovement
from random import randint
random.seed(24)

# host IP and credential
ip='10.173.131.120'
port=22
username='ubuntu'
password='ubuntu'

#Linux commands
file_dir = 'cd DNN_Inference;'
sudo_cmd = 'sudo -S '
command='python run_bench_v2.py --cpu_freq {0} --num_cores {1} --gpu_f
```

```
req {2} --emc_freq {3} --bsize {4}' \
    +' --all_growth {5} --mem_frac {6}'
command2='python DNN_Inference/ssh_ex.py'

# define discrete values of each input space
batch_size_array=[1, 8, 16, 32] # 16 & 32 is excluded for VGG16
all_growth_array=[True, False]
memory_frac_array=[0.15, 0.2, 0.25, 0.3, 0.33] # 0.15, 0.2,  is exclud
ed for VGG16 model
GPU_freq_array=[76800000,153600000, 230400000, 307200000, 384000000, 4
60800000, 537600000, 614400000,
                691200000, 768000000, 844800000, 921600000, 998400000,
537600000,998400000]
CPU_freq_array=[102000,204000,306000, 408000, 510000, 612000, 714000,
816000, 918000,1020000,1122000, 1224000,
                1326000,1224000,1428000,1555000,1632000, 1734000]

# 40800000 frequency excluded due to frequent memory issue
EMC_freq_array=[800000000,1065600000,1331200000,1600000000]
num_dis_core_array=[0,1,2,3]

# define space input, upper and lower bounds for each dimension
domain = gpflowopt.domain.ContinuousParameter('CPU_frequency', 102000,
1734000) + \
         gpflowopt.domain.ContinuousParameter('num_cores_dis', 0, 4) +
\
         gpflowopt.domain.ContinuousParameter('GPU_frequency', 7680000
0, 998400000) + \
         gpflowopt.domain.ContinuousParameter('EMC_frequency', 8000000
00, 1600000000)+ \
         gpflowopt.domain.ContinuousParameter('Batch_size', 1, 32) + \
         gpflowopt.domain.ContinuousParameter('Allow_growth', 0, 1) +
\
         gpflowopt.domain.ContinuousParameter('Memory_fraction', 0.15,
0.33)

is_sampling = True

# connect to linux server using ssh
ssh=paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(ip,port,username,password)

# write output to csv file
def write_to_csv(data, filename):
    with open(filename,'a') as out:
        csv_out= csv.writer(out, lineterminator='\n')
        #csv_out.writerow(['CPU Frequency', '# of enabled cores','GPU
Frequency','EMC Frequency','Batch Size', 'Mem Growth',
        #                  'Mem Fraction', 'Runtime', 'Model Cons','GP
U Cons', 'CPU Cons'])
        for row in data:
            csv_out.writerow(row)

# find closest discrete values to the continuous input
def find_closest(values):
    values[0] = min(CPU_freq_array, key=lambda x:abs(x-values[0]))
    values[1] = min(num_dis_core_array, key=lambda x:abs(x-values[1]))
    values[2] = min(GPU_freq_array, key=lambda x:abs(x-values[2]))
    values[3] = min(EMC_freq_array, key=lambda x:abs(x-values[3]))
    values[4] = min(batch_size_array, key=lambda x:abs(x-values[4]))
    values[5] = min(all_growth_array, key=lambda x:abs(x-values[5]))
    values[6] = min(memory_frac_array, key=lambda x:abs(x-values[6]))


def handle_output(output):
    if len(output) == 0:
        return [None, None, None, None]
    else:
        return [float(output[0]), float(output[1]), float(output[2]),
float(output[3])]

# Optimization multo-objective function
def objective_func(params):
    y1 = []
    y2 = []
    benchmark_data = []
    for i in range(len(params)):
        print ("Iteration:"+str(i))
        find_closest(params[i])
        print (params[i,1])
        cmd = command.format(int(params[i,0]), int(params[i,1]),int(pa
rams[i,2]),int(params[i,3]),int(params[i,4]),
                             int(params[i,5]),float(params[i,6]))
        print ("Command:"+repr(cmd))
        stdin,stdout,stderr=ssh.exec_command(file_dir+sudo_cmd+cmd)
        stdin.write("ubuntu\n")
        stdin.flush()
        outlines=stdout.readlines()
        response=''.join(outlines).split()
        response = handle_output(response)
        print (response)
        benchmark_data.append([params[i,0], params[i,1], params[i,2],
params[i,3], params[i,4], params[i,5],
                         params[i,6], response[0], response[1], respon
se[2], response[3]])
        y1.append([response[0]])
        y2.append([response[1]])
    write_to_csv(benchmark_data, "result.csv")
    return np.hstack((y1,y2))
```

```
def random_search():
    n_samples = 40
    design = gpflowopt.design.RandomDesign(n_samples, domain)
    X = design.generate()
    Y = objective_func(X)
    itemindex = np.where(Y[:n_samples]==None) #discard samples where it doesn't produce output
    Y = np.delete(Y, (itemindex[0]), axis=0)
    Y = np.array(Y, dtype=float)

    plt.scatter(Y[:,0], Y[:,1])
    plt.title('Random set')
    plt.xlabel('Inference Time')
    plt.ylabel('Power Consumption')
    plt.show()

    print (Y.shape[0])

    plt.plot(np.arange(0, Y.shape[0]),np.minimum.accumulate(Y[:,0]) ,'b',label='Inference Time')
    plt.ylabel('fmin')
    plt.xlabel('Number of evaluated points')
    plt.legend()
    plt.show()

    plt.plot(np.arange(0, Y.shape[0]),np.minimum.accumulate(Y[:,1]) ,'g',label='Power Consumption')
    plt.ylabel('fmin')
    plt.xlabel('Number of evaluated points')
    plt.legend()
    plt.show()

def baysian_opt():
    global is_sampling
    n_samples = 10
    design = gpflowopt.design.LatinHyperCube(n_samples, domain)
    X = design.generate()
    X = np.array(X)
    Y = objective_func(X)
    # discard samples where it doesn't produce output
    itemindex = np.where(Y[:n_samples]==None)
    Y = np.delete(Y, (itemindex[0]), axis=0)
    X = np.delete(X, (itemindex[0]), axis=0)
    Y = np.array(Y, dtype=float)
    n_samples = len(X)
    is_sampling = False
    # One model for each objective
    objective_models = [gpflow.gpr.GPR(X.copy(), Y[:,[i]].copy(), gpflow.kernels.Matern52(domain.size, ARD=True)) for i in range(Y.shape[1])
]
    for model in objective_models:
        model.likelihood.variance = 0.01

    hvpoi = gpflowopt.acquisition.HVProbabilityOfImprovement(objective
_models)
    acquisition_opt = gpflowopt.optim.StagedOptimizer([gpflowopt.optim
.MCOptimizer(domain, n_samples),
                                                       gpflowopt.optim
.SciPyOptimizer(domain)])

    # Then run the BayesianOptimizer for 40 iterations
    optimizer = gpflowopt.BayesianOptimizer(domain, hvpoi, optimizer=a
cquisition_opt, verbose=True)
    optimizer.optimize(objective_func, n_iter=30)

    pf, dom = gpflowopt.pareto.non_dominated_sort(hvpoi.data[1])

    plt.scatter(hvpoi.data[1][:,0], hvpoi.data[1][:,1], c=dom)
    plt.title('Pareto set')
    plt.xlabel('Inference Time')
    plt.ylabel('Power Consumption')
    plt.show()

    plt.plot(np.arange(0, hvpoi.data[0].shape[0]),np.minimum.accumulat
e(hvpoi.data[1][:,0]) ,'b',label='Inference Time')
    plt.ylabel('fmin')
    plt.xlabel('Number of evaluated points')
    plt.legend()
    plt.show()

    plt.plot(np.arange(0, hvpoi.data[0].shape[0]),np.minimum.accumulat
e(hvpoi.data[1][:,1]) ,'g',label='Power Consumption')
    plt.ylabel('fmin')
    plt.xlabel('Number of evaluated points')
    plt.legend()
    plt.show()
baysian_opt()
random_search()
```

Baysian Optimization Result

jupyter

jupyter

jupyter

jupyter

jupyter

SVR_prediction.py

In [ ]:

```
from sklearn.datasets import make_regression
from sklearn.multioutput import MultiOutputRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.pipeline import Pipeline
import numpy as np
import pandas as pd
from math import sqrt
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot
import matplotlib.pyplot as plt
from sklearn import cross_validation
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

def polynomial(data_x, data_y, split, init_degree=2, max_degree = 2):
    #data_x = np.array(data_x, dtype="float64")
    #data_y = np.array(data_y, dtype="float64")
    #data_y = np.round(data_y, decimals=1)

    X_train, Y_train = data_x[:split], data_y[:split]
    X_test, Y_test = data_x[split:], data_y[split:]

    degrees = np.arange(init_degree,max_degree+1)
    pred = []
    for i in range(len(degrees)):
        model = Pipeline([
            ('poly', PolynomialFeatures(degree=degrees[i])),
            ('linreg', LinearRegression(normalize=True))
        ])
        model.fit(X_train, Y_train)
        Y_train_pred = model.predict(X_train)
        Y_test_pred = model.predict(X_test)
        pred.append(Y_test_pred)

        intercept = model.named_steps['linreg'].intercept_[0]
        coef = model.named_steps['linreg'].coef_[0]
        features = model.named_steps['poly'].get_feature_names()
        assert(len(coef) == len(features))

        estimated_inf_f = '{:+.2f}'.format(intercept)

        for j in range(0, len(coef)-1):
            if float ('{:+.1f}'.format(coef[j]).replace('\U00002013',
'-')) != 0.0:
                estimated_inf_f += ' + {:+.1f} {}'.format(np.round(coe
f[j], decimals=2), features[j+1])

        print (estimated_inf_f)
        # plot function we want to learn
        rmse_infer = sqrt(mean_squared_error(Y_test_pred[:,0], Y_test[
:,0]))
        rmse_power = sqrt(mean_squared_error(Y_test_pred[:,1], Y_test[
:,1]))

        print('Test RMSE for Inference Time: %.3f' % rmse_infer)
        print('Test RMSE for Power Consumption: %.3f' % rmse_power)
    return pred

def test_multi_target_regression(data_x, data_y):
    n_train = int (len(data_x)*0.80)
    X_train, Y_train = data_x[:n_train], data_y[:n_train]
    X_test, Y_test = data_x[n_train:], data_y[n_train:]

    rgr = MultiOutputRegressor(GradientBoostingRegressor(random_state=
0))
    rgr.fit(X_train, Y_train)
    Y_train_pred = rgr.predict(X_train)
    Y_test_pred = rgr.predict(X_test)

    plot_result(Y_train[:,0],Y_test[:,0], Y_train_pred[:,0], Y_test_pr
ed[:,0])
    plot_result(Y_train[:,1],Y_test[:,1], Y_train_pred[:,1], Y_test_pr
ed[:,1])

    rmse_infer = sqrt(mean_squared_error(Y_test_pred[:,0], Y_test[:,0]
))
    rmse_power = sqrt(mean_squared_error(Y_test_pred[:,1], Y_test[:,1]
))
    rmse_train_infer = sqrt(mean_squared_error(Y_train_pred[:,0], Y_tr
ain[:,0]))
    rmse_train_power = sqrt(mean_squared_error(Y_train_pred[:,1], Y_tr
ain[:,1]))
```

```
    print('Train RMSE for Inference Time: %.3f' % rmse_train_infer)
    print('Train RMSE for Power Consumption: %.3f' % rmse_train_power)
    print('Test RMSE for Inference Time: %.3f' % rmse_infer)
    print('Test RMSE for Power Consumption: %.3f' % rmse_power)


def plot_min_iteration(bay_y, rand_y):
    assert (len(bay_y) == len(rand_y))
    plt.plot(np.arange(0, bay_y.shape[0]),np.minimum.accumulate(bay_y[:,0]) ,'darkslateblue',label='Baysian Opt Inference')
    plt.plot(np.arange(0, rand_y.shape[0]),np.minimum.accumulate(rand_y[:,0]) ,'darkseagreen',label='Random Inference')
    plt.ylabel('fmin')
    plt.xlabel('Number of evaluated points')
    plt.legend()
    plt.show()

    plt.plot(np.arange(0, bay_y.shape[0]),np.minimum.accumulate(bay_y[:,1]) ,'firebrick',label='Baysian Opt Power Consumption')
    plt.plot(np.arange(0, rand_y.shape[0]),np.minimum.accumulate(rand_y[:,1]) ,'royalblue',label='Random Power Consumption')
    plt.ylabel('fmin')
    plt.xlabel('Number of evaluated points')
    plt.legend()
    plt.show()

def plot_result(Y_train,Y_test, Y_train_pred, Y_test_pred):
    data_y = np.concatenate((Y_train, Y_test), axis=0)
    data_y = data_y.reshape((data_y.shape[0], 1))
    sequence_arr = np.arange(1,len(data_y)+1).reshape(len(data_y),1)

    pyplot.plot(sequence_arr,data_y, 'o', color='firebrick', label='ground truth')
    pyplot.plot(sequence_arr[:len(Y_train)],Y_train_pred,'cornflowerblue',label='pred-train')
    pyplot.plot(sequence_arr[len(Y_train):],Y_test_pred,'darkslategray',label='pred-test')
    pyplot.legend(loc='best')
    pyplot.show()

dir_path = 'C:/Users/hcaro/Google Drive/Fall 2018/MLS/Project/DNN_Inference/Experiment Result'
bays_dataset = pd.read_csv(dir_path+'/Bays_VDD_40.csv')
rand_dataset = pd.read_csv(dir_path+'/Random_vgg_40.csv') #

# baysian Optimization Dataset
bays_dataset = bays_dataset.values[:,:-2]
bays_dataset = bays_dataset.astype('float32')
```

```
# random Optimization Dataset
rand_dataset = rand_dataset.values[:,:-2]
rand_dataset = rand_dataset.astype('float32')


# exclude CPU & GPU consumption
bays_data_x, bays_data_y = bays_dataset[:,:-2], bays_dataset[:,-2:]
rand_data_x, rand_data_y = rand_dataset[:,:-2], rand_dataset[:,-2:]

test_multi_target_regression(bays_data_x, bays_data_y)
#print ("---------------------------------------------")
test_multi_target_regression(rand_data_x, rand_data_y)


#print ("##########################")

sequence_arr = np.arange(1,len(bays_data_x)+1).reshape(len(bays_data_x),1)
n_train = int (len(bays_data_x)*0.8)
max_degree = 3
degrees = np.arange(2,max_degree+1)
pyplot.plot(sequence_arr[n_train:],bays_data_y[n_train:,0],'o', color='navy', linewidth="2", marker='o', label='ground truth')
pyplot.plot(sequence_arr[n_train:],bays_data_y[n_train:,0], color='cornflowerblue', linewidth="2", label='ground truth')
pred_list = polynomial(bays_data_x, bays_data_y, n_train, init_degree=3, max_degree=max_degree)
colors = ["navy", "brown", "teal", "darkslategray"]
for i in range(len(pred_list)):
    pyplot.plot(sequence_arr[n_train:],pred_list[i][:,0],color=colors[i],linewidth=2,label='Degree %d' %degrees[i])
pyplot.legend(loc='best')
pyplot.show()

pyplot.plot(sequence_arr[n_train:],bays_data_y[n_train:,1],'o', color='navy', linewidth="2", marker='o', label='ground truth')
pyplot.plot(sequence_arr[n_train:],bays_data_y[n_train:,1], color='cornflowerblue', linewidth="2", label='ground truth')
colors = ["navy", "brown", "teal", "darkslategray"]
for i in range(len(pred_list)):
    pyplot.plot(sequence_arr[n_train:],pred_list[i][:,1],color=colors[i],linewidth=2,label='Degree %d' %degrees[i])
pyplot.legend(loc='best')
pyplot.show()


plot_min_iteration(bays_data_y, rand_data_y)
```

regression_poly.py

In [ ]:

```
from sklearn.datasets import make_regression
from sklearn.multioutput import MultiOutputRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVR
import numpy as np
import pandas as pd
from math import sqrt
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression


def test_multi_target_regression_poly(data_x, data_y):
    print(len(data_x))
    n_train = int (len(data_x)*0.80)
    X_train, Y_train = data_x[:n_train], data_y[:n_train]
    X_test, Y_test = data_x[n_train:], data_y[n_train:]
    references = np.zeros_like(Y_test)
    for n in range(2):
        rgr = GradientBoostingRegressor(random_state=0)
        rgr.fit(X_train, Y_train[:, n])
        references[:,n] = rgr.predict(X_test)
    rgr = MultiOutputRegressor(GradientBoostingRegressor(random_state=
0))
    rgr.fit(X_train, Y_train)
    "" """"

    # Create matrix and vectors
    X = [[0.44, 0.68], [0.99, 0.23]]
    y = [109.85, 155.72]
    X_test = [0.49, 0.18]
    "" """[1]
    # PolynomialFeatures (prepreprocessing)
    poly = PolynomialFeatures(degree=2)
    X_ = poly.fit_transform(X_train)
    X_test_ = poly.fit_transform(X_test)
    # Instantiate
    lg = LinearRegression()
    # Fit
    lg.fit(X_, Y_train)
    # Obtain coefficients
    lg.coef_
    # Predict
    Y_train_pred = lg.predict(X_train)
    Y_test_pred = lg.predict(X_test_)
    "" """"
        Y_train_pred = rgr.predict(X_)
        Y_test_pred = rgr.predict(X_test)
    "" """[2]

    sequence_arr = np.arange(1,len(data_x)+1).reshape(len(data_x),1)
    print(len(Y_test_pred))
    print(len(sequence_arr[n_train:]))

    plot_result(sequence_arr[:n_train], Y_train[:,0], sequence_arr[n_t
rain:],Y_test[:,0], Y_train_pred[:,0], Y_test_pred[:,0])
    plot_result(sequence_arr[:n_train], Y_train[:,1], sequence_arr[n_t
rain:],Y_test[:,1], Y_train_pred[:,1], Y_test_pred[:,1])

    rmse_infer = sqrt(mean_squared_error(Y_test_pred[:,0], Y_test[:,0]
))
    rmse_power = sqrt(mean_squared_error(Y_test_pred[:,1], Y_test[:,1]
))
    rmse_train_infer = sqrt(mean_squared_error(Y_train_pred[:,0], Y_tr
ain[:,0]))
    rmse_train_power = sqrt(mean_squared_error(Y_train_pred[:,1], Y_tr
ain[:,1]))
    print('Train RMSE for Inference Time: %.3f' % rmse_train_infer)
    print('Train RMSE for Power Consumption: %.3f' % rmse_train_power)
    print('Test RMSE for Inference Time: %.3f' % rmse_infer)
    print('Test RMSE for Power Consumption: %.3f' % rmse_power)


def test_multi_target_regression(data_x, data_y):
    print(len(data_x))
    n_train = int (len(data_x)*0.80)
    X_train, Y_train = data_x[:n_train], data_y[:n_train]
    X_test, Y_test = data_x[n_train:], data_y[n_train:]
    references = np.zeros_like(Y_test)

    for n in range(2):
        rgr = GradientBoostingRegressor(random_state=0)
        rgr.fit(X_train, Y_train[:, n])
        references[:,n] = rgr.predict(X_test)
    rgr = MultiOutputRegressor(GradientBoostingRegressor(random_state=
0))
    rgr.fit(X_train, Y_train)

    Y_train_pred = rgr.predict(X_train)
    Y_test_pred = rgr.predict(X_test)
    sequence_arr = np.arange(1,len(data_x)+1).reshape(len(data_x),1)
    print(len(Y_test_pred))
    print(len(sequence_arr[n_train:]))

    plot_result(sequence_arr[:n_train], Y_train[:,0], sequence_arr[n_t
rain:],Y_test[:,0], Y_train_pred[:,0], Y_test_pred[:,0])
    plot_result(sequence_arr[:n_train], Y_train[:,1], sequence_arr[n_t
rain:],Y_test[:,1], Y_train_pred[:,1], Y_test_pred[:,1])

    rmse_infer = sqrt(mean_squared_error(Y_test_pred[:,0], Y_test[:,0]
))
    rmse_power = sqrt(mean_squared_error(Y_test_pred[:,1], Y_test[:,1]
))
    rmse_train_infer = sqrt(mean_squared_error(Y_train_pred[:,0], Y_tr
ain[:,0]))
    rmse_train_power = sqrt(mean_squared_error(Y_train_pred[:,1], Y_tr
ain[:,1]))
    print('Train RMSE for Inference Time: %.3f' % rmse_train_infer)
    print('Train RMSE for Power Consumption: %.3f' % rmse_train_power)
    print('Test RMSE for Inference Time: %.3f' % rmse_infer)
    print('Test RMSE for Power Consumption: %.3f' % rmse_power)


def plot_result(X_train, Y_train, X_test,Y_test, Y_train_pred, Y_test_
pred):
    pyplot.plot(X_train,Y_train_pred,'b',label='pred-train')
    pyplot.plot(X_test,Y_test_pred,'g',label='pred-test')
    pyplot.plot(X_train,Y_train,'rx',label='ground truth')
    pyplot.plot(X_test,Y_test,'rx')
    pyplot.legend(loc='best')
    pyplot.show()

dir_path = '/home/rick/Project_DNN/Experiment Result'
dataset = pd.read_csv(dir_path+'/23Nov_30Samples.csv')
dataset = dataset.values[:,:-2]
dataset = dataset.astype('float32')
data_x, data_y = dataset[:,:-2], dataset[:,-2:]
test_multi_target_regression_poly(data_x, data_y)
#rmse = sqrt(mean_squared_error(predict, test_y))
#print('Test RMSE: %.3f' % rmse)
```

script.sh

In [ ]:

```
# possible configuration space
$CPU_FREQ = $1
$CPU_DIS_CORES = $2
$GPU_FREQ = $3
$EMC_FREQ = $4

sh ./cpu_freq.sh $CPU_FREQ $CPU_DIS_CORES
sh ./gpu_freq.sh $GPU_FREQ
sh ./emc_freq.sh $EMC_FREQ

python run_benchmark.py --bsize 32 --all_growth 1 --mem_frac 0.25

# verify configuration
cur_gpu_freq=$(cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq)
cur_cpu_freq=$(cat /sys/devices/57000000.gpu/devfreq/57000000.gpu/cur_freq)
cur_emc_freq=$(cat /sys/kernel/debug/tegra_bwmgr/emc_rate)
dis_cpu_core_1=$(cat /sys/devices/system/cpu/cpu1/online)
dis_cpu_core_2=$(cat /sys/devices/system/cpu/cpu2/online)
dis_cpu_core_3=$(cat /sys/devices/system/cpu/cpu3/online)


echo "GPU Frequency: ${cur_gpu_freq}"
echo "CPU Frequency: ${cur_cpu_freq}"
echo "EMC Frequency: ${cur_emc_freq}"
echo "CPU 1 core Status: ${dis_cpu_core_1}"
echo "CPU 2 core Status: ${dis_cpu_core_2}"
echo "CPU 3 core Status: ${dis_cpu_core_3}"
#echo "$host, `date`, checkout,$Time_checkout" >> log.csv
```

Readme File in our repo will provide more running information.