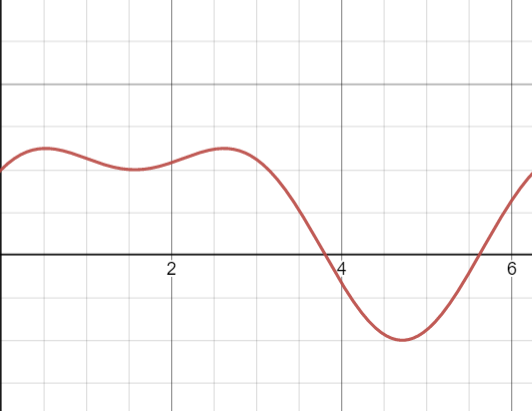
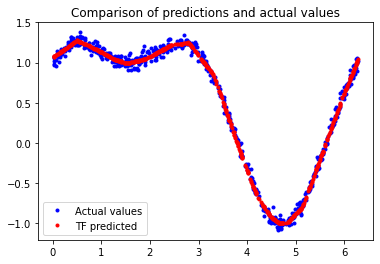
**Neural Network Approximation - Machine Learning on Arduino**

By Karam Al-Kofahi and Xingyu Bian

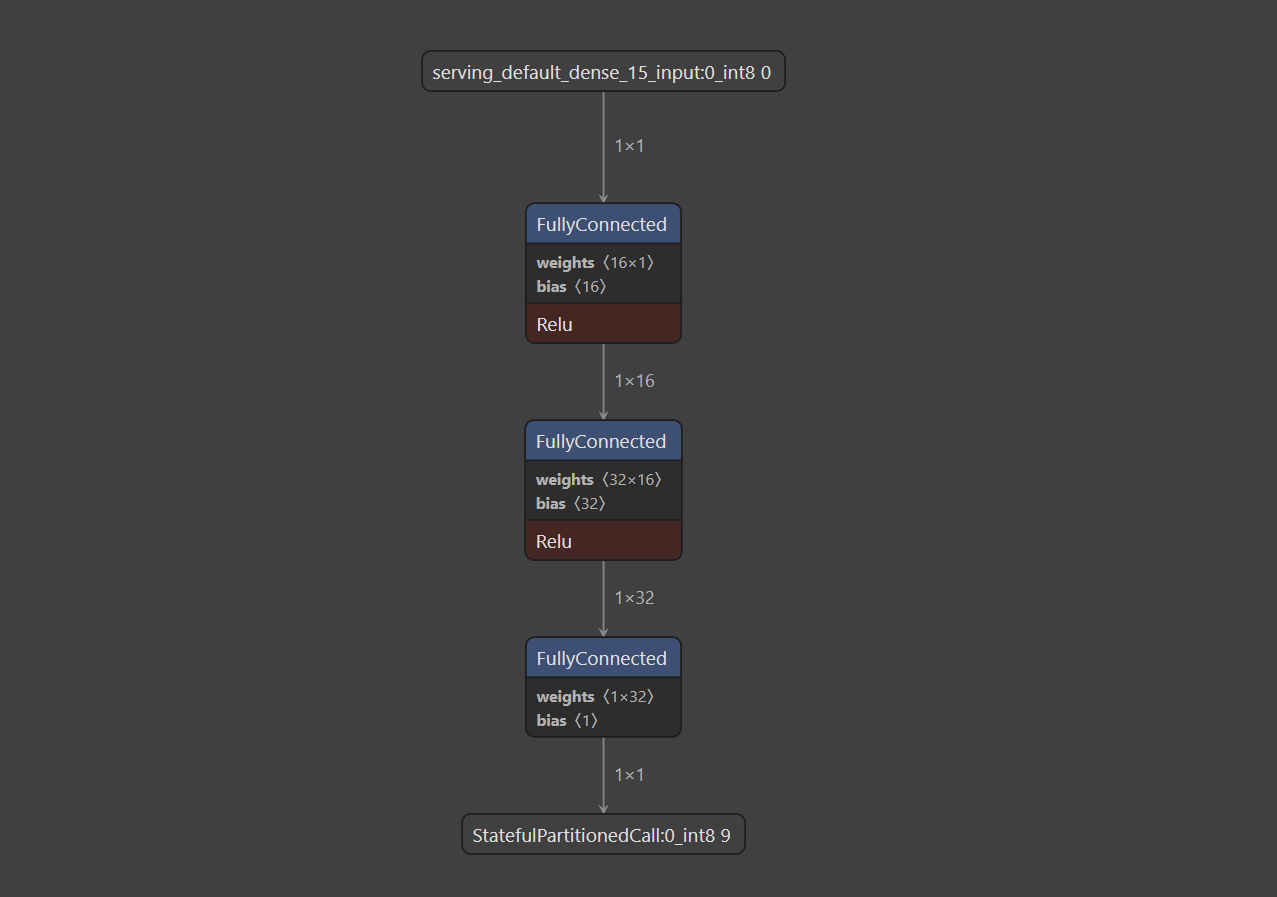
**1: Description**

The project uses TensorFlow Lite running on the microcontroller to create a trend approximation of points using machine learning. We successfully approximated sin(x)+cos2(x) using only points randomly scattered amongst that trend. To mimic real-life scenarios, we added some random noise for some variations. This demonstrates how neural networks are “universal function approximators”, being able to model complex non-linear relationships. Specifically, our model features a multilayer perceptron (MLP), consisting of 3 fully connected layers. The first layer consists of 16 neurons, the second layer consists of 32 neurons, and finally the output layer consists of a single neuron, predicting the resulting output. Please see the Netron visualization for a detailed breakdown of each component for the neural network.

Here is what the model looks like:

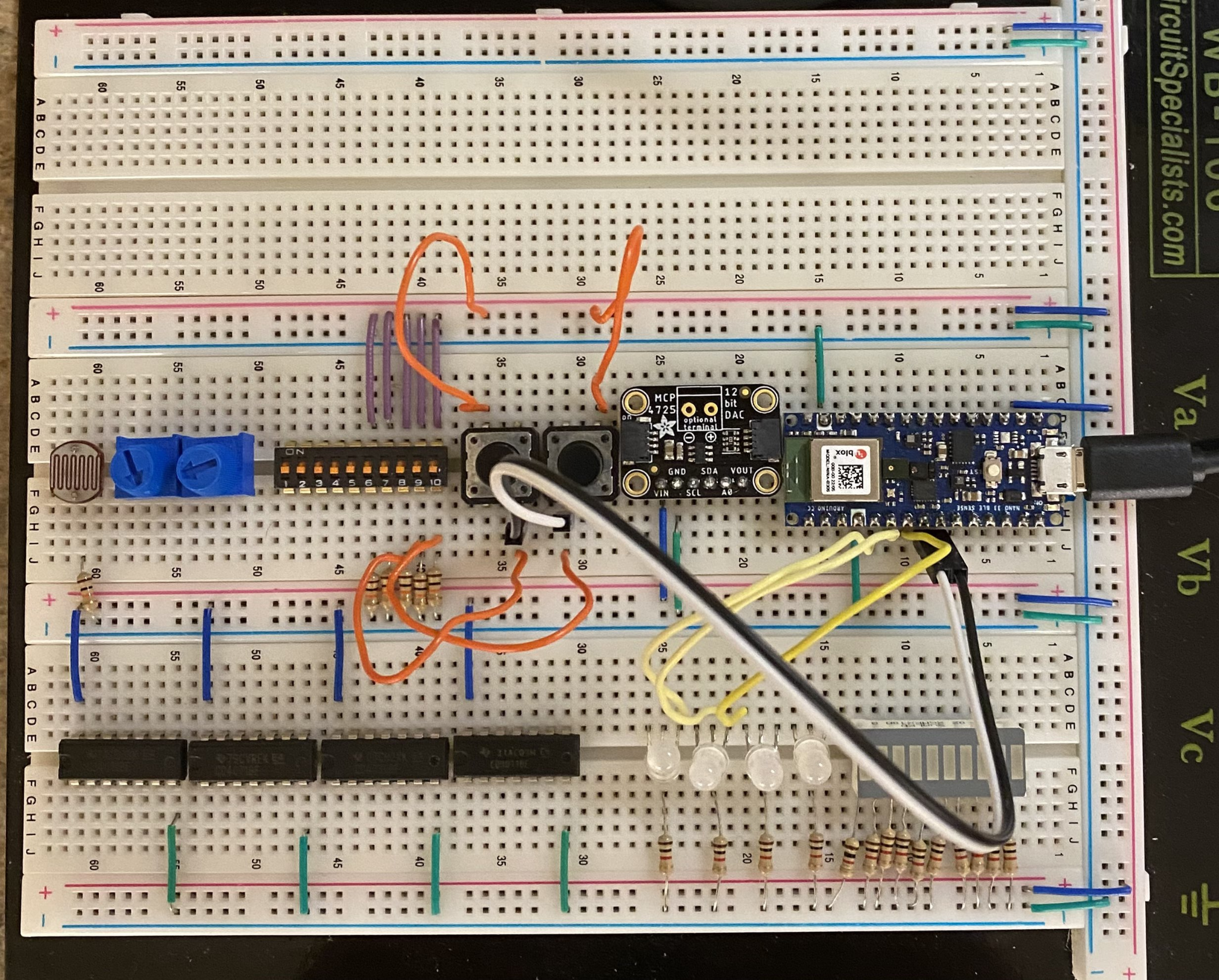


Compared to sin(x)+cos2(x) itself (desmos) on the right



Neural network architecture visualized using Netron

The red line represents the neural network approximation. We see it is able to fill gaps in the training data. The graph shown above is only one period of the function, and our code outputs live visuals to the serial plotter. The speed of the period can be adjusted using the two buttons on the board. The left button decreases the speed, while the right button increases it. The wiring looking like this:



In addition, the board also features 3 RGB LEDs, the brightness adjusted based on the neural network approximation. When the neural network predicts a high value, it will shine bright. On the other hand, if the neural network predicts a low value, it will dim.

**2: Development**

The first step is coming up with an idea: we played around with a few but decided on something that we are passionate about and presents a good exercise for this project: machine learning. The project consists of two parts: the first is written in python using the TensorFlow library. The second is written in C++ using TensorFlow Lite and Arduino.

To start, we need to train a neural network. This cannot be done on the arduino due to its limited computing resources. We utilized Google Colabs as it comes with GPU acceleration, which speeds up the training time. To get started, we first generate 3000 random points using the function sin(x)+cos2(x). We then divide the data into 60% training, 20% validation, and 20% testing. We only train the model on the training data and evaluate it on the testing and validation sets. The training process takes 500 epochs or iterations. After finding the desired model, we convert it into a TensorFlow-Lite model via quantization (reduces precision of model weights). This makes the resulting model more space efficient and possible to run on a microcontroller. Lastly, we convert the model into an unsigned char C array for compatibility with the arduino using xxd.

Thankfully, there is an official TensorFlow notebook that shows this for approximating basic functions. However, we changed the data generation and the neural network architecture for this project.

For the second part, we installed TensorFlow Lite manually as it is not available from the arduino libraries. This involves cloning a specific github repository and putting it in the arduino libraries folder. We then used the hello world example as a guide, and kept all the boiler plate code for initialization and model inference. We also added button controls and LED lighting using the knowledge we learned from the class. In addition, we also removed any files in the starter code that didn’t prove to be useful.

The wiring is pretty straight forward, the button control reads the value from D2 and D3 for speeding up and slowing down the period. As for the LED lights, they are done using PinMode() on pin 2, 3, and 4.

As for the workload, Xingyu worked on the machine learning part: including training the neural network and implementing it on the arduino. Karam implemented the button control code and is responsible for all the wirings for the project. He spent a lot of his time trying to fit the LCD screen in the project, which was unfortunately scrapped near the end as we weren’t able to make it work in the end.

Sources:

Train a Simple TensorFlow Lite for Microcontrollers notebook: <https://colab.research.google.com/github/tensorflow/tflite-micro/blob/main/tensorflow/lite/micro/examples/hello_world/train/train_hello_world_model.ipynb?authuser=0>

TensorFlow-Lite Example Hello World (included with library)

**3: What we learned**

We were surprised that machine learning could run on the nano 33BLE! We would not have guessed that a microcontroller like ours had the processing power. We further tinkered with the serial monitor for live updates, and while that was challenging, it was rewarding to have everything come together in the end. Additionally, we haven’t used a library that is not natively in the arduino software. Using TensorFlow-Lite was a great learning experience. You have to worry about memory management and pointers, which is not something you do in regular TensorFlow.

This project also involves the use of two programming languages: Python and C++. Getting everything to play well with each other is not the most straight forward, but the result did not disappoint. We converted a neural network trained using python into something that C++ understands using xxd.

One of our largest issues (and biggest source of learning) was implementing TensorFlow operations on the arduino. The installation is a bit tricky and can take a while to compile the first time. Debugging is also difficult as there are many sources of errors. Doing this project also gives you a new perspective of machine learning. In this case, we don’t have powerful systems to run anything we want. We need to utilize every Kb of RAM as efficiently as possible. This is one of the main limitations of running on a microcontroller, everything is limited.

Another challenge we faced is using an LCD screen. Although the wiring is adequate since the panel does light up, we had problems doing anything meaningful with it due to compilation errors. This is because of library issues and incompatibilities. If we had more time, we would look into it more.

Until then, we mostly experienced environments where there was more control and preliminary automatic setup for us. Having to work on some core mechanics from scratch was a great hands-on opportunity to better understand machine learning and arduino as a whole.