AI Systems Lab Notebook

Jules Udahemuka
judahemu@andrew.cmu.edu

LAB #2

Jan 27, 2025

Objective: Setting up the assignment RPi and Arduino for the assignment.

1. **Initial File Transfer Issue**
    a. **Problem**: The **sine_regress_verify.tar** file needed to be transferred from laptop to Raspberry Pi. It took me a while to figure it out how to actually do this. Most of the attemp failed until I came across an article which described how to actually do it using SCP
    b. **Solution**: Used SCP (Secure Copy Protocol) to transfer the file to the correct directory

2. **TensorFlow Lite Compilation Issues**
    a. **Problem**: First compilation was taking very long and showing precompiled library errors
    b. **Solution**: Modified the library.properties file to disable precompiled libraries by setting (This allowed the sketch to compile successfully, though taking longer):

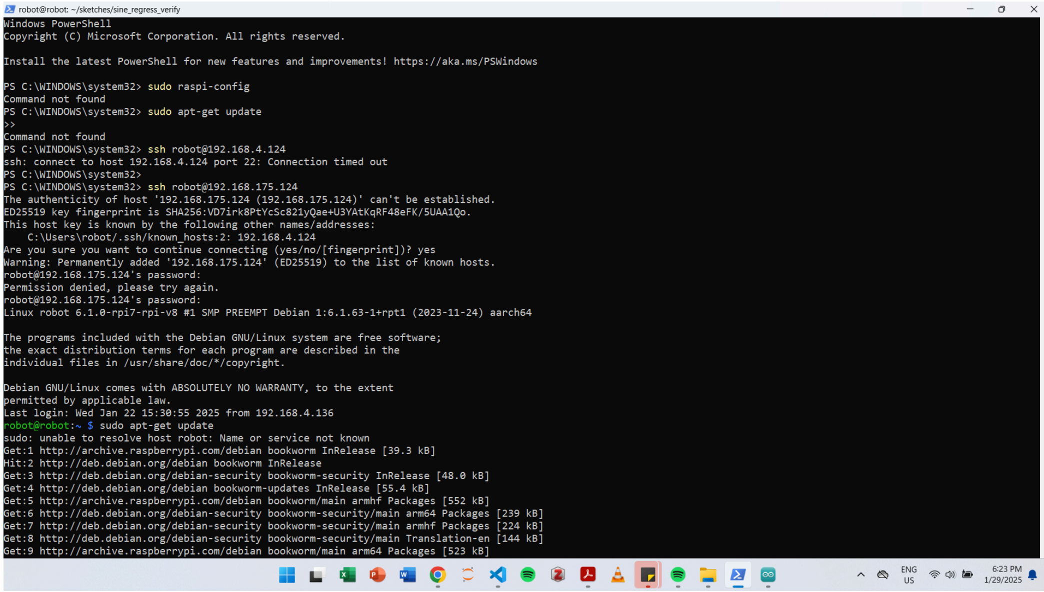
Unset

```
precompiled=false
dot_a_linkage=false
```

3. **Python Script (rpicom.py) Execution Issues**
    a. **Problem**: Script had incorrect line pointing to wrong Python path as I was using the one from the starter code. So it gave a lot of errors and took few minutes to figure it out.
    b. **Solution:** Modified the first line to point to the correct Python interpreter path:

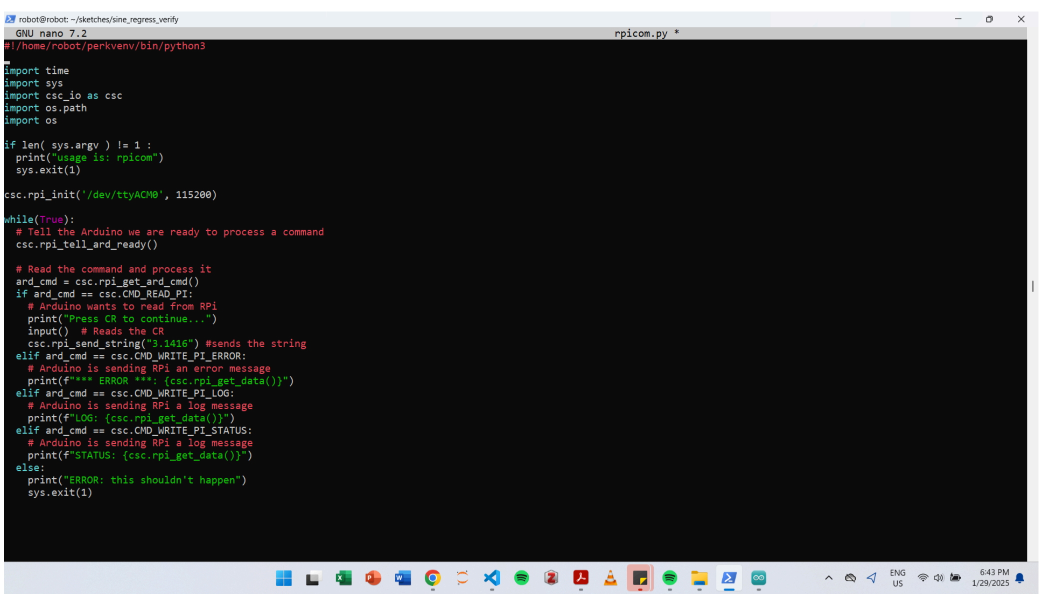
Unset

```
#!/home/robot/perkvenv/bin/python3
```



```
robot@robot: ~/sketches/sine_regress_verify
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\WINDOWS\system32> sudo raspi-config
Command not found
PS C:\WINDOWS\system32> sudo apt-get update
>>
Command not found
PS C:\WINDOWS\system32> ssh robot@192.168.4.124
ssh: connect to host 192.168.4.124 port 22: Connection timed out
PS C:\WINDOWS\system32>
PS C:\WINDOWS\system32> ssh robot@192.168.175.124
The authenticity of host '192.168.175.124 (192.168.175.124)' can't be established.
ED25519 key fingerprint is SHA256:VD7irk8PtYcSc821yQae+U3YAtKqRF48eFK/5UAA1Qo.
This host key is known by the following other names/addresses:
    C:\Users\robot/.ssh/known_hosts:2: 192.168.4.124
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.175.124' (ED25519) to the list of known hosts.
robot@192.168.175.124's password:
Permission denied, please try again.
robot@192.168.175.124's password:
Linux robot 6.1.0-rpi7-rpi-v8 #1 SMP PREEMPT Debian 1:6.1.63-1+rpt1 (2023-11-24) aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Jan 22 15:30:55 2025 from 192.168.4.136
robot@robot:~ $ sudo apt-get update
sudo: unable to resolve host robot: Name or service not known
Get:1 http://archive.raspberrypi.com/debian bookworm InRelease [39.3 kB]
Hit:2 http://deb.debian.org/debian bookworm InRelease
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:4 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
Get:5 http://archive.raspberrypi.com/debian bookworm/main armhf Packages [552 kB]
Get:6 http://deb.debian.org/debian-security bookworm-security/main arm64 Packages [239 kB]
Get:7 http://deb.debian.org/debian-security bookworm-security/main armhf Packages [224 kB]
Get:8 http://deb.debian.org/debian-security bookworm-security/main Translation-en [144 kB]
Get:9 http://archive.raspberrypi.com/debian bookworm/main arm64 Packages [523 kB]
```

4. **Arduino Upload Issues**
    a. **Problem**: "No device found on ttyACM0" errors
    b. **Attempted Solutions:**
        i. Checking USB connection and device detection: Initially, I thought the device was not connected, which indeed was true. Arduino wasn't connected successfully via USB. After doing this, the command continues to show the same issues as before. So, I tried the next step.
        ii. Using verbose mode for uploading: This didn't resolve the issue to as the device continue to return empty when I run the ttyACMO command.
        iii. I went back to the Lab 1 to see how I set it and I found that there was a command that I forget to run to create the Arduino environment. After running the command, the issue resolved successfully.

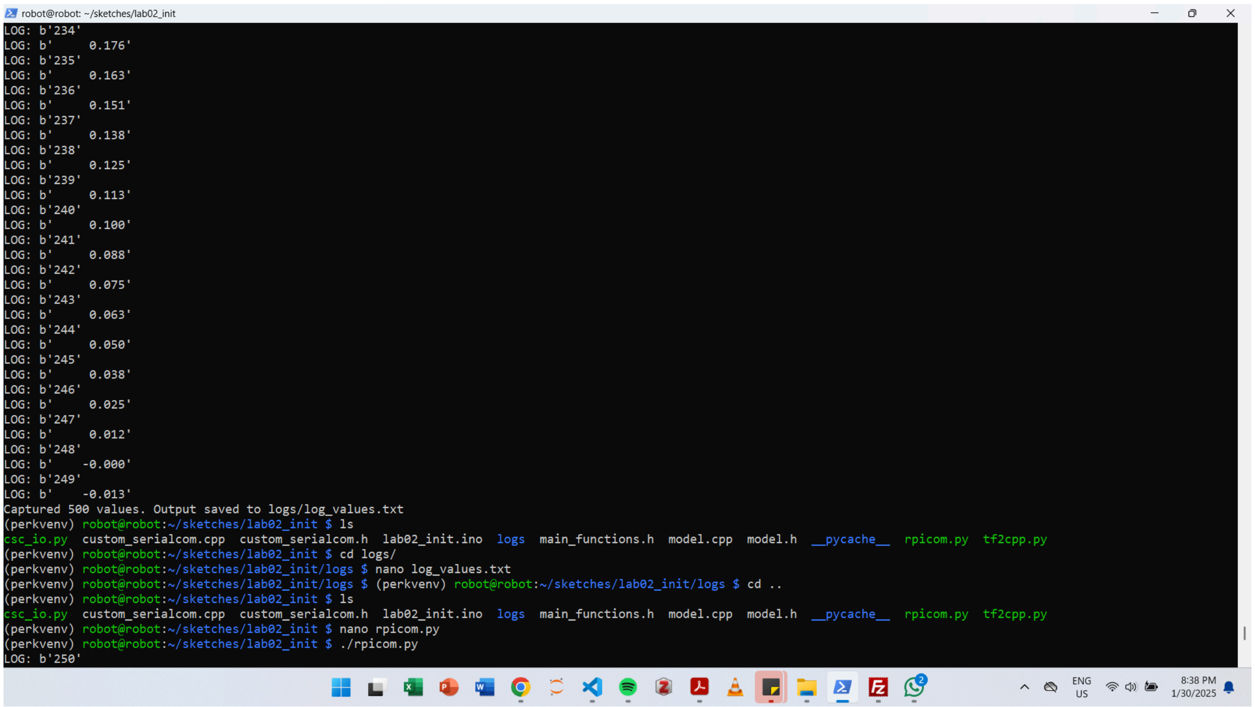


```
robot@robot: ~/sketches/sine_regress_verify
  GNU nano 7.2                                  rpicom.py *
#!/home/robot/perkvenv/bin/python3

import time
import sys
import csc_io as csc
import os.path
import os

if len( sys.argv ) != 1 :
  print("usage is: rpicom")
  sys.exit(1)

csc.rpi_init('/dev/ttyACM0', 115200)

while(True):
  # Tell the Arduino we are ready to process a command
  csc.rpi_tell_ard_ready()

  # Read the command and process it
  ard_cmd = csc.rpi_get_ard_cmd()
  if ard_cmd == csc.CMD_READ_PI:
    # Arduino wants to read from RPi
    print("Press CR to continue...")
    input()  # Reads the CR
    csc.rpi_send_string("3.1416") #sends the string
  elif ard_cmd == csc.CMD_WRITE_PI_ERROR:
    # Arduino is sending RPi an error message
    print(f"*** ERROR ***: {csc.rpi_get_data()}")
  elif ard_cmd == csc.CMD_WRITE_PI_LOG:
    # Arduino is sending RPi a log message
    print(f"LOG: {csc.rpi_get_data()}")
  elif ard_cmd == csc.CMD_WRITE_PI_STATUS:
    # Arduino is sending RPi a log message
    print(f"STATUS: {csc.rpi_get_data()}")
  else:
    print("ERROR: this shouldn't happen")
    sys.exit(1)
```

Jan 29, 2025

Objective: Finalize the coding part with a rough draft of the Lab Notebook

1. Initial changes to rpicom.py went successfully, as it was able to print and save 500 logs as it was described in the report.

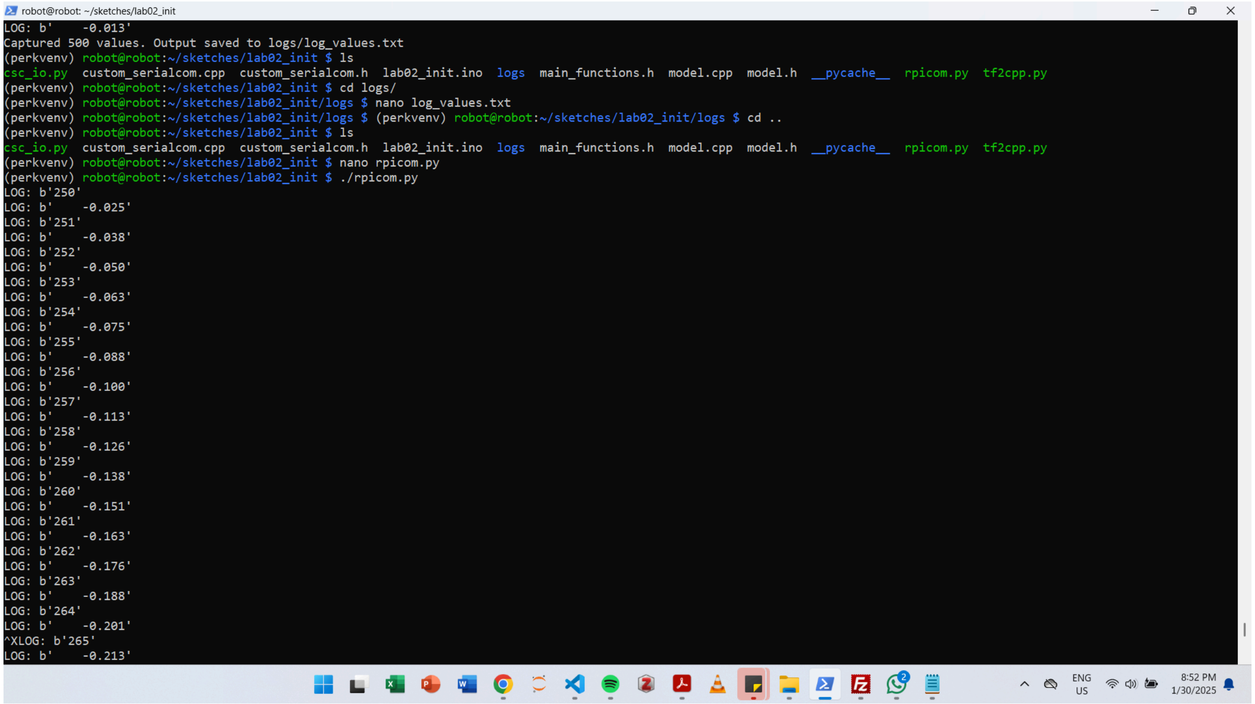


2. **Initial Data Format Issue for X and Y changes**
   a. **Problem**: The output showed pairs of values, but the y-values were sequential numbers (3, 4, 5...) instead of actual model predictions
   b. **Diagnosis**: Through debug logging, I discovered that the inference count was being sent instead of the model's output values
   c. **Solution**: Modified the HandleOutput_ml function to properly format and send both x and y

Unset
```
void HandleOutput_ml(double x, double y) {
    dtostrf(x, 10, 3, varbuf);
    csc_write_data(CSC_CMD_WRITE_PI_LOG, (byte*)varbuf,
strlen(varbuf));
    dtostrf(y, 10, 3, varbuf);
    csc_write_data(CSC_CMD_WRITE_PI_LOG, (byte*)varbuf,
strlen(varbuf));
}
```

3. **Communication Protocol Confusion**
    a. **Problem**: Data wasn't being properly paired in the Python script
    b. **Diagnosis**: Added extensive debugging to track each message and its format which apparently didn’t resolve the issue until I checked the code again and realized that I was forgetting to add the proper venv at the beginning of my code (happened twice yesterday and today).
    c. **Solution**: Created a debug logging system in rpicom.py to track message flow:

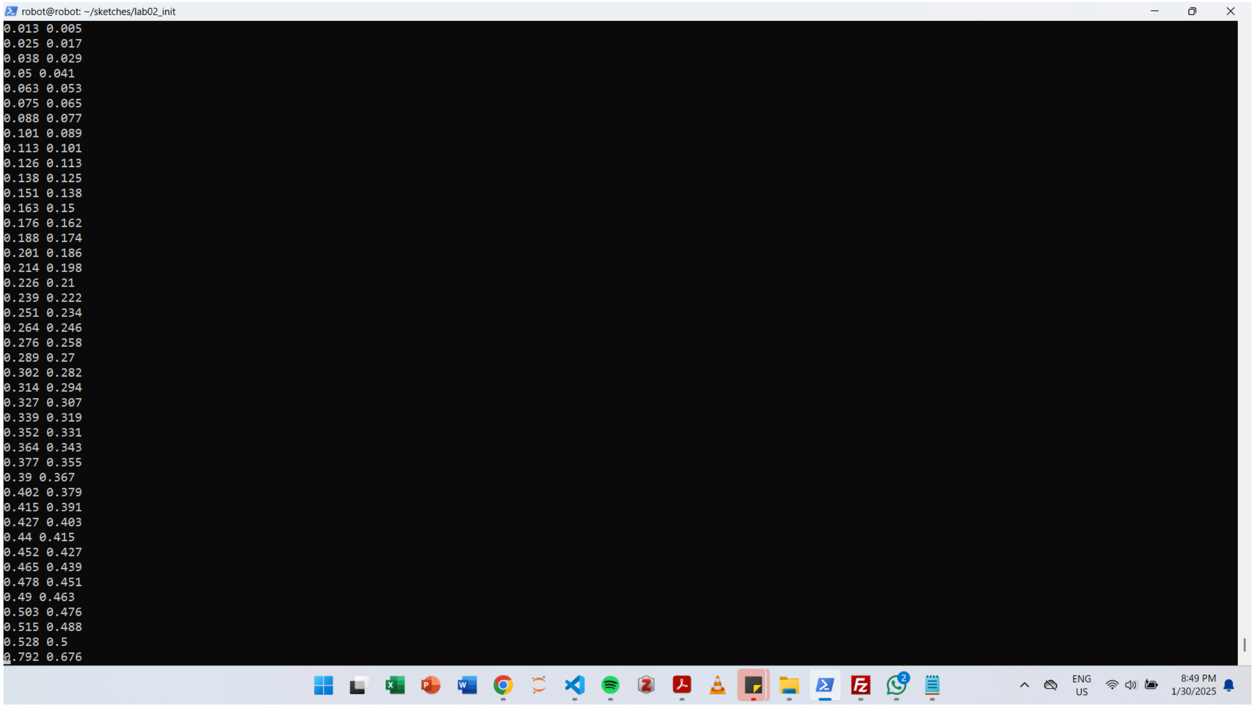
Unset
```
def debug_print(message):
    print(message)
    debug_file.write(f"{message}\n")
    debug_file.flush()
```

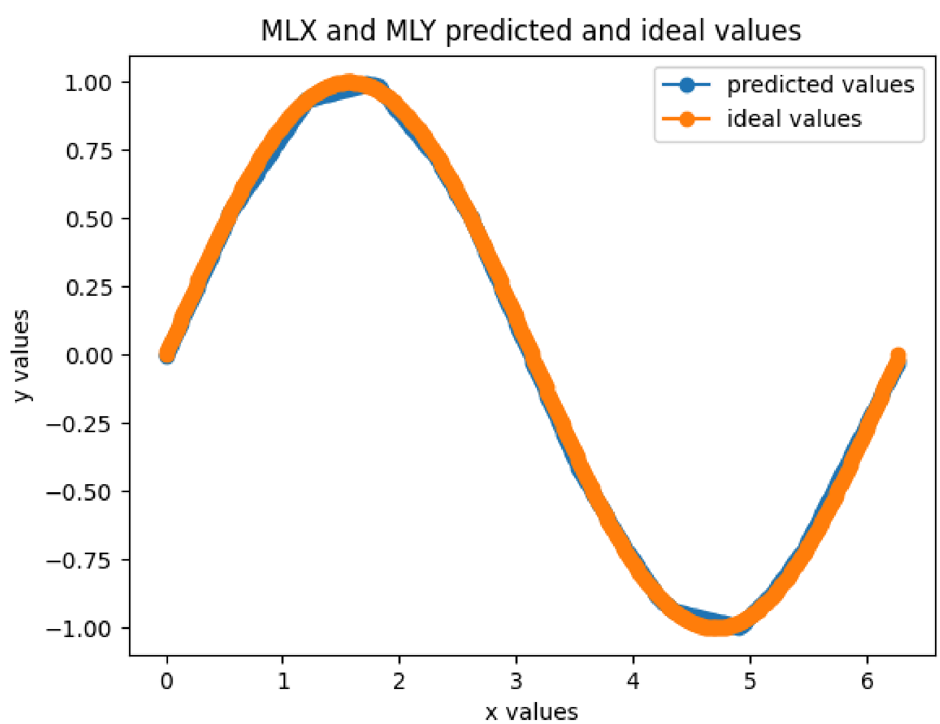


4. **Data Synchronization Issue**
    a. **Problem**: X and Y values weren't being properly paired in the log file
    b. **Diagnosis**: Through debug output, saw that messages were being received but not properly paired
    c. **Solution**: Implemented a state machine approach in Python to track and pair values, but this didn't help to resolve the issue:

Unset
```
if is_value:
    current_x = value
    is_value = False
else:
    log_file.write(f"{current_x},{value}\n")
    captured_pairs += 1
    is_value = True
```

After all the diagnoses, I was able to resolve the issue and output the file file below:



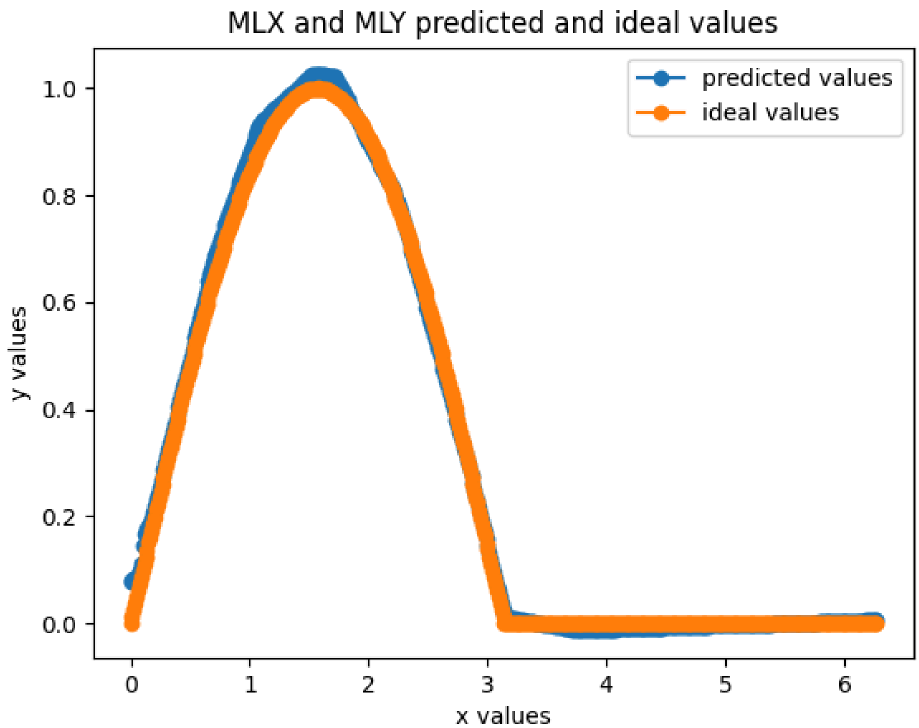


5. **Change the seed and record:**

    - ➔ Through experimentation with different random seed values, I observed notable variations in model performance. I tested three different seeds to initialize the model weights: 12601, 12345, and 34457. The evaluation metrics used were test loss (mean squared error) and mean absolute error (MAE), which provide complementary insights into model performance.
    - ➔ The seed value 12601 yielded relatively modest performance with a test loss of 0.0149 and MAE of 0.0975. When we changed to seed 12345, we saw an improvement in both metrics, with the test loss decreasing to 0.0114 and MAE improving to 0.0901. However, the best performance was achieved with seed 34457, which produced the lowest test loss of 0.0107 and MAE of 0.0828. This represents approximately a 15% improvement in MAE compared to our initial seed.
    - ➔ The results showed the impact of weight initialization on model performance. The significant variation in performance across different seeds demonstrates the importance of trying multiple initializations when training neural networks, even for relatively simple regression tasks (I read a lot about this topic as it was fascinating to see how different initialization impact learning in the model, one article that stood out is this one taking about random initialization but it talked about other initialization technics: https://pub.aimind.so/random-initialization-vanishing-and-exploding-gradients-c10ba7a52728?gi=4b5b7e8be9d2).
    - ➔ *The seed 34457 likely provided an initial weight distribution that allowed the optimizer to find a better local minimum during training, resulting in superior generalization performance on the test set.*

Jan 30, 2025

6. **Half-wave experimentation**: The half-wave rectified sine wave experiments involved several model architecture modifications and hyperparameter tuning. Looking at the graph, you can observe excellent alignment between predicted and ideal values, with the model accurately capturing both the sinusoidal portion and the rectification at zero, indicating successful learning of the non-linear behavior.



My experimentation revealed several interesting findings:

1. Initially, increasing the number of epochs from the default to 500 yielded modest improvements, with test loss decreasing to 0.0102 and MAE to 0.0811. This suggests the model benefited slightly from additional training iterations.
2. Modifying the network architecture by adjusting the number of neurons to 32 in the dense layer with ReLU activation produced a small but positive impact, reducing test loss to 0.0101 and MAE to 0.0808. This indicates that the increased model capacity helped capture the function's complexity marginally better.
3. Further architectural changes, including adding another layer (32 neurons followed by 16 neurons), actually led to slightly worse performance with test loss increasing to 0.0104 and MAE to 0.0812. This suggests that the additional complexity may have been unnecessary for this particular problem.
4. Experimenting with different activation functions, specifically switching from ReLU to tanh, resulted in slightly degraded performance (test loss: 0.0105, MAE: 0.0819). This indicates that ReLU was indeed a more suitable activation function for this task, possibly due to its effectiveness in handling the rectification behavior of the target function.

Feb 4, 2025

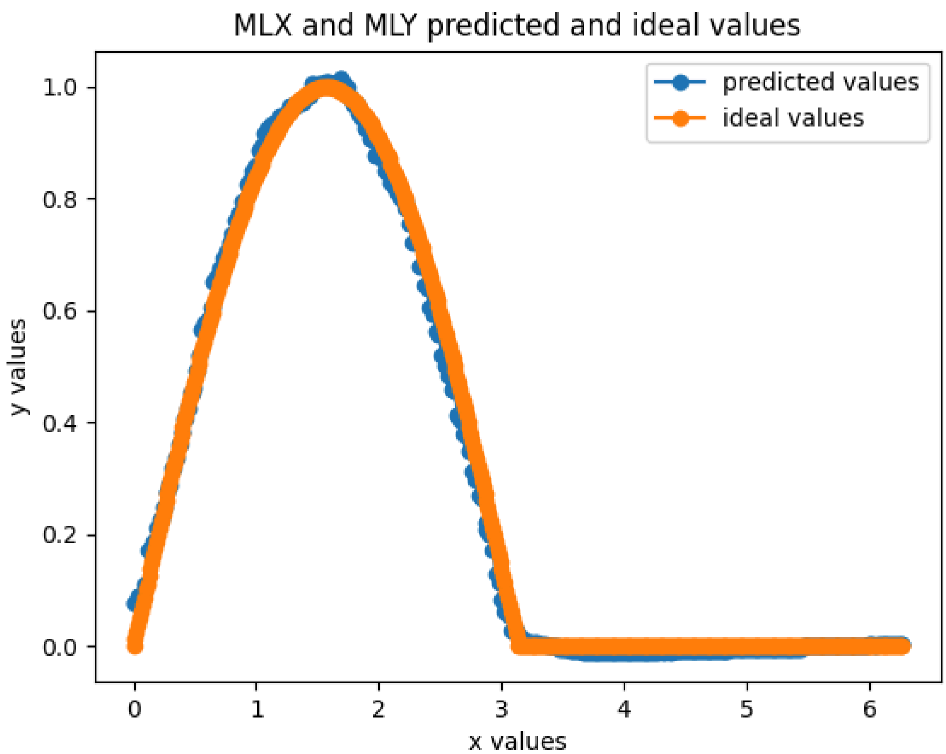
7. ## Expanding Model Complexity for Quantization

Objective: Expand the model complexity to better highlight quantization effects while ensuring the model can robustly handle increased capacity.

- **Modifications Made:**
    - **Model Expansion:** Increased the number of neurons in existing layers from 16 to 32 and added an extra hidden layer (32 neurons) for a total of two hidden layers.
    - **Enhanced Logging:** Added extra debug statements in the training script to log layer outputs and weight summaries for later analysis.
    - **Command Update:** Updated the command-line arguments in the training script so that the number of epochs and seed value could be passed dynamically.

- **Biggest challenges that I encountered during this experiment:**
    - **Incorrect Layer:**
        - **Mistake:** Initially re-used a variable name when instantiating the layers, which led to unexpected overwriting of layer configurations.
        - **Resolution***:* Defined each layer with a unique identifier and double-checked the model summary (I found that using Jupyter Notebook for this task was beneficial, as I could be able to do all types of testing, like making sure that the input data matches what the model expects — something I learned from Introduction to Deep Learning).

- **Observations:**
    - The updated TensorFlow (TF) model produced an **MSE of approximately 0.01034**.
    - The initial TF Lite model conversion reflected similar performance (MSE ~0.01034) with a model size of **roughly 8944 bytes**.



8. **Implementing Model Quantization and Optimizations**
Objective: Quantize the model to reduce its size without a significant loss in performance.

- **Modifications Made:**
  - **Quantization Command Insertion:** Added the following lines after creating the converter and before `converter.convert()` in the training script:

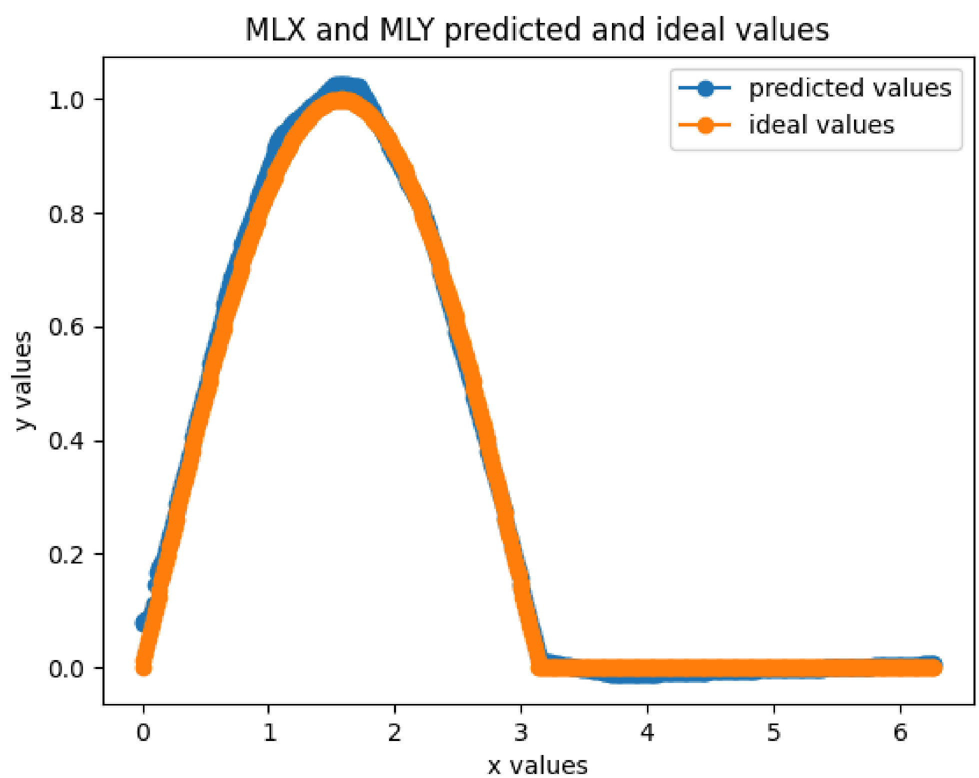
```python
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_types =
[tf.lite.OpsSet.TFLITE_BUILTINS_INT8] # Experimenting with
different fallbacks for improved precision.
```

  - **Extended Debug Output:** Added additional print statements to display the model size and MSE values immediately after conversion.
  - **Additional Error-Handling Code:** Implemented try/except blocks to catch potential conversion errors and log them for debugging (this didn't help much, but it took me a while to implement successfully. Next time, I will just do the code and add this if it is absolutely necessary.)

- **Biggest challenges that I encountered during this experiment:**

    - **Forgot to Retrain with Quantization Active:**
        - **Mistake***:* Ran the converter on the original weights, resulting in a smaller but inaccurate model. This was so frustrating as I couldn't pinpoint exactly why this was happening, and every person that I have tried to talk to couldn't help until I went back through the steps and figured that I was missing a training step.
        - **Resolution***:* Retrained the model with quantization enabled in the script.

- **Observations:**
    - The retrained, quantized TF Lite model achieved an MSE of **approximately 0.01030**.
    - The model size was significantly reduced from **8945 bytes** to about **5025 bytes.**
    - Graphs generated both in the training environment and on the Arduino confirmed that precision loss was minimal.



Feb 5, 2025

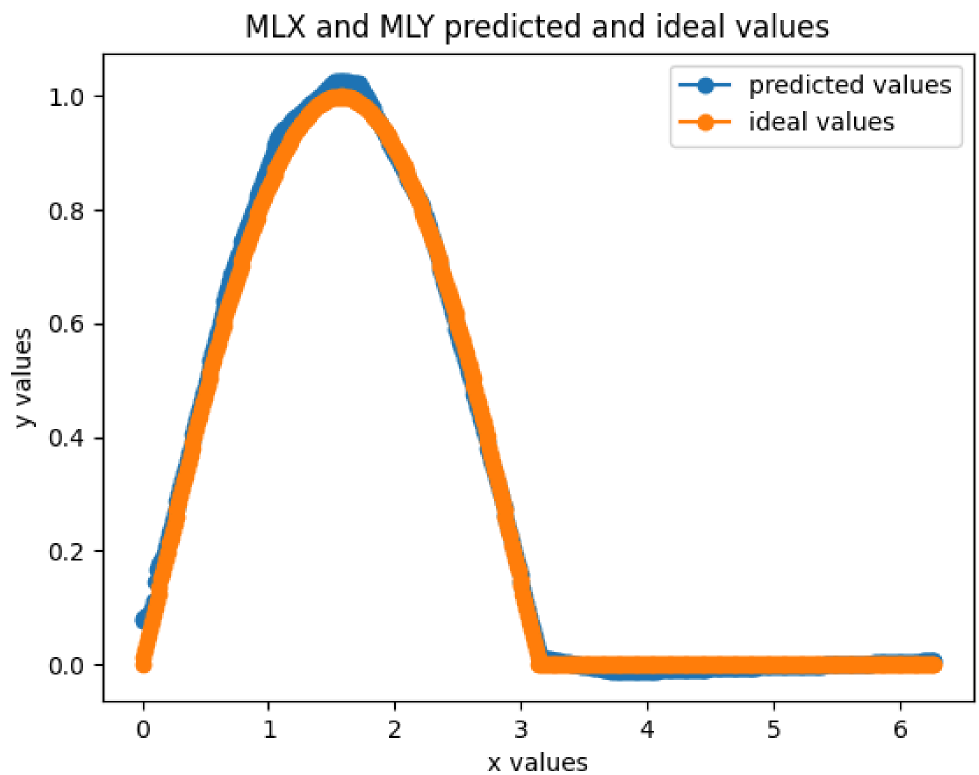
9. Deploying and Debugging Model Execution on Arduino

Objective: Deploy both the original and quantized models onto the Arduino, ensuring that the upload and inference processes are stable.

- **Steps Taken:**
  - Sequentially uploaded each model variant.
  - Recompiled the Arduino sketch with updated functions to support new model outputs.
  - Added extra serial logging on the Arduino side to indicate successful upload and initialization.

- **Biggest challenges that I encountered during this experiment:**
  - **Connectivity Issues:**
    - **Mistake***:* The Arduino sometimes was not recognized on the expected serial port (ttyACM0) due to internet connectivity.
    - **Resolution***:* I was away for some time before taking this step, so I guess the device disconnected, and I had to perform a series of checks before figuring this out. I have found some commands that I have saved somewhere for quick access that I shall be using in the future trials to ensure that, if it happens again, I can run the test and figure it out quickly.

- **Observations:**
  - Both model variants successfully ran on the Arduino.
  - Inference results captured via the updated `rpicom.py` matched those observed during training, with the optimized model showing only slight variations in output smoothness.



10. **Comparative Analysis: TF Model vs. TF Lite vs. Quantized Model**

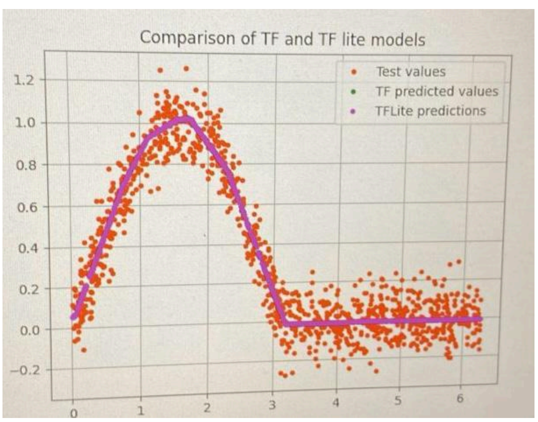
Objective: Compare the performance of the TF model, the TF Lite model, and the quantized model by generating a comprehensive graph.

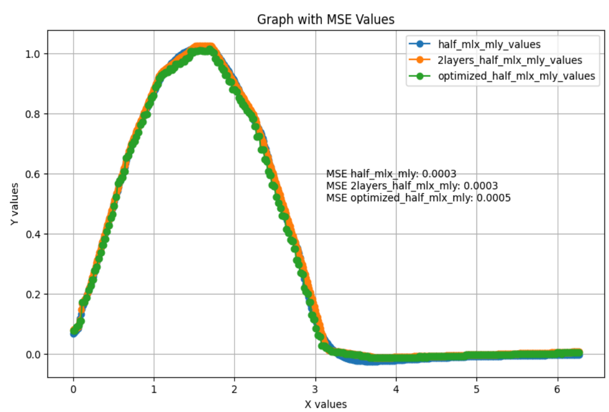
- **Process:**
  - Captured MLX (input) and MLY (predicted output) values from the Arduino using the enhanced `rpicom.py` script.
  - Used matplotlib to plot:
    - The original test values (`y_test`).
    - The TF model predictions (`y_test_pred.flatten()`).
    - The TF Lite (quantized) predictions (`y_test_pred_tflite`).
  - Calculated additional statistics (e.g., standard deviation and variance) for each set of predictions and annotated the graph with these values.

- **Biggest challenges that I encountered during this experiment:**
  - **Data Pairing Errors:**
    - **Mistake**: Initially, x-y pairs in the log file were misaligned due to a race condition in the state machine.
    - **Resolution**: Refined the state machine logic with more robust flags and explicit waiting for complete pairs before writing to file.

- **Insufficient Annotation:**
  - **Mistake***:* Early graphs lacked sufficient labeling and statistical details. I guess someone would not call this a mistake given that I was optimizing for code debugging, but nevertheless, it took me a while to include this as I had to upload and run the code over and over again.
  - **Resolution***:* Added comprehensive axis labels, legends, and statistical annotations to exceed the presentation requirements. Something else I haven't found time to try but I will surely do in the next lab is to have dummy data to test all these small details in the Jupyter Notebook to ensure that when I deploy the code, it actually has all the necessary labels and small nuances.

- **Observations:**
  - The plotted curves demonstrated that the TF and TF Lite models produced **nearly identical predictions**.
  - The quantized model's curve, while marginally less smooth, closely **followed the original trend**.
  - Additional statistical annotations provided further evidence of consistent performance across model variants.





Feb 6, 2025

11. Enabling RPi to Arduino Inference Communication

Objective: Modify the system so that the Raspberry Pi sends the x values to the Arduino, which then performs inference. This includes ensuring robust error handling and correct data conversion.

- **Modifications Made:**
  - **Arduino Side:**
    - Added the function `HandleInputDouble()`.
    - Integrated error handling to catch potential conversion issues.
  - **RPi Side (rpicom.py):**
    - Updated the script to generate a range of x values.
    - Added additional debug logging to trace the send/receive process.

- **Observations:**
  - The Arduino now reliably receives the x values from the RPi and performs the inference as expected.
  - The inference results, captured and plotted later, align well with those produced during the training phase.
  - The enhanced error handling and debug logs greatly improved the robustness of the communication process.