Instructions for benchmark reproduction

Interpolating neural network: A novel unification of machine learning and interpolation theory

1. Open-source repository

The source codes for INN can be found in https://github.com/hachanook/pyinn

A python project of INN, "pyinn", can be found in https://pypi.org/project/pyinn/

2. PyINN installation

General guidance for PyINN installation can be found in https://github.com/hachanook/pyinn

This document will guide the simplest way to reproduce the benchmarks presented in the article.

Clone the repository

```
$ git clone https://github.com/hachanook/pyinn.git
$ cd pyinn
```

Create a conda environment

```
$ conda clean --all # [optional] to clear cache files in the base conda environment
$ conda env create -f environment.yaml
$ conda activate pyinn-env
```

Install JAX

See jax installation instructions. Depending on your hardware, you may install the CPU or GPU version of JAX. Both will work, while GPU version usually gives better performance.

- For CPU only (Linux/macOS/Windows), one can simply install JAX using:

```
$ pip install -U jax
```

- For GPU (NVIDIA, CUDA 12)

```
$ pip install -U "jax[cuda12]"
```

- For TPU (Google Cloud TPU VM)

```
$ pip install -U "jax[tpu]" -f https://storage.googleapis.com/jax-releases/libtpu_releases.html
```

- For TPU (Google Cloud TPU VM)

Install Optax – optimization library of JAX

```
$ pip install optax
```

Install PyINN

```
$ pip install pyinn
```

3. Benchmarking INN trainer

All INN benchmarks are standalone. Training data will be generated once the job is submitted. General running procedure of INN trainer follows:

1) Modify setting file: /pyinn/pyinn/settings.yaml
2) Modify configuration file: /pyinn/config/<data_name>.yaml
3) Run main.py

```
$ python ./pyinn/main.py
```

3.1. Manuscript Section 2.5: Training 10-input 5-output physical function

To try out different hyperparameters, a user may change the highlighted lines below.

1) Modify setting file: /pyinn/pyinn/settings.yaml

```
GPU:
  gpu_idx: 0 # specify gpu index if there are mutiple GPUs. If not, leave it as it is.

PROBLEM:
  run_type: "regression"
  TD_type: "CP"
  interp_method: "nonlinear" # set “nonlinear” for INN Q=2 and “MLP” for MLP

DATA:
  data_name : '10D_5D_physics' # physcial equation
```

2) Modify configuration file: /pyinn/config/10D_5D_physics.yaml

```
MODEL_PARAM:

  ## INN linear
  nmode : 14 # number of modes
  nelem : 10 # number of segments

  ## INN nonlinear
  s_patch : 2
  alpha_dil: 20
  p_order: 2
  radial_basis : "cubicSpline" # Activataion functions
  INNactivation : 'polynomial'
```

```
  ## MLP
  nlayers : 3 # number of hidden layers
  nneurons : 100 # number of neurons per layer
  activation : "sigmoid"

DATA_PARAM:

  input_col : [0,1,2,3,4,5,6,7,8,9]
  output_col : [10,11,12,13,14]

  bool_data_generation : True
  data_size: 100_000
  split_ratio: [0.8,0.2]

  bool_normalize: True
  bool_shuffle : True

TRAIN_PARAM:
  num_epochs_INN : 1000
  num_epochs_MLP : 1000
  batch_size : 128 # 128
  learning_rate : 1e-3
  bool_train_acc : False

  validation_period : 10
  bool_denormalize: False
  error_type : "mse"
  patience : 10
  stopping_loss_train: 4e-4


PLOT:
  bool_plot: False
  plot_in_axis: [3,4] # plot input axis
  plot_out_axis: [0] # plot output axis
```

3) Run main.py

```
$ python ./pyinn/main.py
```

3.2. SI Section 2.2: Adaptive INN activation function

To try out different hyperparameters, a user may change the highlighted lines below. Figure S4(b) uses the '1D_1D_sine' data whereas Figure S4(c) uses the '1D_1D_exp' data

1) Modify setting file: /pyinn/pyinn/settings.yaml

```
GPU:
  gpu_idx: 0 # specify gpu index if there are mutiple GPUs. If not, leave it as it is.

PROBLEM:
  run_type: "regression"
  TD_type: "CP"
  interp_method: "nonlinear"

DATA:
  data_name : '1D_1D_sine'
  # data_name: '1D_1D_exp'
```

2) Modify configuration file: /pyinn/config/1D_1D_sine.yaml

```
MODEL_PARAM:

  ## INN linear
  nmode : 1
  nelem : 5

  ## INN nonlinear
  s_patch : 2
  alpha_dil: 20
  p_order: 2
  radial_basis : "cubicSpline" # Activataion functions
  INNactivation : 'polynomial'
  # INNactivation : 'sinusoidal'
  # INNactivation : 'exponential'
  # INNactivation : 'sigmoid'
  # INNactivation : 'tanh'
  # INNactivation : 'gelu'

  ## MLP
  nlayers : 2
  nneurons : 10
  # activation : "relu"
  activation : "sigmoid"
```

```
DATA_PARAM:

  input_col : [0]
  output_col : [1]

  bool_data_generation : True # data is already stored and splitted
  data_size: 10_000
  split_ratio: [0.7,0.15,0.15]

  bool_normalize: False
  bool_shuffle : True

TRAIN_PARAM:
  num_epochs_INN : 100
  num_epochs_MLP : 200
  batch_size : 128
  learning_rate : 1e-3
  bool_train_acc : True

  validation_period : 1
  bool_denormalize:
  error_type : "rmse" # or mse
  patience : 3

PLOT:
  bool_plot: True
  plot_in_axis: [0] # plot input axis
  plot_out_axis: [0] # plot output axis
```

3) Run main.py

```
$ python ./pyinn/main.py
```

- **Adaptive activation function described in Figure S4(e)**

This benchmark is a special case where we optimize the activation function at the same time. This feature has not been implemented in the pyinn library. However, we can reproduce this benchmark at the “adaptive_activation” branch of the github repo. You can change the branch from “main” to “adaptive_activation” by this command:

```
$ git checkout adaptive_activation
```

Then, run main.py with the default setup:

```
$ python ./pyinn/main.py
```

The code will then print out the trainable hyperparameters $\boldsymbol{\Psi} = [\psi_1, \psi_2, \psi_3]$.

3.3. SI Section 2.3: One-input two-outputs functional relationship

To try out different hyperparameters, a user may change the highlighted lines below.

4) Modify setting file: /pyinn/pyinn/settings.yaml

```
GPU:
  gpu_idx: 0 # specify gpu index if there are mutiple GPUs. If not, leave it as it is.

PROBLEM:
  run_type: "regression"
  TD_type: "CP"
  interp_method: "nonlinear"
  # set “linear” for INN Q=1, set “nonlinear” for INN Q=2, and “MLP” for MLP

DATA:
  data_name : '1D_2D_sine_exp'
```

5) Modify configuration file: /pyinn/config/1D_2D_sine_exp.yaml

```
MODEL_PARAM:

  ## INN linear
  nmode : 1
  nelem : 5 # nnode = nelem + 1

  ## INN nonlinear
  s_patch : 2
  alpha_dil: 20
  p_order: 2
  radial_basis : "cubicSpline" # Activataion functions
  INNactivation : 'polynomial'

  ## MLP
  nlayers : 3
  nneurons : 21
  activation : "sigmoid"

DATA_PARAM:

  input_col : [0]
  output_col : [1,2]

  bool_data_generation : True
  data_size: 10_000
  split_ratio: [0.7,0.15,0.15]
```

```
  bool_normalize: False
  bool_shuffle : True


TRAIN_PARAM:
  num_epochs_INN : 100
  num_epochs_MLP : 300
  batch_size : 128
  learning_rate : 1e-3
  bool_train_acc : True

  validation_period : 1
  bool_denormalize: False
  error_type : "mse"
  patience : 10


PLOT:
  bool_plot: True
  plot_in_axis: [0] # plot input axis
  plot_out_axis: [0,1] # plot output axis
```

6) Run main.py

```
$ python ./pyinn/main.py
```

3.4. SI Section 2.5: Spiral classification

To try out different hyperparameters, a user may change the highlighted lines below.

1) Modify setting file: /pyinn/pyinn/settings.yaml

```
GPU:
  gpu_idx: 0 # specify gpu index if there are mutiple GPUs. If not, leave it as it is.

PROBLEM:
  run_type: "classification"
  TD_type: "CP"
  interp_method: "nonlinear"
  # set “linear” for INN Q=1, set “nonlinear” for INN Q=2, and “MLP” for MLP

DATA:
  data_name : 'spiral'
```

2) Modify configuration file: /pyinn/config/spiral.yaml

```
MODEL_PARAM:

  ## INN linear
  nmode : 10
  nelem : 20 # nnode = nelem + 1

  ## INN nonlinear
  s_patch : 2
  alpha_dil: 20
  p_order: 2
  radial_basis : "cubicSpline" # Activataion functions
  INNactivation : 'polynomial'

  ## MLP
  nlayers : 4
  nneurons : 50
  activation : "sigmoid"

DATA_PARAM:

  input_col : [0,1]
  output_col : [2]
  nclass : 2

  bool_data_generation : True
  split_ratio : [0.7,0.15,0.15]
```

```
  bool_normalize : True
  bool_image : False
  bool_shuffle : True


TRAIN_PARAM:
  num_epochs_INN : 5
  num_epochs_MLP : 2000
  batch_size : 128
  learning_rate : 1e-1
  bool_train_acc : False # measure train accuracy

  validation_period : 100
  bool_denormalize: False # or True, whether we denormalize when measuring errors
  error_type : "accuracy"
  patience : 10

PLOT:
  bool_plot: True
  plot_in_axis: [0,1] # plot input axis
  plot_out_axis: [2] # plot output axis
```

3) Run main.py

```
$ python ./pyinn/main.py
```

4. Benchmarking INN solver

The INN solver benchmark can be conducted by running the following command:

```
$ python ./pyinn/solver_benchmark/FEM_INN_PINN.py
```

User setups can be found at the beginning of the code “FEM_INN_PINN.py”:

```
################## User setup ###############
gpu_idx = 0
os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"  # GPU indexing
os.environ["CUDA_VISIBLE_DEVICES"] = str(gpu_idx)  # GPU indexing

# Problem settings
s_patches = [2]         # patch_size
ps = [-1]               # reproducing polynomial order. [0, 1, 2, 3]. -1 means that p is equal to s.
alpha_dils = [20]       # dilation parameter
nelems = [8, 16, 32, 64]        # number of segments [8, 16, 32, 64, 128, 256, 512, 1024, …]
elem_types = ['D1LN2N']

run_FEM = True
run_INN = True
run_PINN = True

plot_bool = True
non_uniform_mesh_bool = False

###########################################
```