

HOMework-3

REPORT

CSE 421

Efe Kayadelen

Student ID : 20220701088

Samet Alper Özdemir

Student ID : 20210702021

GITHUB REPO: <https://github.com/blackwh1p/CSE421-Homeworks>

Part-1 (Q1):

C Code (Mbed):

This is the inference engine running on the microcontroller. It initializes the UART serial communication and the random-number-generator. In its main loop, it operates in "PC Input" mode: it waits for a packet of 5 temperature features sent from python_1.2 code, passes those features into the linear_reg_predict function (which uses the exported model weights), and transmits the resulting temperature prediction back to the PC for verification.

Code:

```
#include "mbed.h"
#include "lib_serial.h"
#include "lib_rng.h"
#include "linear_reg_inference.h"

// Configuration
#define INPUT_PC      1
#define INPUT_MCU     2
#define INPUT         INPUT_PC
float input_buffer[NUM_FEATURES];
float prediction[1];

int main()
{
    // 1. Initialize Serial and RNG
    LIB_UART_Init();
    LIB_RNG_Init();

    while (1) {
        #if (INPUT == INPUT_PC)
            // 2a. Receive 5 float features from Python (setup_reg_lr.py)
            if (LIB_SERIAL_Receive(input_buffer, NUM_FEATURES, TYPE_F32) == SERIAL_OK) {

                // 3. Perform the calculation
                linear_reg_predict(input_buffer, prediction);

                // 4. Send the result back to Python for verification
                LIB_SERIAL_Transmit(prediction, 1, TYPE_F32);
            }
        #elif (INPUT == INPUT_MCU)
            // 2b. Standalone testing using Random Data
            for (int i = 0; i < NUM_FEATURES; i++) {
                input_buffer[i] = (float) (LIB_RNG_GetRandomNumber() % 1000) / 10.0f;
            }
            // Send input to PC for logging
            LIB_SERIAL_Transmit(input_buffer, NUM_FEATURES, TYPE_F32);

            linear_reg_predict(input_buffer, prediction);

            // Send output to PC
            LIB_SERIAL_Transmit(prediction, 1, TYPE_F32);

            ThisThread::sleep_for(1s);
        #endif
    }
}
```

Python 1.1 Code (setup_reg_lr):

This code loads the original trained model and a CSV dataset on our PC. It iterates through the dataset, sending a batch of temperature samples to the MCU via a serial port (COM6). Simultaneously, it calculates the prediction on the PC and compares it against the result sent back by the MCU, printing a "MATCH" or "MISMATCH" status to ensure the embedded model is performing identically to the desktop version.

Code:

```
import os
import numpy as np
import pandas as pd
import serial
import struct
from sklearn2c import LinearRegressor

# --- CONFIG ---
PORT = "COM6" # Ensure this is correct
BAUD = 115200

script_dir = os.path.dirname(os.path.abspath(__file__))
model_path = os.path.join(script_dir, "../temperature_pred_linreg.joblib")
csv_path = os.path.join(script_dir, "dataset/temperature_dataset.csv")

# --- LOAD DATA ---
print("Loading model and dataset...")
linear = LinearRegressor.load(model_path)
df = pd.read_csv(csv_path)
y_data = df["Room_Temp"][:4]
test_samples = []
for i in range(5, 0, -1):
    test_samples.append(y_data.shift(i))
test_samples = pd.concat(test_samples, axis=1)[5:].to_numpy()

total_samples = len(test_samples)
print(f"Total samples to process: {total_samples}")

# --- OPEN SERIAL ---
try:
    ser = serial.Serial(PORT, BAUD, timeout=1)
    ser.flushInput()
    ser.flushOutput()
    print(f"Connected to {PORT}. Waiting for MCU...")
except Exception as e:
    print(f"Error: {e}")
    exit()

# --- PROCESSING LOOP ---
# We use range(total_samples) so it stops when the data ends
for i in range(total_samples):
    found_packet = False
    pc_prediction = 0.0

    while not found_packet:
        char = ser.read(1)
        if char == b'S':
```

```

suffix = ser.read(2)
header = "S" + suffix.decode('ascii', errors='ignore')

if header == "STR":
    # 1. Send Data to MCU
    sample = test_samples[i].astype(np.float32)
    ser.write(sample.tobytes())

    # 2. Calculate PC Prediction
    pred = linear.predict(sample.reshape(1, -1))
    pc_prediction = float(np.ravel(pred)[0])

    # 3. Wait for MCU to send result back (STW)
    # We loop briefly to find the STW response
    for _ in range(100):
        res_char = ser.read(1)
        if res_char == b'S':
            res_suffix = ser.read(2)
            if res_suffix == b'TW':
                ser.read(1) # skip type
                ser.read(4) # skip length
                payload = ser.read(4)
                mcu_prediction = struct.unpack('f', payload)[0]

            # 4. Print Comparison
            status = "MATCH" if abs(pc_prediction - mcu_prediction) < 0.05 else "MISMATCH"
            print(f"Sample {i+1}/{total_samples}: PC: {pc_prediction:.4f} | MCU: {mcu_prediction:.4f} | {status}")

            found_packet = True
            break

print("\n" + "="*30)
print("TEST COMPLETE: All samples processed.")
print("="*30)
ser.close()

```

Python_1.2 Code (export_reg_lr.py):

This is the deployment tool. It uses the sklearn2c library to load the saved .joblib model and converts the high-level Python model into low-level C source files (.h and .c). Specifically, it extracts the weights and the bias into a format that the Mbed compiler can understand, effectively "transferring" the intelligence of your trained model into the microcontroller's memory.

```

import os
from sklearn2c import LinearRegressor

# Detect current directory
script_dir = os.path.dirname(os.path.abspath(__file__))

# Load the model we just trained with the new main.py
model_path = os.path.join(script_dir, "../temperature_pred_linreg.joblib")
export_path = os.path.join(script_dir, "../linear_reg_config")

print(f"Exporting model from: {model_path}")

```

```

try:
    # Load the model (Now it will be the correct sklearn2c type)
    linear_regressor = LinearRegressor.load(model_path)
    # Export to .h and .c
    linear_regressor.export(export_path)

    print("\nSUCCESS!")
    print(f"Files generated in: {script_dir}")
    print("- linear_reg_config.h")
    print("- linear_reg_config.c")

    # Print the values for your reference
    print("\nCheck your linear_reg_config.h file.")
    print("You will need the COEFFS and OFFSET values for Mbed.")

except Exception as e:
    print(f"ERROR: {e}")

```

Part-2 (Q2):

This Python script implements the training phase for the Human Activity Recognition (HAR) application. The objective is to utilize a single-neuron neural network to classify whether a user is "Walking" based on tri-axial accelerometer data from the WISDM dataset.

Code:

```

import os
import numpy as np
import pandas as pd
import keras
from sklearn import metrics
from matplotlib import pyplot as plt

# Import your local utility functions
from read_data import read_data
from create_features import create_features

# --- PATH CONFIGURATION ---
# This ensures the script looks for the 'Data' folder in the same directory as main.py
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
WISDM_PATH = os.path.join(BASE_DIR, "Data", "WISDM_ar_v1.1_raw.txt")
KERAS_MODEL_DIR = os.path.join(BASE_DIR, "Models")

# Ensure the Models directory exists
if not os.path.exists(KERAS_MODEL_DIR):
    os.makedirs(KERAS_MODEL_DIR)

# --- DATA PROCESSING ---
TIME_PERIODS = 80
STEP_DISTANCE = 40

print(f>Loading data from: {WISDM_PATH}")
data_df = read_data(WISDM_PATH)

# Textbook split: Users <= 28 for training, > 28 for testing
df_train = data_df[data_df["user"] <= 28]
df_test = data_df[data_df["user"] > 28]

train_segments_df, train_labels = create_features(df_train, TIME_PERIODS, STEP_DISTANCE)
test_segments_df, test_labels = create_features(df_test, TIME_PERIODS, STEP_DISTANCE)

```

```

# --- MODEL DEFINITION (Section 10.7: Single Neuron) ---
model = keras.models.Sequential([
    # Input shape [10] corresponds to the 10 features extracted in Section 5.4
    keras.layers.Dense(1, input_shape=[10], activation='sigmoid')
])

model.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-3),
              loss=keras.losses.BinaryCrossentropy(),
              metrics=[keras.metrics.BinaryAccuracy(),
                      keras.metrics.FalseNegatives()])

# --- LABEL ENCODING ---
# Map "Walking" to 0 and all others to 1 for binary classification
train_labels_binary = np.where(train_labels == "Walking", 0, 1).astype(int)
test_labels_binary = np.where(test_labels == "Walking", 0, 1).astype(int)

# --- TRAINING ---
train_segments_np = train_segments_df.to_numpy()
test_segments_np = test_segments_df.to_numpy()

model.fit(train_segments_np, train_labels_binary, epochs=50, verbose=1)

# --- EVALUATION ---
perceptron_preds = model.predict(test_segments_np)
y_pred = (perceptron_preds > 0.5).astype(int)

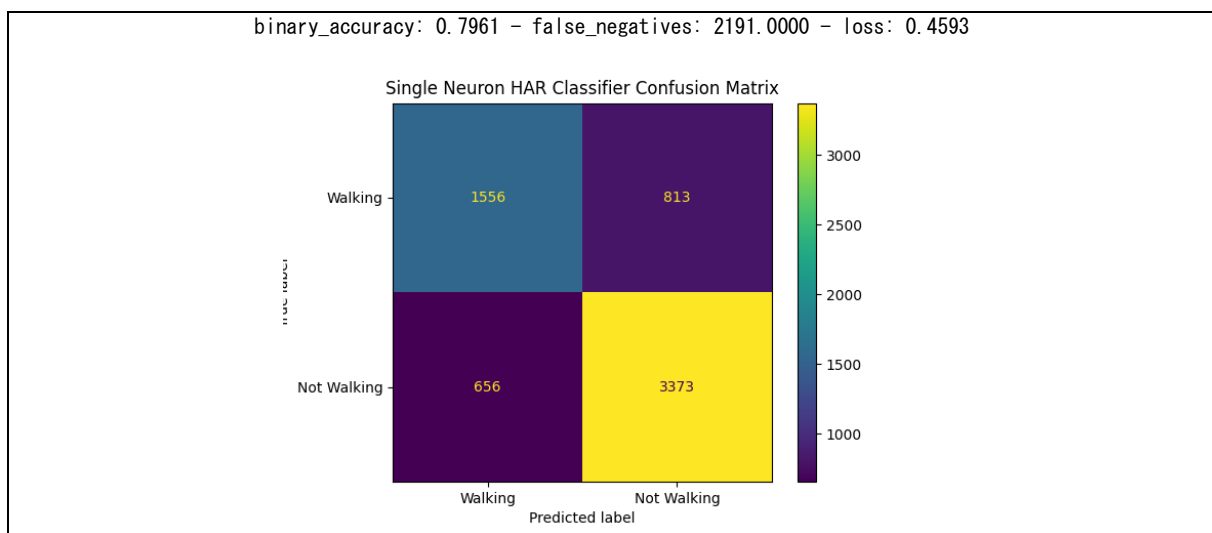
conf_matrix = metrics.confusion_matrix(test_labels_binary, y_pred)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix=conf_matrix,
                                             display_labels=["Walking", "Not Walking"])

cm_display.plot()
plt.title("Single Neuron HAR Classifier Confusion Matrix")
plt.show()

# --- SAVE MODEL ---
model_save_path = os.path.join(KERAS_MODEL_DIR, "har_perceptron.h5")
model.save(model_save_path)
print(f"Model saved to: {model_save_path}")

```

Results:



Part-3 (Q3):

This Python script facilitates the training of a Keyword Spotting system using the Free Spoken Digit Dataset (FSDD). The goal is to train a single-neuron perceptron to recognize a specific spoken digit (Digit 0) against all other digits in a binary classification task.

Code:

```
import os
import numpy as np
import scipy.signal as sig
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from matplotlib import pyplot as plt
import keras
import tensorflow as tf
import sys

# Ensure mfcc_func.py is in your C:\Users\ASUS\Desktop\Fall-2025\CSE 421\hw3\10.8\ folder
from mfcc_func import create_mfcc_features

# --- 1. PATH CONFIGURATION ---
BASE_DIR = os.path.dirname(os.path.abspath(__file__))

# Based on your note "data files are inside the recordings file"
# We check if they are in Data/recordings/ OR Data/recordings/recordings/
FSDD_PATH = os.path.join(BASE_DIR, "Data", "recordings")

# --- 2. VERIFY DATA ---
if not os.path.exists(FSDD_PATH):
    print(f"ERROR: Folder not found at {FSDD_PATH}")
    sys.exit()

all_files = os.listdir(FSDD_PATH)
recordings_list = [os.path.join(FSDD_PATH, f) for f in all_files if f.endswith('.wav')]

# If empty, try one level deeper (sometimes the zip extraction creates a nested folder)
if len(recordings_list) == 0:
    DEEPER_PATH = os.path.join(FSDD_PATH, "recordings")
    if os.path.exists(DEEPER_PATH):
        FSDD_PATH = DEEPER_PATH
        all_files = os.listdir(FSDD_PATH)
        recordings_list = [os.path.join(FSDD_PATH, f) for f in all_files if f.endswith('.wav')]

print(f"--- PATH CHECK ---")
print(f"Looking in: {FSDD_PATH}")
print(f"Found {len(recordings_list)} .wav files.")

if len(recordings_list) == 0:
    print("CRITICAL ERROR: No .wav files found. Script cannot continue.")
    sys.exit()

# --- 3. PARAMETERS ---
FFTSize = 1024
sample_rate = 8000
numOfMelFilters = 20
numOfDctOutputs = 13
window = sig.get_window("hamming", FFTSize)

# --- 4. SPLIT AND EXTRACT ---
```

```

# "yweweler" is one of the speakers in the FSDD dataset used for testing
test_list = {record for record in recordings_list if "yweweler" in os.path.basename(record)}
train_list = set(recordings_list) - test_list

print(f"Training: {len(train_list)} files | Testing: {len(test_list)} files")
print("Extracting MFCC features (Please wait)...")

train_mfcc_features, train_labels = create_mfcc_features(train_list, FFTSize, sample_rate, numOfMelFilters,
numOfDctOutputs, window)
test_mfcc_features, test_labels = create_mfcc_features(test_list, FFTSize, sample_rate, numOfMelFilters,
numOfDctOutputs, window)

# --- 5. MODEL DEFINITION ---
model = keras.models.Sequential([
    keras.layers.Dense(1, input_shape=[numOfDctOutputs * 2], activation='sigmoid')
])

model.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-3),
              loss=keras.losses.BinaryCrossentropy(),
              metrics=[keras.metrics.BinaryAccuracy()])

# --- 6. LABELING & TRAINING ---
# Convert to Binary: Target Digit 0 (Label 0) vs All Others (Label 1)
train_labels_bin = np.where(train_labels == 0, 0, 1)
test_labels_bin = np.where(test_labels == 0, 0, 1)

print("Starting training for 50 epochs...")
model.fit(train_mfcc_features, train_labels_bin, epochs=50, verbose=1, class_weight={0: 10., 1: 1.})

# --- 7. SAVE ---
KERAS_MODEL_DIR = os.path.join(BASE_DIR, "Models")
if not os.path.exists(KERAS_MODEL_DIR):
    os.makedirs(KERAS_MODEL_DIR)

model.save(os.path.join(KERAS_MODEL_DIR, "kws_perceptron.h5"))
print(f"Done! Model saved in {KERAS_MODEL_DIR}")

```

Result:

```
binary_accuracy: 0.8372 - loss: 0.5497
```

Part-4 (Q4):

This script implements a Handwritten Digit Recognition system using the MNIST dataset. It trains a single-neuron classifier to distinguish "Digit 0" from all other digits (1–9) based on geometric shape descriptors.

Code:

```
import os
```

```

import numpy as np
import cv2
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import keras
from matplotlib import pyplot as plt
import sys

# --- 1. PATH CONFIGURATION ---
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
# Data is inside 'MNIST-dataset' which is inside 'Data'
MNIST_PATH = os.path.join(BASE_DIR, "Data", "MNIST-dataset")
KERAS_MODEL_DIR = os.path.join(BASE_DIR, "Models")

if not os.path.exists(KERAS_MODEL_DIR):
    os.makedirs(KERAS_MODEL_DIR)

# --- 2. OFFLINE DATA LOADING ---
# Since you have data locally, we define a function to read MNIST uubyte files
def load_mnist_local(path, kind='train'):
    import struct
    labels_path = os.path.join(path, f'{kind}-labels.idx1-uubyte')
    images_path = os.path.join(path, f'{kind}-images.idx3-uubyte')

    with open(labels_path, 'rb') as lbpath:
        magic, n = struct.unpack('>II', lbpath.read(8))
        labels = np.fromfile(lbpath, dtype=np.uint8)

    with open(images_path, 'rb') as imgpath:
        magic, num, rows, cols = struct.unpack('>IIII', imgpath.read(16))
        images = np.fromfile(imgpath, dtype=np.uint8).reshape(len(labels), 28, 28)

    return images, labels

print(f"Loading data from {MNIST_PATH}...")
try:
    train_images, train_labels = load_mnist_local(MNIST_PATH, kind='train')
    test_images, test_labels = load_mnist_local(MNIST_PATH, kind='t10k')
    print(f"Loaded {len(train_images)} training and {len(test_images)} test images.")
except FileNotFoundError as e:
    print(f"Error: Could not find MNIST files. Ensure filenames are exactly 'train-images.idx3-uubyte', etc.")
    sys.exit()

# --- 3. FEATURE EXTRACTION (HU MOMENTS) ---
# Hu Moments provide shape descriptors invariant to scale/rotation

train_huMoments = np.empty((len(train_images), 7))
test_huMoments = np.empty((len(test_images), 7))

print("Extracting Hu Moments...")
for train_idx, train_img in enumerate(train_images):
    train_moments = cv2.moments(train_img, True)
    train_huMoments[train_idx] = cv2.HuMoments(train_moments).reshape(7)

for test_idx, test_img in enumerate(test_images):
    test_moments = cv2.moments(test_img, True)
    test_huMoments[test_idx] = cv2.HuMoments(test_moments).reshape(7)

# --- 4. Z-SCORE NORMALIZATION ---
# Using the GitHub logic for feature scaling
features_mean = np.mean(train_huMoments, axis=0)
features_std = np.std(train_huMoments, axis=0)
train_huMoments = (train_huMoments - features_mean) / features_std
test_huMoments = (test_huMoments - features_mean) / features_std

# --- 5. MODEL DEFINITION (Single Neuron) ---
model = keras.models.Sequential([
    keras.layers.Dense(1, input_shape=[7], activation='sigmoid')
])

```

```

])

model.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-3),
              loss=keras.losses.BinaryCrossentropy(),
              metrics=[keras.metrics.BinaryAccuracy()])

# --- 6. BINARY LABELING ---
# Convert to: Digit 0 vs. Everything Else (Label 1)
train_labels[train_labels != 0] = 1
test_labels[test_labels != 0] = 1

# --- 7. TRAINING ---
print("Starting training...")
model.fit(train_huMoments,
          train_labels,
          batch_size=128,
          epochs=50,
          class_weight={0: 8, 1: 1},
          verbose=1)

# --- 8. EVALUATION ---
perceptron_preds = model.predict(test_huMoments)
conf_matrix = confusion_matrix(test_labels, perceptron_preds > 0.5)
cm_display = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=["Digit 0", "Not 0"])
cm_display.plot()
plt.title("Single Neuron Digit Classifier Confusion Matrix")
plt.show()

# --- 9. SAVE ---
model_path = os.path.join(KERAS_MODEL_DIR, "hdr_perceptron.h5")
model.save(model_path)
print(f"Model saved successfully at: {model_path}")

```

Results:

