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),
```

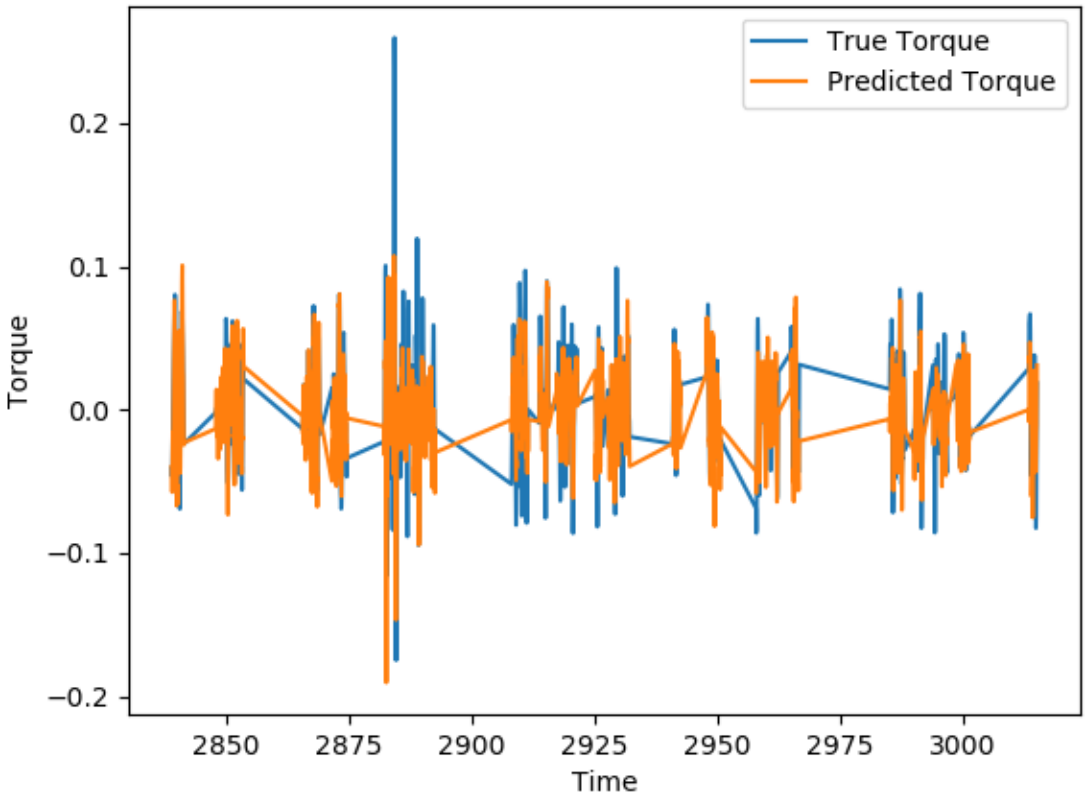
```
        "val": [0]*len(n_train_folds_list),
        "test": [0]*len(n_train_folds_list)
    }

    # 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
```

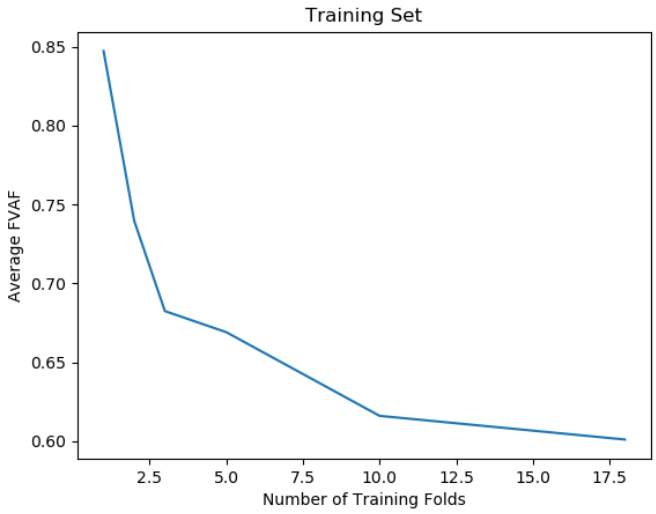
```
##############################
# supercomputer_job.sh file #
##############################
#!/bin/bash

#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 $@
```

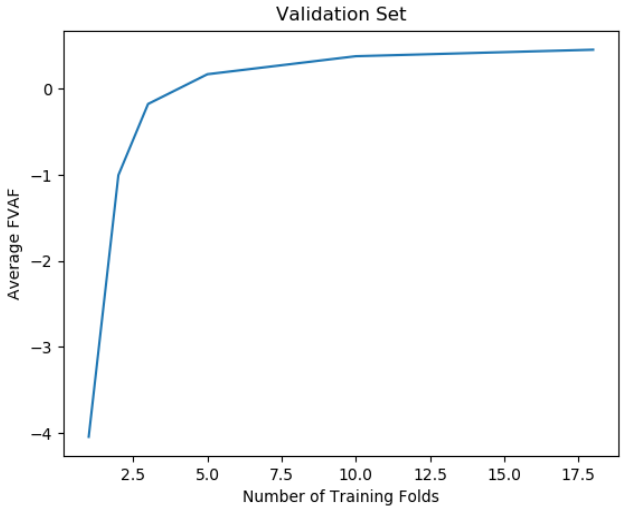
```
##############################
# standard_job.sh file #
##############################

#!/bin/bash

python3 supercomputer.py $@
```

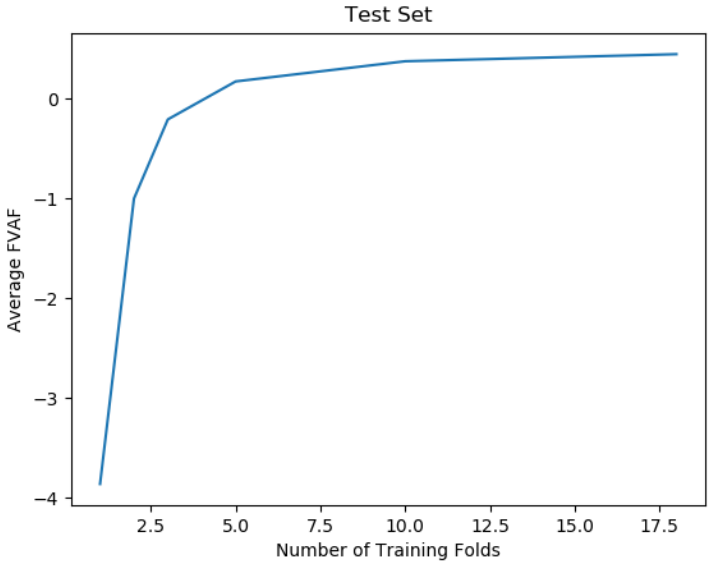
###############
Torque Figure
###############



#############
FVAF Plots
#############


Training Set
Average FVAF
0.85
0.80
0.75
0.70
0.65
0.60
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Number of Training Folds


Validation Set
Average FVAF
0
−1
−2
−3
−4
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Number of Training Folds


Test Set
Average FVAF
0
−1
−2
−3
−4
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Number of Training Folds