Will Spaeth

2-14-20

Homework 2

#########################

# Supercomputer.py file #

#########################

#!/usr/bin/env python3

from collections import deque

import pickle

import os

import sys

import json

import subprocess

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.callbacks import EarlyStopping

from models import dnn

from symbiotic\_metrics import FractionOfVarianceAccountedFor

def main():

# Loop through all of the hyperparameters

rotation\_list = list(range(20))

n\_train\_folds\_list = [1, 2, 3, 5, 10, 18]

create\_index\_log(rotation\_list, n\_train\_folds\_list)

# Start a job for each hyperparameter

for rotation in rotation\_list:

for n\_train\_folds in n\_train\_folds\_list:

start\_training\_job(rotation, n\_train\_folds)

def start\_training\_job(rotation, n\_train\_folds):

"""

Starts a job for the fed arguments. This takes the form of a subprocess,

whether on a normal computer or supercomputer

"""

print("Starting job: Rotation {:02d}, # Training Folds {:02d}".format(rotation, n\_train\_folds))

# Decide which script to run

if "-s" in sys.argv:

script\_to\_run = ["sbatch", "supercomputer\_job.sh", "-s"]

else:

script\_to\_run = ["./standard\_job.sh"]

# Run chosen script with correct arguments

process = subprocess.Popen(

[\*script\_to\_run,

"-job", # Indicate to the subprocess that it is a subprocess

"-rotation={}".format(rotation),

"-n\_train\_folds={}".format(n\_train\_folds)

])

# Wait if not parallel

if "-p" not in sys.argv:

process.wait()

def parse\_args():

# Parse the hyperparameter arguments

for arg in sys.argv:

if "-rotation=" in arg:

rotation = int(arg.replace("-rotation=", ""))

elif "-n\_train\_folds=" in arg:

n\_train\_folds = int(arg.replace("-n\_train\_folds=", ""))

return rotation, n\_train\_folds

def train(rotation=0, n\_train\_folds=18):

print("PARAMETERS: Rotation {:02d}, # Training Folds {:02d}".format(rotation, n\_train\_folds))

# Rotate indices based on current rotation

rotation\_indices = get\_rotation\_indices(n\_folds=20, rotation=rotation)

# Get the training, validation, and test fold indices

fold\_inds = get\_set\_indices(rotation\_indices=rotation\_indices, n\_train\_folds=n\_train\_folds)

''' Load data

Key MI, Length 20, Shape (1193, 960)

Key theta, Length 20, Shape (1193, 2)

Key dtheta, Length 20, Shape (1193, 2)

Key ddtheta, Length 20, Shape (1193, 2)

Key torque, Length 20, Shape (1193, 2)

Key time, Length 20, Shape (1193, 1)

'''

if "-s" in sys.argv:

data\_path = "/home/fagg/ml\_datasets/bmi/bmi\_dataset.pkl"

else:

data\_path = "bmi\_dataset.pkl"

with open(data\_path, "rb") as fp:

hw2\_dataset = pickle.load(fp)

# Splits the data into its respective train, validation, and test sets / ins and outs

processed\_data = process\_dataset(hw2\_dataset, fold\_inds)

# Build model

model = dnn(

input\_size=(processed\_data["train"]["ins"].shape[1],),

hidden\_sizes=[100, 50],

output\_size=processed\_data["train"]["outs"].shape[1],

hidden\_act="elu",

output\_act="linear")

# Compile model with fvaf metric

fvaf = FractionOfVarianceAccountedFor(processed\_data["test"]["outs"].shape[1])

model.compile(optimizer="adam", loss="mse", metrics=[fvaf], verbose=2)

model.summary()

# Callbacks

es\_callback = EarlyStopping(

monitor="val\_loss",

patience=5,

restore\_best\_weights=True,

min\_delta=.0001)

# Train model

history = model.fit(

x=processed\_data["train"]["ins"],

y=processed\_data["train"]["outs"],

validation\_data = (processed\_data["val"]["ins"], processed\_data["val"]["outs"]),

epochs=100,

batch\_size=32,

callbacks=[es\_callback]

)

# Log results

log(model, processed\_data, fold\_inds, rotation, n\_train\_folds)

# Plot the torque and save figure

plot\_torque(model, processed\_data, rotation, n\_train\_folds)

def plot\_torque(model, data, rotation, n\_train\_folds):

"""Plots the torque graph"""

# Create results directory

save\_path = "results/"

if not os.path.exists(save\_path):

os.mkdir(save\_path)

# Create specific experiment directory

save\_path += "r{:02d}\_t{:02d}/".format(rotation, n\_train\_folds)

if not os.path.exists(save\_path):

os.mkdir(save\_path)

true\_torque = data["test"]["outs"][:, 0]

predicted\_torque = model.predict(data["test"]["ins"])[:, 0]

# Create and configure plot

fig = plt.figure()

ax = fig.add\_subplot(1, 1, 1)

ax.plot(data["test"]["time"], true\_torque, label="True Torque")

ax.plot(data["test"]["time"], predicted\_torque, label="Predicted Torque")

ax.legend()

plt.ylabel("Torque")

plt.xlabel("Time")

# Save plot

fig.savefig(save\_path + f"torque\_plot.png", dpi=fig.dpi)

def create\_index\_log(rotation\_list, n\_train\_folds\_list):

"""Write index to file that describes experiment hyperparameters"""

index = {

"rotation\_list": rotation\_list,

"n\_train\_folds\_list": n\_train\_folds\_list

}

fbase = "results/"

if not os.path.exists(fbase):

os.mkdir(fbase)

with open('{}index.json'.format(fbase), 'w') as f:

json.dump(index, f)

def log(model, data, fold\_inds, rotation, n\_train\_folds):

"""Log results to file"""

print("Logging results")

# Generate results

results = {}

results['predict\_train'] = model.predict(data["train"]["ins"])

results['eval\_train'] = model.evaluate(data["train"]["ins"], data["train"]["outs"])

results['predict\_val'] = model.predict(data["val"]["ins"])

results['eval\_val'] = model.evaluate(data["val"]["ins"], data["val"]["outs"])

results['predict\_test'] = model.predict(data["test"]["ins"])

results['eval\_test'] = model.evaluate(data["test"]["ins"], data["test"]["outs"])

results['folds'] = fold\_inds

results['rotation'] = rotation

results['n\_train\_folds'] = n\_train\_folds

# Create results directory

fbase = "results/"

if not os.path.exists(fbase):

os.mkdir(fbase)

fbase += "r{:02d}\_t{:02d}/".format(rotation, n\_train\_folds)

if not os.path.exists(fbase):

os.mkdir(fbase)

# Save results

with open("{}results.pkl".format(fbase, rotation, n\_train\_folds), "wb") as fp:

pickle.dump(results, fp)

fp.close()

# Create model directory

if not os.path.exists("{}/model/".format(fbase)):

os.mkdir("{}/model/".format(fbase))

# Save model

model.save("{}/model/".format(fbase))

def process\_dataset(dataset, fold\_inds):

"""

Process the dataset into the train, validation, and test folds;

Also split into ins & out sets

"""

processed\_data = {}

for key in fold\_inds.keys():

processed\_data[key] = split\_dataset(dataset, fold\_inds[key])

return processed\_data

def split\_dataset(dataset, inds):

# Placeholder for data splits

processed\_data = {

"ins": None,

"outs": [],

"time": None

}

for key in dataset.keys():

# Get folds for this key

folds = [dataset[key][ind] for ind in inds]

# Join the folds

joined = np.concatenate((folds), axis=0)

# See if the key is for the ins or outs of the dataset

if key == "MI":

processed\_data["ins"] = joined

elif key == "time":

processed\_data["time"] = joined

elif key == "torque":

processed\_data["outs"] = np.expand\_dims(joined[:, 1], axis=1)

return processed\_data

def get\_set\_indices(rotation\_indices, n\_train\_folds):

"""Get the fold indices for each set"""

inds = {}

inds["train"] = [rotation\_indices[i] for i in range(n\_train\_folds)]

inds["val"] = [rotation\_indices[len(rotation\_indices)-2]]

inds["test"] = [rotation\_indices[len(rotation\_indices)-1]]

return inds

def get\_rotation\_indices(n\_folds, rotation=0):

"""Rotate folds to get the right indices"""

fold\_list = list(range(n\_folds))

fold\_list = deque(fold\_list)

fold\_list.rotate(rotation)

fold\_list = list(fold\_list)

return fold\_list

if \_\_name\_\_ == "\_\_main\_\_":

# If this is a subprocess, run the training program

if "-job" in sys.argv:

rotation, n\_train\_folds = parse\_args()

try:

train(rotation=rotation, n\_train\_folds=n\_train\_folds)

# If any exception occurs, write to error folder to differentiate between all the job outputs

except Exception as e:

fbase = "error/"

if not os.path.exists(fbase):

os.mkdir(fbase)

with open("{}r{:02d}\_t{:02d}\_err.txt".format(fbase, rotation, n\_train\_folds), "a") as f:

err\_str = "Error: {}".format(e)

f.write(err\_str)

else:

main()

#################

# local.py file #

#################

#!/usr/bin/env python3

import sys

import json

import pickle

import os

import subprocess

import matplotlib.pyplot as plt

def main():

# Use -s argument to scp results from supercomputer before continuing

if "-s" in sys.argv:

script\_to\_run = [

"scp",

"-r",

"jwspaeth@schooner.oscer.ou.edu:/home/jwspaeth/workspaces/advanced-ml/homework\_2/results",

"./"]

process = subprocess.Popen([\*script\_to\_run])

process.wait()

# Read index file

index = load\_index\_log()

rotation\_list = index["rotation\_list"]

n\_train\_folds\_list = index["n\_train\_folds\_list"]

# Load all results

results = []

for rotation in rotation\_list:

for n\_train\_folds in n\_train\_folds\_list:

with open("results/r{:02d}\_t{:02d}/results.pkl".format(rotation, n\_train\_folds), "rb") as fp:

results.append(pickle.load(fp))

# Compute average fvafs

avg\_fvafs = compute\_avg\_fvafs(results, n\_train\_folds\_list)

# Plot and save all the fvafs

plot\_fvaf(avg\_fvafs, n\_train\_folds\_list, "train")

plot\_fvaf(avg\_fvafs, n\_train\_folds\_list, "val")

plot\_fvaf(avg\_fvafs, n\_train\_folds\_list, "test")

def load\_index\_log():

with open("results/index.json") as f:

return json.load(f)

def plot\_fvaf(avg\_fvafs, n\_train\_folds\_list, set\_name):

"""Plot fvaf based on the set name"""

# Create results directory

save\_path = "results/"

if not os.path.exists(save\_path):

os.mkdir(save\_path)

# Create plots directory

save\_path += "fvaf\_plots/"

if not os.path.exists(save\_path):

os.mkdir(save\_path)

# Create and configure plot

fig = plt.figure()

ax = fig.add\_subplot(1, 1, 1)

ax.plot(n\_train\_folds\_list, avg\_fvafs[set\_name])

plt.ylabel("Average FVAF")

plt.xlabel("Number of Training Folds")

if set\_name == "train":

plt.title("Training Set")

elif set\_name == "val":

plt.title("Validation Set")

elif set\_name == "test":

plt.title("Test Set")

# Save

fig.savefig("{}{}\_fvaf\_plot.png".format(save\_path, set\_name), dpi=fig.dpi)

def compute\_avg\_fvafs(results, n\_train\_folds\_list):

# Sum all the fvafs and count how many values there are

# Each index represents a n\_train\_folds hyperparameter

avg\_fvafs = {

"train": [0]\*len(n\_train\_folds\_list),

"val": [0]\*len(n\_train\_folds\_list),

"test": [0]\*len(n\_train\_folds\_list)

}

# Loop through each split

for key in avg\_fvafs.keys():

# Start summing and count the fvaf values

sum\_fvafs = [0]\*len(n\_train\_folds\_list)

count\_fvafs = [0]\*len(n\_train\_folds\_list)

for i in range(len(n\_train\_folds\_list)):

for result in results:

if result["n\_train\_folds"] == n\_train\_folds\_list[i]:

sum\_fvafs[i] += result["eval\_{}".format(key)][1]

count\_fvafs[i] += 1

# Create average fvafs based on the sum and counts

for i in range(len(n\_train\_folds\_list)):

if count\_fvafs[i] != 0:

avg\_fvafs[key][i] = sum\_fvafs[i] / count\_fvafs[i]

return avg\_fvafs

if \_\_name\_\_ == "\_\_main\_\_":

main()

##################

# models.py file #

##################

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense

def dnn(input\_size, hidden\_sizes, output\_size, hidden\_act="sigmoid", output\_act="tanh"):

"""Construct a simple deep neural network"""

inputs = Input(shape=input\_size)

hidden\_stack\_out = hidden\_stack(hidden\_sizes, hidden\_act)(inputs)

outputs = Dense(output\_size, activation=output\_act)(hidden\_stack\_out)

return Model(inputs=inputs, outputs=outputs)

def hidden\_stack(hidden\_sizes, hidden\_act="sigmoid"):

"""Represents a stack of neural layers"""

layers = []

for size in hidden\_sizes:

layers.append(Dense(size, activation=hidden\_act))

def hidden\_stack\_layer(inputs):

"""Layer hook for stack"""

for i in range(len(layers)):

if i == 0:

carry\_out = layers[i](inputs)

else:

carry\_out = layers[i](carry\_out)

return carry\_out

return hidden\_stack\_layer

#############################

# supercomputer\_job.sh file #

#############################

#!/bin/bash

#SBATCH --partition=normal

#SBATCH --ntasks=1

#SBATCH --mem=2000

#SBATCH --output=job-output/subprocess-%j-stdout.txt

#SBATCH --error=job-output/subprocess--%j-stderr.txt

#SBATCH --time=7:00:00

#SBATCH --job-name=subprocess-%j

#SBATCH --mail-user=john.w.spaeth-1@ou.edu

#SBATCH --mail-type=ALL

#SBATCH --chdir=/home/jwspaeth/workspaces/advanced-ml/homework\_2/

#SBATCH --wait

python3 supercomputer.py $@

#############################

# standard\_job.sh file #

#############################

#!/bin/bash

python3 supercomputer.py $@

##############

# Torque Figure #

##############

A screenshot of a cell phone

Description automatically generated

############

# FVAF Plots #

############

A close up of a map

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A close up of a map

Description automatically generated