Will Spaeth

2-23-20

Homework 3

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

# supercomputer.py #

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

#!/usr/bin/env python3

from collections import deque

import pickle

import os

import sys

import json

import subprocess

import itertools

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():

"""Spits out training jobs for each configuration"""

# Create error file

with open("error/err.txt", "w") as f:

pass

# Create option dictionary

options = {

"rotation": list(range(20)),

"n\_train\_folds": [1, 2, 3, 5, 10, 18],

"dropout": [0, .3, .6, .8]

#"l2": [0, .001, .01, .1]

}

option\_combinations = create\_combinations(options)

option\_combinations = create\_index\_log(options, option\_combinations)

# Start a job for each hyperparameter

for option\_combo in option\_combinations:

start\_training\_job(\*\*option\_combo)

def create\_combinations(option\_dictionaries):

'''

Used to create a list of dictionaries containing all possible combinations

of input dictionary arguments

Found on tutorial website: https://riptutorial.com/python/example/10160/all-combinations-of-dictionary-values

'''

keys = option\_dictionaries.keys()

values = (option\_dictionaries[key] for key in keys)

combinations = [dict(zip(keys, combination)) for combination in itertools.product(\*values)]

for i in range(len(combinations)):

combinations[i]["experiment\_num"] = i

return combinations

def create\_index\_log(options, option\_combinations):

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

fbase = "results/"

if not os.path.exists(fbase):

os.mkdir(fbase)

batch\_num = 0

while ( os.path.exists("{}batch\_{}/".format(fbase, batch\_num)) ):

batch\_num += 1

fbase = "{}batch\_{}/".format(fbase, batch\_num)

os.mkdir(fbase)

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

f.write("Number of experiments: {}\n".format(len(option\_combinations)))

json.dump(options, f)

f.write("\n")

for i in range(len(option\_combinations)):

option\_combinations[i]["batch\_num"] = batch\_num

json.dump(option\_combinations[i], f)

f.write("\n")

return option\_combinations

def start\_training\_job(\*\*kwargs):

"""

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

whether on a normal computer or supercomputer

"""

print("Starting job:\n\t{}".format(kwargs))

# 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"]

# Build script with hyperparameters

full\_command = [

\*script\_to\_run,

"-job"

]

for key, value in kwargs.items():

full\_command.append("--{}={}".format(key, value))

# Run chosen script with correct arguments

process = subprocess.Popen(full\_command)

# Wait if not parallel

if "-p" not in sys.argv:

process.wait()

def parse\_args():

# Parse the hyperparameter arguments

kwargs = {}

for arg in sys.argv:

if "--" in arg:

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

key, value = arg.split("=")

kwargs[key] = value

return kwargs

def train(\*\*kwargs):

print("PARAMETERS: {}".format(kwargs))

# Unpack relevant kwargs

rotation = int(kwargs["rotation"])

n\_train\_folds = int(kwargs["n\_train\_folds"])

experiment\_num = int(kwargs["experiment\_num"])

batch\_num = int(kwargs["batch\_num"])

if "dropout" in kwargs.keys():

dropout = float(kwargs["dropout"])

else:

dropout = 0

if "l2" in kwargs.keys():

l2 = float(kwargs["l2"])

else:

l2 = 0

# 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 = "../homework\_2/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=[300, 150, 100, 50, 10],

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

hidden\_act="elu",

output\_act="linear",

dropout=dropout,

l2=l2)

# 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=20,

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=10000,

batch\_size=32,

callbacks=[es\_callback]

)

# Log results

log(model, processed\_data, kwargs)

# Plot the torque and save figure

if experiment\_num == 0:

plot\_shoulder\_orientation(model, processed\_data, kwargs)

def plot\_shoulder\_orientation(model, data, kwarg\_dict):

"""Plots the torque graph"""

# Create results directory

save\_path = "results/"

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

os.mkdir(save\_path)

save\_path += "batch\_{}/".format(kwarg\_dict["batch\_num"])

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

os.mkdir(save\_path)

# Create specific experiment directory

save\_path += "experiment\_{}/".format(kwarg\_dict["experiment\_num"])

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

os.mkdir(save\_path)

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

predicted\_orientation = 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\_orientation, label="True Orientation")

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

ax.legend()

plt.ylabel("Orientation")

plt.xlabel("Time")

# Save plot

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

def log(model, data, kwarg\_dict):

"""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"])

for key, value in kwarg\_dict.items():

results[key] = value

# Create results directory

fbase = "results/"

if not os.path.exists(fbase):

os.mkdir(fbase)

fbase += "batch\_{}/".format(kwarg\_dict["batch\_num"])

if not os.path.exists(fbase):

os.mkdir(fbase)

fbase += "experiment\_{}/".format(kwarg\_dict["experiment\_num"])

if not os.path.exists(fbase):

os.mkdir(fbase)

# Save results

with open("{}results\_dict.pkl".format(fbase), "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 == "theta":

processed\_data["outs"] = np.expand\_dims(joined[:, 0], 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:

kwargs = parse\_args()

try:

train(\*\*kwargs)

# 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("{}err.txt".format(fbase), "a") as f:

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

f.write(err\_str)

else:

main()

#########

# local.py #

#########

#!/usr/bin/env python3

import sys

import json

import pickle

import os

import subprocess

import matplotlib.pyplot as plt

import numpy as np

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\_3/results",

"./"]

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

process.wait()

# Load dropout experiments and results

dropout\_options, dropout\_experiments = load\_index\_log(batch\_num=0)

dropout\_results = [load\_result\_from\_experiment(experiment) for experiment in dropout\_experiments]

# Load l2 experiments and results

l2\_options, l2\_experiments = load\_index\_log(batch\_num=1)

l2\_results = [load\_result\_from\_experiment(experiment) for experiment in l2\_experiments]

# Compute fvaf curves for each key value

# (training folds x hyperparameter value), so average across rotations

dropout\_avg\_fvafs = compute\_avg\_fvaf\_curves(dropout\_results,

dropout\_options["n\_train\_folds"],

dropout\_options["dropout"],

"dropout",

dropout\_options["rotation"])

l2\_avg\_fvafs = compute\_avg\_fvaf\_curves(l2\_results,

l2\_options["n\_train\_folds"],

l2\_options["l2"],

"l2",

l2\_options["rotation"])

# Plot fvaf curves for each key value

plot\_fvaf\_curves(dropout\_options["n\_train\_folds"],

dropout\_options["dropout"],

dropout\_avg\_fvafs["val"],

"Validation Dropout")

plot\_fvaf\_curves(l2\_options["n\_train\_folds"],

l2\_options["l2"],

l2\_avg\_fvafs["val"],

"Validation L2")

# Get argmax for the best hyperparameter values

dropout\_argmax\_fvafs = np.argmax(dropout\_avg\_fvafs["val"], axis=1)

l2\_argmax\_fvafs = np.argmax(l2\_avg\_fvafs["val"], axis=1)

# Build arrays containing test values for best validation models

test\_dropout = []

test\_l2 = []

for i in range(dropout\_argmax\_fvafs.shape[0]):

test\_dropout.append(dropout\_avg\_fvafs["test"][i, dropout\_argmax\_fvafs[i]])

test\_l2.append(l2\_avg\_fvafs["test"][i, l2\_argmax\_fvafs[i]])

test\_dropout = np.expand\_dims(np.concatenate(test\_dropout), axis=1)

test\_l2 = np.expand\_dims(np.concatenate(test\_l2), axis=1)

test = np.concatenate((test\_dropout, test\_l2), axis=1)

# Plot the test set fvaf for the argmaxes

plot\_fvaf\_curves(dropout\_options["n\_train\_folds"],

["Dropout", "L2"],

test,

"Test Curves")

def load\_result\_from\_experiment(experiment):

"""Load result of given experiment"""

file\_str = "results/batch\_{}/experiment\_{}/results\_dict.pkl".format(experiment["batch\_num"], experiment["experiment\_num"])

with open(file\_str, "rb") as fp:

return pickle.load(fp)

def load\_index\_log(batch\_num):

"""Load the index log of the batch"""

experiments = []

with open("results/batch\_{}/index.txt".format(batch\_num), "r") as f:

contents = f.read().split("\n")

options = json.loads(contents[1])

for i, experiment\_str in enumerate(contents[2:len(contents)-1]):

experiments.append(json.loads(experiment\_str))

return options, experiments

def plot\_fvaf\_curves(n\_train\_folds\_list, key\_list, curves, plot\_name):

"""Plot fvaf based on the given curves"""

# 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)

for i in range(curves.shape[1]):

ax.plot(n\_train\_folds\_list, curves[:, i], label=str(key\_list[i]))

plt.legend()

plt.ylabel("Average FVAF")

plt.xlabel("Number of Training Folds")

plt.title(plot\_name)

# Save

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

def get\_matching\_result(results, n\_train\_fold, key\_val, key\_name, rotation\_val):

"""Find the result that matches the given values"""

for i, result in enumerate(results):

if (int(result["n\_train\_folds"]) == n\_train\_fold) and (float(result[key\_name]) == key\_val) and (int(result["rotation"]) == rotation\_val):

return result

def compute\_avg\_fvaf\_curves(results, n\_train\_fold\_list, key\_list, key\_name, rotation\_list):

"""Gets the average fvaf across rotations for both the validation and test sets"""

# Create 3d array of results

fvaf\_curves = {

"val": np.zeros(shape=(len(n\_train\_fold\_list), len(key\_list), len(rotation\_list)), dtype=object),

"test": np.zeros(shape=(len(n\_train\_fold\_list), len(key\_list), len(rotation\_list)), dtype=object)

}

for i, n\_train\_fold in enumerate(n\_train\_fold\_list):

for j, key\_val in enumerate(key\_list):

for k, rotation\_val in enumerate(rotation\_list):

# Get result with the matching parameters

result = get\_matching\_result(results, n\_train\_fold, key\_val, key\_name, rotation\_val)

# Store the validation or test fvaf

fvaf\_curves["val"][i, j, k] = result["eval\_val"][1]

fvaf\_curves["test"][i, j, k] = result["eval\_test"][1]

# Average across rotations

fvaf\_curves["val"] = np.average(fvaf\_curves["val"], axis=2)

fvaf\_curves["test"] = np.average(fvaf\_curves["test"], axis=2)

return fvaf\_curves

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

main()

###########

# models.py #

###########

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense, Dropout

from tensorflow.keras import regularizers

def pipe\_model(inputs, layers):

"""Pipes an input through a model to obtain the output hook"""

for i in range(len(layers)):

if i == 0:

carry\_out = layers[i](inputs)

else:

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

return carry\_out

def dnn(input\_size, hidden\_sizes, output\_size, hidden\_act="sigmoid", output\_act="tanh", dropout=0, l2=0):

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

layers = []

# Dropout layer if applicable

if dropout > 0:

layers.append(Dropout(rate=dropout))

# Add hidden layers with respective dropout and l2 values

layers.append(

hidden\_stack(hidden\_sizes, hidden\_act, dropout=dropout, l2=l2)

)

# l2 regularization if applicable

if l2 > 0:

layers.append(

Dense(

output\_size,

activation=output\_act,

kernel\_regularizer=regularizers.l2(l2)

)

)

else:

layers.append(

Dense(

output\_size,

activation=output\_act

)

)

# Pipe model by feeding through input placeholder

inputs = Input(shape=input\_size)

outputs = pipe\_model(inputs, layers)

return Model(inputs=inputs, outputs=outputs)

def hidden\_stack(hidden\_sizes, hidden\_act="sigmoid", dropout=0, l2=0):

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

layers = []

for size in hidden\_sizes:

# Apply l2 if applicable

if l2 > 0:

layers.append(Dense(

size,

activation=hidden\_act,

kernel\_regularizer=regularizers.l2(l2)

)

)

else:

layers.append(Dense(

size,

activation=hidden\_act,

)

)

# Apply dropout if applicable

if dropout > 0:

layers.append(Dropout(rate=dropout))

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 #

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

#!/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\_3/

#SBATCH --wait

python3 supercomputer.py $@

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

# standard\_job.sh #

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

#!/bin/bash

python3 supercomputer.py $@

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

# Dropout Validation Curves #

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

A close up of a map

Description automatically generated

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

# L2 Validation Curves #

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

A screenshot of a map

Description automatically generated

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

# Dropout and L2 Test Curves #

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

A close up of a map

Description automatically generated