LURA Project: Enhancing the Machine Learning-based Detection of Long-lasting Continuous Gravitational Waves

This is the Google Colab notebook of the LURA project with GitHUb repo here: https://github.com/dhanfort/LaSPACE_LURA2324_DetectContinuousGW.git.

- NASA Primary Grant Number: 80NSSC20M0110
- LSU/LaSPACE Sub-award Number: GR00013285
- Project Period: 15-Aug-23 to 14-Aug-24
- Mentored LURA Student: Lorin Bernard
- Faculty PI: Dr. Dhan Lord B. Fortela
- LaSPACE: https://laspace.lsu.edu//
- Louisiana NASA EPSCoR: https://lanasaepscor.lsu.edu/

Generation of Continupus GW data

- Using LALSuite and PyFstate. Installing PyFstate will also install LALSuite
- Run the codes below. Just use the basic Python runetime of Google Colab because GPU cannot accelarae PyFstate and LALSuite

```
# need to install PyFstat becuase Colab does not have it in its repo

!pip install pyfstat
```

```
import os, sys
os.environ["DISPLAY"] = "1.0"

import h5py
import argparse

import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import pandas as pd

import pyfstat
logger = pyfstat.set_up_logger(outdir="pyfstat_log", label="1_generating_signals", log_level="WARNING")


if __name__ == "__main__":

    # Read arguments:
    msg = "Generate dataset containing continuous gravitational wave data.\n"
    msg += "Example usage:\n"
    msg += "> python generate_data.py --sensitivity 10.0 --num_signals 1000"

    #sensitivity
    sensitivity = 5
    #num_signals = args.num_signals
    num_signals = 50


    out_dir = "./data/"
    dataset_name = f"data_sensitivity_{int(sensitivity)}"

    print("============================================")
    print(f"Generating dataset {dataset_name} with sensitivity {sensitivity} and {num_signals} files.\n")

    if not os.path.isdir(out_dir):
        os.mkdir(out_dir)

    if not os.path.isdir(os.path.join(out_dir, dataset_name)):
```

```
    os.mkdir(os.path.join(out_dir, dataset_name))


# Parameters for Pyfstat:

# These parameters describe background noise and data format
writer_kwargs = {
    "label": "single_detector_gaussian_noise",
    "outdir": "Generated_data",
    "tstart": 1238166018,  # Starting time of the observation [GPS time]
    "duration": 120 * 86400,  # Duration [seconds]
    "detectors": "H1,L1",  # Detector to simulate, in this case LIGO Hanford
    "Band": 0.18,  # Frequency band-width around F0 [Hz]
    "sqrtSX": 1e-15,  # Single-sided Amplitude Spectral Density of the noise
    "Tsft": 1800,  # Fourier transform time duration
    "SFTWindowType": "tukey",  # Window function to compute short Fourier transforms
    "SFTWindowParam": 0.01,  # Parameter associated to the window function

}


# Sample signal parameters from a specific population:
# Vary these parameters to generate varied data for CNN training

signal_parameters_generator = pyfstat.AllSkyInjectionParametersGenerator(
    priors={
        "tref": writer_kwargs["tstart"],
        "F0": {"stats.uniform": {"loc": 390, "scale": 50}}, # Central frequency of the band to be generated [Hz]
        "F1": {"stats.uniform": {"loc": -1e-10, "scale": 2e-10}},
        "F2": 0,
        "Alpha": 2.3,
        "Delta": 1.8,
        "psi": 0,
        "phi": 0,
        "cosi": 0.3,
        "h0": writer_kwargs["sqrtSX"]/sensitivity,
        **pyfstat.injection_parameters.isotropic_amplitude_distribution,
    },
)

columns = ['id', 'target', 'SNR', 'sqrtSX', 'F0', 'F1', 'h0', 'alpha', 'delta', 'psi', 'phi', 'cosi']
df = pd.DataFrame(columns=columns, index=None)

print(columns[:7])

i_signal = 0

while i_signal < num_signals:

    file_id = f"{i_signal:05}"

    # Draw signal parameters.
    params = signal_parameters_generator.draw()
    writer_kwargs["outdir"] = f"Generated_SFT_data_tmp/Signal{i_signal}"
    writer_kwargs["label"] = f"Signal{i_signal}"

    target = stats.bernoulli(0.5).rvs()  # 0 or 1
    params["h0"] *= target               # set h0 randomly to zero to generate only noise

    writer = pyfstat.Writer(**writer_kwargs, **params)
    try:
        writer.make_data()
    except:
        print(f"couldn't write {file_id}. Trying again with new set of parameters")
        continue  # do not increment iterator

    # SNR can be compute from a set of SFTs for a specific set
    # of parameters as follows:
    snr_ = pyfstat.SignalToNoiseRatio.from_sfts(F0=writer.F0, sftfilepath=writer.sftfilepath)

    squared_snr = snr_.compute_snr2(
        Alpha=writer.Alpha,
        Delta=writer.Delta,
```

```
            psi=writer.psi,
            phi=writer.phi,
            h0=writer.h0,
            cosi=writer.cosi
        )

        SNR = np.sqrt(squared_snr)

        # Data can be read as a numpy array using PyFstat
        frequency, timestamps, amplitudes = pyfstat.utils.get_sft_as_arrays(writer.sftfilepath)

    #     columns = ['file_id', 'target', 'SNR', 'sqrtSX', 'F0', 'F1', 'h0', 'alpha', 'delta', 'psi', 'phi', 'cosi']
        vals = [file_id, target, SNR, writer.sqrtSX, writer.F0, writer.F1, writer.h0, writer.Alpha, writer.Delta, writer.psi, writer
        df.loc[i_signal] = vals
        print(vals[:7])

        # save hdf5 files:
        out_file = os.path.join(out_dir, dataset_name, file_id+".hdf5")
        with h5py.File(out_file, "w") as f:

            H1 = f.create_group(file_id+'/H1')
            L1 = f.create_group(file_id+'/L1')

            f.create_dataset(file_id+'/frequency_Hz', data=frequency)

            H1.create_dataset('SFTs', data=amplitudes["H1"])
            H1.create_dataset('timestamps_GPS', data=timestamps["H1"])

            L1.create_dataset('SFTs', data=amplitudes["L1"])
            L1.create_dataset('timestamps_GPS', data=timestamps["L1"])

        i_signal += 1


    # save labels as csv file:
    df.to_csv(out_dir+dataset_name+"_labels.csv", index=False)

    print()
    print(f"Data saved to {os.path.join(out_dir, dataset_name)}")

    print()
    print("Deleting temporary SFT data in Generated_SFT_data_tmp/")
    os.system("rm -r Generated_SFT_data_tmp/")
```

```
# To downlowd dataset, first zip the "data" folder
!zip -r /content/out_file.zip /content/data

# Then download the zipped "data" folder now as "out_file.zip"
from google.colab import files
files.download("/content/out_file.zip")
```

CNN Model Training

- Assumption: You want to use Pytorch from Google Colab so it is best to upload your Continuous GW data to Google Drive and access your Drive folder from Colab or mount your Drive folder to Colab
- Make sure to upload all download zipped file to Google Drive
- Mount the Drive folder using the code below. An series of authorization and authentication dialogue boxes will prompt you to answer. Complete these.
- Then make sure to copy the link to the specific folder and paste them the the pertinent code lines below requiring access to your data in Drive.

```
#from google.colab import drive
#drive.mount("/content/drive")
```

```
!pip install timm
```

```
import os, sys
import copy
import h5py
import argparse

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
import sklearn.metrics

import torch
from torch import nn
from torch.nn import functional as F
from torchvision.transforms import ToTensor
from torch.utils.data import DataLoader

import utils # make sure to uplaod the 'utils' folder from the GitHub repo of our LURA project
import utils.ML, utils.models
from utils.data import G2NetDataset

if __name__ == "__main__":

    # Read arguments:
    msg = "Train a model using dataset present in data/\n"
    msg += "Example usage:\n"
    msg += "> python train_model.py --model_name CNN --num_epochs 15 --batch_size 32 --device mps"
    #parser = argparse.ArgumentParser(description=msg, formatter_class=argparse.RawTextHelpFormatter)

    #parser.add_argument('--model_name', required=True)
    #parser.add_argument('--num_epochs', default='10', type=int)
    #parser.add_argument('--batch_size', default='32', type=int)
    #parser.add_argument('--device', default='mps')
    #parser.add_argument('--num_workers', default='1', type=int)

    #args = parser.parse_args()

    #model_name = args.model_name
    model_name = "CNN"
    #num_epochs = args.num_epochs
    num_epochs = 15
    #batch_size = args.batch_size
    batch_size = 32
    #device = args.device
    device =  "cpu" # 'cuda'
    #num_workers = args.num_workers
    num_workers = 1

    # Load data:
    data_path = '/content/drive/MyDrive/Colab Notebooks/LURA_2324/data_all/' # use your own Drive path

    # List all datasets in data_all/:
    #dataset_list = [d for d in os.listdir(data_path) if os.path.isdir('data_all/'+d)]

    # or manually specify which data set in your Drive folder you want to use:
    dataset_list = ["data_sensitivity_5","data_sensitivity_6",
                    "data_sensitivity_61",
                    "data_sensitivity_7","data_sensitivity_8",
                    "data_sensitivity_9","data_sensitivity_95",
                    "data_sensitivity_10","data_sensitivity_12",
                    "data_sensitivity_13","data_sensitivity_14",
                    "data_sensitivity_15"]

    print(f"datasets found for train/validation/test:\n{dataset_list}")

    normalize = True  # normalize all spectrograms to zero-mean unit variance
    augment = True    # data augmentation

    data = torch.utils.data.ConcatDataset(
        [G2NetDataset(data_path, dataset_name, normalize=normalize, augment=augment) for dataset_name in dataset_list]
```

```
    )

train_val_test_split = [0.6, 0.2, 0.2] # you can vary this default we used in our LURA work

# split training data into train and validation sets:
train_split, val_split, test_split = torch.utils.data.random_split(data, train_val_test_split, generator=torch.Generator().manua

num_data = len(data)
num_train, num_val, num_test = len(train_split), len(val_split), len(test_split)

print(f"\nDataset size: {num_data}")
print(f"train/val/test split: {train_val_test_split[0]}/{train_val_test_split[1]}/{train_val_test_split[2]}")
print(f"Number of train/val/test samples: {num_train}/{num_val}/{num_test}")

# Create data loaders.
train_dataloader = DataLoader(train_split, batch_size=batch_size, shuffle=True, num_workers=num_workers)
val_dataloader = DataLoader(val_split, batch_size=batch_size, shuffle=False, num_workers=num_workers)
test_dataloader = DataLoader(test_split, batch_size=batch_size, shuffle=False, num_workers=num_workers)

# Load model:
if model_name == "CNN":
    model = utils.models.CNN().to(device)
elif model_name == "CNN_2":
    model = utils.models.CNN_v2().to(device)
elif model_name == "EfficientNet":
    model = utils.models.EfficientNet(freeze_blocks=False).to(device)
else:
    raise NotImplementedError(f"{model_name} not implemented. Choose among 'CNN', 'CNN_2', or 'EfficientNet'.")
print(f"\nTraining model {model_name}:")

# Freeze moving average parameters:
model.moving_average.weight.requires_grad = False

optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
loss_fn = nn.BCEWithLogitsLoss()

train_loss_log, val_loss_log = utils.ML.train_model(model,
    train_dataloader,
    val_dataloader,
    optimizer,
    loss_fn,
    num_epochs,
    save_model=True,
    model_name=model_name,
    device=device,
    verbose=True)

# Evaluate model:

# Load the model with best validation AUC:
model = torch.jit.load('saved_models/'+model_name+'.pt', map_location='cpu') # make sure to creat the "saved_models" folder in "
model.to(device)

test_loss, test_acc, test_auc = utils.ML.evaluate_model(model, test_dataloader, loss_fn, device=device)
print(f"Test loss: {test_loss:.3f}, test accuracy: {test_acc:.3f}, test AUC: {test_auc:.3f}")
```