Intelligent Machines Workbook #3 Robotics and AI

Table of Contents

MIT Licence 4

Workbook #3 Quick Content Summary 5

Module C Unit #1: collecting data 7

The index.html file 8

The port.js file 9

Sketch C1.1 our basic Arduino sketch 12

Sketch C1.2 collecting data 13

Reformatting 16

The order doesn’t matter 17

Sketch C1.3 adjusting the data 18

Module C Unit #2: accelerometer prediction 21

The index.html file 22

The port.js file 23

Sketch C2.1 basic code 26

Sketch C2.2 mouse gesture model 27

Sketch C2.3 movement model 30

Module D Unit #1: OR and XOR 34

What you will need 35

Circuit Diagram for the button 36

The OR truth table 37

Sketch D1.1 hard coding OR truth table 38

The XOR truth table 39

Sketch D1.2 hard coding XOR truth table 40

Module D Unit #2: a neural network 42

The parameters and hyperparameters 43

Sketch D2.1 let us begin 44

Sketch D2.2 the buttons function 45

Sketch D2.3 input and target data 47

Sketch D2.4 naming key variables 49

Sketch D2.5 training is true 51

Sketch D2.6 filling the weights 53

Sketch D2.7 creating an array 56

Sketch D2.8 randomising the truth table 59

Sketch D2.9 cycle through 62

Sketch D2.10 introducing the sum 65

Sketch D2.11 the weighted sum 68

Sketch D2.12 activation function 72

Sketch D2.13 more weighted sums 76

Sketch D2.14 error 80

Sketch D2.15 delta	84
Sketch D2.16 delta to hidden	88
Sketch D2.17 updating the hidden weights	93
Sketch D2.18 change the output weights	98
Sketch D2.19 seeing the error	103
Sketch D2.20 RGB LED training finished	109
Sketch D2.21 truth table buttons	114
Sketch D2.22 inputs to hidden	120
Sketch D2.23 hidden to output	126
Sketch D2.24 output to LED	133
Module E Unit #1: the servo	**140**
What you will need	141
The breadboard power supply	142
The 9v battery and connector	143
The servo motor	144
Alternative pin connectors	145
The circuit diagram	146
Installing libraries	148
Sketch E1.1 position zero	150
Sketch E1.2 now 90 degrees	151
Sketch E1.3 finally 180 degrees	152
Sketch E1.4 repeated rotation	153
Sketch E1.5 variation on a theme	155
Module E Unit #2: slider servo	**158**
Sketch E2.1 reading a value	159
Sketch E2.2 the port sketch	160
Sketch E2.3 the main sketch	162
Sketch E2.4 attaching a value	164
Sketch E2.5 drawing the servo arm	165
Module E Unit #3: AI finger servo	**168**
Sketch E3.1 basic servo sketch	169
Sketch E3.2 index.html	170
Sketch E3.3 in the beginning	172
Sketch E3.4 getting the video	173
Sketch E3.5 the handPose model	174
Sketch E3.6 now to see	176

MIT Licence

Brief summary:

Permissions (what you can do)

Commercial use
Modification
Distribution
Private use

Limitations (what is not covered)

Liability
Warranty

Workbook #3 Quick Content Summary

This cannot be emphasised enough, I am assuming you have completed the workbooks on AI which gives you all the background information you need to understand this workbook. However, if you are pretty competent coder and have a good grasp of machine learning, and coding the Arduino then you can work through this workbook confidently.

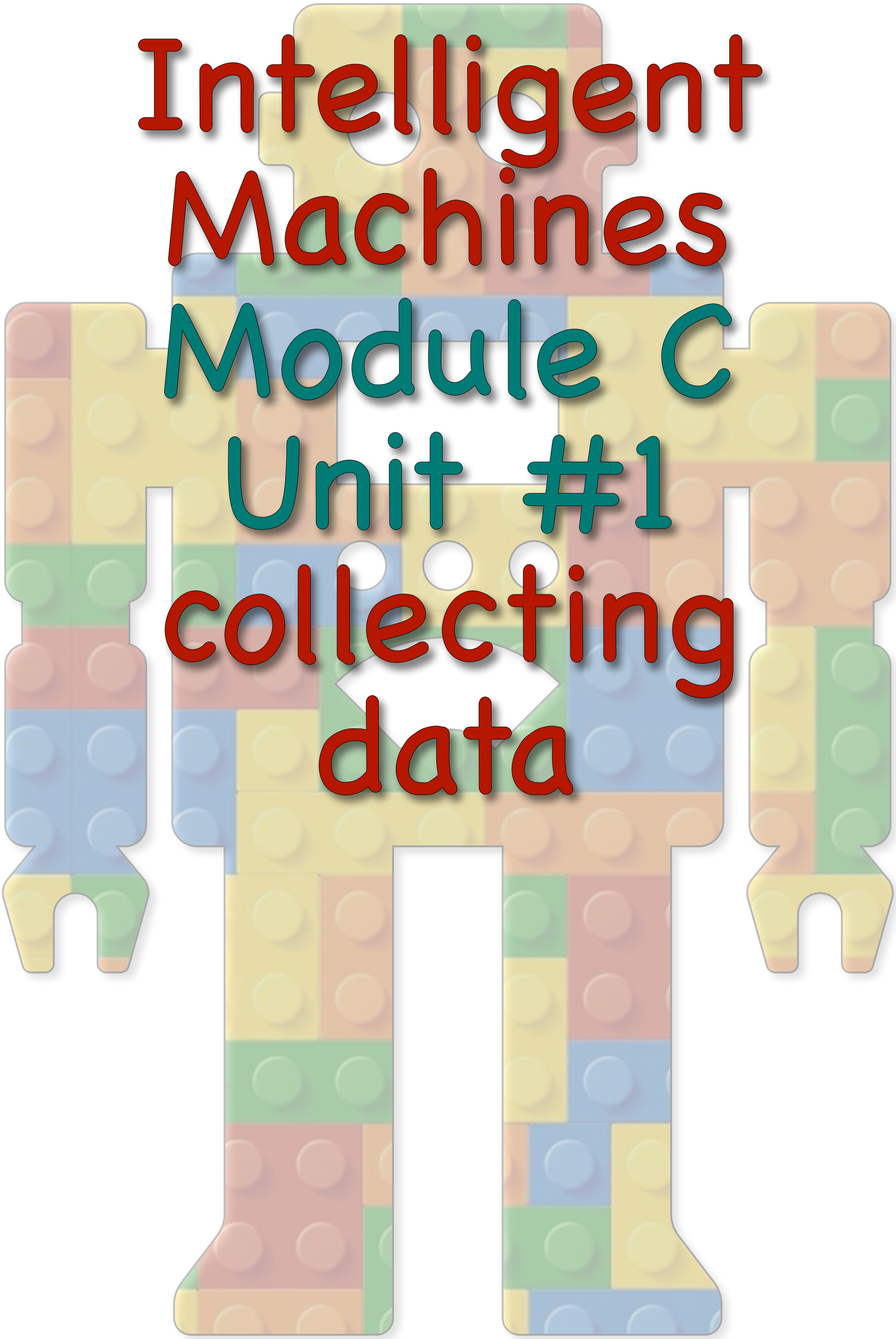
For those who are going to have a go and didn't work through either the Algorithmic Art, or the AI modules then I will give you a quick summary. We are going to create a neural network using a machine learning library called ml5.js which is going to sit inside the p5.js platform.

We are going to connect the p5.js web browser editor with the Arduino using the webserial code. We collect the necessary data from the Arduino IMU, train the model on that data based on the movement the board makes. It will then predict what movement it is making as you move the board around.

We will do something a bit extraordinary by creating a neural network straight onto the board, train it on the board and then predict. This has to be a very simple neural network. Basically an XOR response to two buttons. A lot of coding but worth it. Do need to have some understanding of neural networks.

Finally we use a pretrained model (handPose) to move a servo motor by moving your finger in front the camera. For this you will need a few more bits of equipment: a servo, power supply and 9v supply.

Intelligent Machines
Module C
Unit #1
collecting data



🤖 Module C Unit #1: collecting data

We are collecting data from the accelerometer from the built-in IMU.

To train a model, we need data. We can create our own data called `synthetic` data, but ideally, you want good-quality `real-world` data and lots of it. Collecting data and then tidying it up is very time-consuming and costly. Yet, companies do this so they can train their models on huge amounts of data.

We could do the following by creating rough synthetic data that mirrors real data, but actually, we can collect real data from the IMU and use that. We want the accelerometer data from the movement in multiple directions. We save the data into a `JSON` file. Labelling each set of data as vertical, horizontal, circular, and none.

These are the four movements (or non-movements) we require. For the sake of brevity, we will have ten bits of data in each group of movements. This is not very much at all, and you might consider increasing it to `100`, but for this simple exercise, we will just do `10`.

There are numerous ways we can do this, and I have selected one, but you might have a better or easier way of doing this.

The `index.html` file

We have the `ml5.js` library and the `port.js` file as before.

index.html

```
<!DOCTYPE html>
<html lang="en"><head>
    <script src="https://cdn.jsdelivr.net/npm/p5@2.1.1/lib/p5.js"></script>
      <script src="https://unpkg.com/p5-webserial@0.1.1/build/
p5.webserial.js"></script>
      <script src="https://unpkg.com/ml5@1/dist/ml5.min.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8">
  </head>
  <body>
    <main>
    </main>
    <script src="sketch.js"></script>
    <script src="port.js"></script>
</body></html>
```

Notes

Nothing new.

The port.js file

❗ This is our port.js, just for reference.

port.js

```
const serial = new p5.WebSerial()
let portButton
let inString
let list = []
let x = 0
let y = 0
let z

function navigation()
{
  if (!navigator.serial)
  {
    alert("WebSerial is not supported in this browser. Try Chrome")
  }
  serial.getPorts()
  serial.on("noport", makePortButton)
  serial.on("portavailable", openPort)
  serial.on("requesterror", portError)
  serial.on("data", serialEvent)
  serial.on("close", makePortButton)
}

function makePortButton()
{
  portButton = createButton("choose port")
  portButton.position(10, 10)
  portButton.mousePressed(choosePort)
}

function choosePort()
{
  if (portButton) portButton.show()
  serial.requestPort()
}
```

```
function openPort()
{
  serial.open().then(initiateSerial)
  function initiateSerial()
  {
    console.log("port open")
    if (portButton) portButton.hide()
  }
}

function portError(err)
{
  alert("Serial port error: " + err)
}

function portConnect()
{
  console.log("port connected")
  serial.getPorts()
}

function portDisconnect()
{
  serial.close()
  console.log("port disconnected")
}

function closePort()
{
  serial.close()
}

function serialEvent()
{
  inString = serial.readStringUntil("\r\n")
  if (inString)
  {
```

```
    list = splitTokens(inString, ",")
    if (list.length > 2)
    {
      x = float(list[0])
      y = float(list[1])
      z = float(list[2])
    }
  }
}
```

Notes

Just the same as before.

Sketch C1.1 our basic Arduino sketch

❗ The basic Arduino Nano sketch for the IMU.

Arduino sketch

```
#include "Arduino_BMI270_BMM150.h"
float x;
float y;
float z;

void setup()
{
  Serial.begin(9600);
  if (!IMU.begin())
  {
    Serial.println("Failed to initialise IMU");
    while (true);
  }
}

void loop()
{
  if (IMU.accelerationAvailable())
  {
    IMU.readAcceleration(x, y, z);
  }
    Serial.print(x);
    Serial.print(",");
    Serial.print(y);
    Serial.print(",");
    Serial.println(z);
}
```

Notes
Nothing new here.

Sketch C1.2 collecting data

❗ Start a new sketch.js.

We repeat this for none, horizontal, vertical, and circular. Each time we run this, you have to change the name of the movement. The line of code: label: "xxxxxx" needs to have the name of the movement inserted here.

sketch.js

```
let count = 0
let data = []
let button

function setup()
{
  createCanvas(400, 400)
  navigation()
  button = createButton('save data')
}

function draw()
{
  background(220)
  let inputs = {
    x: x,
    y: y,
    label: "xxxxxx"
  }

  data.push(inputs)
  count++
  if (count == 10)
  {
    noLoop()
  }
  button.mousePressed(saveData)
  console.log(data)
}

function saveData()
```

```
{
  save(data, 'accelerometer.json', true)
}
```

Notes

We will save the data to a file called `accelerometer.json`. You will get four sets of data. We do this for 10 bits of data in each set of movements. This is nowhere near enough, but we don't want to be spending all day training at this stage. We want to get the concept first.

Challenge

For ease of use, I have given each file the same name, but it would be better to have the file name reflect the data labelling as well.

Code Explanation

`save(data, 'accelerometer.json', true)`	This creates a json file called accelerometer

Figure 1: no movement

[{"x":-0.03,"y":0.02,"label":"none"},{"x":-0.03,"y":0.02,"label":"none"},
{"x":-0.03,"y":0.02,"label":"none"},{"x":-0.03,"y":0.02,"label":"none"},
{"x":-0.03,"y":0.01,"label":"none"},{"x":-0.03,"y":0.02,"label":"none"},
{"x":-0.03,"y":0.01,"label":"none"},{"x":-0.03,"y":0.01,"label":"none"},
{"x":-0.03,"y":0.01,"label":"none"},{"x":-0.03,"y":0.02,"label":"none"}]

Figure 2: vertical movement

[{"x":0.2,"y":0.07,"label":"vertical"},{"x":0.13,"y":-0.04,"label":"vertical"},
{"x":0.25,"y":-0.01,"label":"vertical"},{"x":0.15,"y":-0.03,"label":"vertical"},
{"x":0.04,"y":-0.02,"label":"vertical"},{"x":0.01,"y":0.02,"label":"vertical"},
{"x":-0.04,"y":0.03,"label":"vertical"},{"x":-0.02,"y":0.01,"label":"vertical"},
{"x":-0.03,"y":0,"label":"vertical"},{"x":-0.07,"y":-0.01,"label":"vertical"}]

Figure 3: circular movement

[{"x":0.19,"y":-0.29,"label":"circular"},{"x":0.16,"y":-0.37,"label":"circular"},
{"x":0.05,"y":-0.36,"label":"circular"},{"x":-0.02,"y":-0.37,"label":"circular"},
{"x":-0.11,"y":-0.45,"label":"circular"},{"x":-0.12,"y":-0.42,"label":"circular"},
{"x":-0.27,"y":-0.36,"label":"circular"},{"x":-0.32,"y":-0.31,"label":"circular"},
{"x":-0.37,"y":-0.19,"label":"circular"},{"x":-0.44,"y":-0.07,"label":"circular"}]

Figure 4: horizontal movement

[{"x":0.14,"y":0.29,"label":"horizontal"},{"x":0.1,"y":0.28,"label":"horizontal"},
{"x":0.06,"y":0.14,"label":"horizontal"},{"x":0.03,"y":-0.06,"label":"horizontal"},
{"x":-0.01,"y":-0.11,"label":"horizontal"},{"x":-0.02,"y":-0.22,"label":"horizontal"},
{"x":-0.02,"y":-0.24,"label":"horizontal"},{"x":-0.05,"y":-0.28,"label":"horizontal"},
{"x":-0.08,"y":-0.34,"label":"horizontal"},{"x":-0.13,"y":-0.48,"label":"horizontal"}]

Reformatting

We want the format to be like this, so copy the data into a document and use the find and replace function to arrange the data as it shows below. This mirrors the data format from the gesture example. Also, make sure the speech marks are the correct style when you copy this data into the Sketch; for instance, “none” won't work, it needs to be "none". You'll soon find out!

One final note, make sure that the square brackets and the commas all match up in the sketch.

```
[{x: -0.03, y: 0.02, label: "none"},
{x: -0.03, y: 0.02, label: "none"},
{x: -0.03, y: 0.02, label: "none"},
{x: -0.03, y: 0.02, label: "none"},
{x: -0.03, y: 0.01, label: "none"},
{x: -0.03, y: 0.02, label: "none"},
{x: -0.03, y: 0.01, label: "none"},
{x: -0.03, y: 0.01, label: "none"},
{x: -0.03, y: 0.01, label: "none"},
{x: -0.03, y: 0.02, label: “none”}]

[{x: 0.14, y: 0.29, label: "horizontal"},
{x: 0.10, y: 0.28, label: "horizontal"},
{x: 0.06, y: 0.14, label: "horizontal"},
{x: 0.03, y: -0.06, label: "horizontal"},
{x: -0.01, y: -0.11, label: "horizontal"},
{x: -0.02, y: -0.22, label: "horizontal"},
{x: -0.02, y: -0.24, label: "horizontal"},
{x: -0.05, y: -0.28, label: "horizontal"},
{x: -0.08, y: -0.34, label: "horizontal"},
{x: -0.13, y: -0.48, label: "horizontal"}]

[{x: 0.20, y: 0.07, label: "vertical"},
{x: 0.13, y: -0.04, label: "vertical"},
{x: 0.25, y: -0.01, label: "vertical"},
{x: 0.15, y: -0.03, label: "vertical"},
{x: 0.04, y: -0.02, label: "vertical"},
{x: 0.01, y: 0.02, label: "vertical"},
{x: -0.04, y: 0.03, label: "vertical"},
{x: -0.02, y: 0.01, label: "vertical"},
{x: -0.03, y: 0.00, label: "vertical"},
{x: -0.07, y: -0.01, label: “vertical"}]

[{x: 0.19, y: -0.29, label: "circular"},
{x: 0.16, y: -0.37, label: "circular"},
{x: 0.05, y: -0.36, label: "circular"},
{x: -0.02, y: -0.37, label: "circular"},
{x: -0.11, y: -0.45, label: "circular"},
{x: -0.12, y: -0.42, label: "circular"},
{x: -0.27, y: -0.36, label: "circular"},
{x: -0.32, y: -0.31, label: "circular"},
{x: -0.37, y: -0.19, label: "circular"},
{x: -0.44, y: -0.07, label: “circular"}]
```

The order doesn’t matter

The above becomes the following: all you need to do is copy and paste, and then do a bit of tidying.

```
[{x: -0.03, y: 0.02, label: “none"},
{x: -0.03, y: 0.02, label: "none"},
{x: -0.03, y: 0.02, label: "none"},
{x: -0.03, y: 0.02, label: "none"},
{x: -0.03, y: 0.01, label: "none"},
{x: -0.03, y: 0.02, label: "none"},
{x: -0.03, y: 0.01, label: "none"},
{x: -0.03, y: 0.01, label: "none"},
{x: -0.03, y: 0.01, label: "none"},
{x: -0.03, y: 0.02, label: “none”},
{x: 0.14, y: 0.29, label: "horizontal"},
{x: 0.10, y: 0.28, label: "horizontal"},
{x: 0.06, y: 0.14, label: "horizontal"},
{x: 0.03, y: -0.06, label: "horizontal"},
{x: -0.01, y: -0.11, label: "horizontal"},
{x: -0.02, y: -0.22, label: "horizontal"},
{x: -0.02, y: -0.24, label: "horizontal"},
{x: -0.05, y: -0.28, label: "horizontal"},
{x: -0.08, y: -0.34, label: "horizontal"},
{x: -0.13, y: -0.48, label: “horizontal"},
{x: 0.20, y: 0.07, label: "vertical"},
{x: 0.13, y: -0.04, label: "vertical"},
{x: 0.25, y: -0.01, label: "vertical"},
{x: 0.15, y: -0.03, label: "vertical"},
{x: 0.04, y: -0.02, label: "vertical"},
{x: 0.01, y: 0.02, label: "vertical"},
{x: -0.04, y: 0.03, label: "vertical"},
{x: -0.02, y: 0.01, label: "vertical"},
{x: -0.03, y: 0.00, label: "vertical"},
{x: -0.07, y: -0.01, label: “vertical”},
[{x: 0.19, y: -0.29, label: "circular"},
{x: 0.16, y: -0.37, label: "circular"},
{x: 0.05, y: -0.36, label: "circular"},
{x: -0.02, y: -0.37, label: "circular"},
{x: -0.11, y: -0.45, label: "circular"},
{x: -0.12, y: -0.42, label: "circular"},
{x: -0.27, y: -0.36, label: "circular"},
{x: -0.32, y: -0.31, label: "circular"},
{x: -0.37, y: -0.19, label: "circular"},
{x: -0.44, y: -0.07, label: “circular”}]
```

Sketch C1.3 adjusting the data

Here I have reduced the values of x and y for "none" to give better results when the Arduino isn't moving. You should have something like this.

```
let data = [
  {x: -0.02, y: 0.02, label: "none"},
  {x: -0.02, y: 0.02, label: "none"},
  {x: -0.02, y: 0.02, label: "none"},
  {x: -0.02, y: 0.02, label: "none"},
  {x: -0.02, y: 0.00, label: "none"},
  {x: -0.02, y: 0.01, label: "none"},
  {x: -0.02, y: 0.00, label: "none"},
  {x: -0.02, y: 0.00, label: "none"},
  {x: -0.02, y: 0.00, label: "none"},
  {x: -0.02, y: 0.01, label: "none"},
  {x: 0.14, y: 0.29, label: "horizontal"},
  {x: 0.10, y: 0.28, label: "horizontal"},
  {x: 0.06, y: 0.14, label: "horizontal"},
  {x: 0.03, y: -0.06, label: "horizontal"},
  {x: -0.01, y: -0.11, label: "horizontal"},
  {x: -0.02, y: -0.22, label: "horizontal"},
  {x: -0.02, y: -0.24, label: "horizontal"},
  {x: -0.05, y: -0.28, label: "horizontal"},
  {x: -0.08, y: -0.34, label: "horizontal"},
  {x: -0.13, y: -0.48, label: "horizontal"},
  {x: 0.20, y: 0.07, label: "vertical"},
  {x: 0.13, y: -0.04, label: "vertical"},
  {x: 0.25, y: -0.01, label: "vertical"},
  {x: 0.15, y: -0.03, label: "vertical"},
  {x: 0.04, y: -0.02, label: "vertical"},
  {x: 0.01, y: 0.02, label: "vertical"},
  {x: -0.04, y: 0.03, label: "vertical"},
  {x: -0.02, y: 0.01, label: "vertical"},
  {x: -0.03, y: 0, label: "vertical"},
  {x: 0.20, y: 0.07, label: "vertical"},
  {x: 0.19, y: -0.29, label: "circular"},
  {x: 0.16, y: -0.37, label: "circular"},
  {x: 0.05, y: -0.36, label: "circular"},
```

```
  {x: -0.02, y: -0.37, label: "circular"},
  {x: -0.11, y: -0.45, label: "circular"},
  {x: -0.12, y: -0.42, label: "circular"},
  {x: -0.27, y: -0.36, label: "circular"},
  {x: -0.32, y: -0.31, label: "circular"},
  {x: -0.37, y: -0.19, label: "circular"},
  {x: -0.44, y: -0.07, label: "circular"}
]
```

Notes

The above is how your data needs to look if it is to be read by the model for training purposes. Now we have our data, let's go and do something with it.

Challenge

Another solution might be to have four files that are loaded. That would be a neater solution, especially if there is a large amount of data.

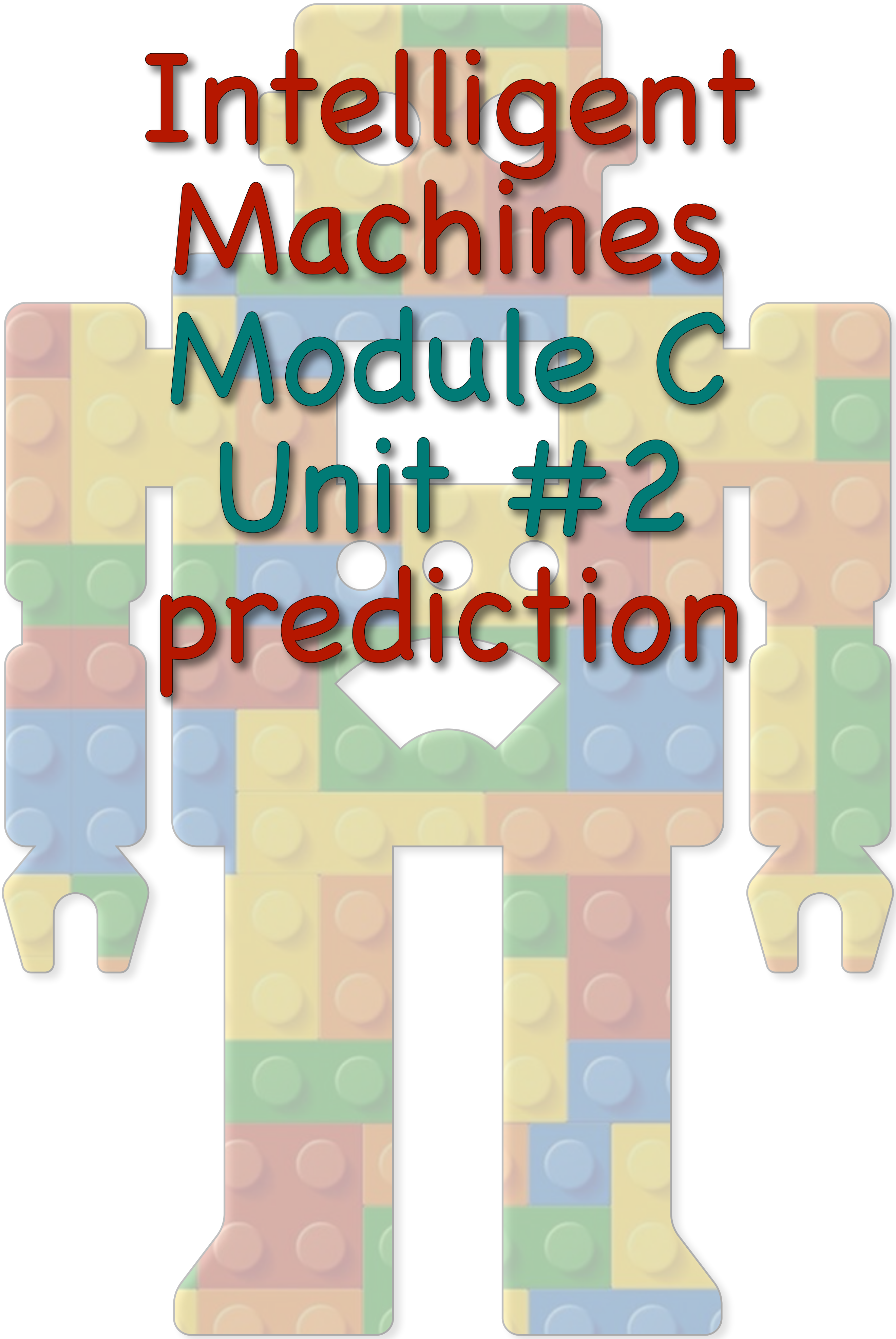
❗ please keep all this for the next unit

Intelligent Machines

Module C

Unit #2

prediction



Module C Unit #2: accelerometer prediction

Now we have the data, it is time to train the model and test it by predicting the movement you are making, whether it is none, vertical, horizontal, or circular.

We will have the predicted results appear on the canvas as you move the Arduino.

The index.html file

We have the ml5.js library and the port.js file as before.

index.html

```
<!DOCTYPE html>
<html lang="en"><head>
    <script src="https://cdn.jsdelivr.net/npm/p5@2.1.1/lib/p5.js"></script>
      <script src="https://unpkg.com/p5-webserial@0.1.1/build/
p5.webserial.js"></script>
      <script src="https://unpkg.com/ml5@1/dist/ml5.min.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8">
  </head>
  <body>
    <main>
    </main>
    <script src="sketch.js"></script>
    <script src="port.js"></script>
</body></html>
```

Notes

Nothing new.

The port.js file

❗ This is our port.js, just for reference.

port.js

```
const serial = new p5.WebSerial()
let portButton
let inString
let list = []
let x = 0
let y = 0
let z

function navigation()
{
  if (!navigator.serial)
  {
    alert("WebSerial is not supported in this browser. Try Chrome")
  }
  serial.getPorts()
  serial.on("noport", makePortButton)
  serial.on("portavailable", openPort)
  serial.on("requesterror", portError)
  serial.on("data", serialEvent)
  serial.on("close", makePortButton)
}

function makePortButton()
{
  portButton = createButton("choose port")
  portButton.position(10, 10)
  portButton.mousePressed(choosePort)
}

function choosePort()
{
  if (portButton) portButton.show()
  serial.requestPort()
}
```

```
function openPort()
{
  serial.open().then(initiateSerial)
  function initiateSerial()
  {
    console.log("port open")
    if (portButton) portButton.hide()
  }
}

function portError(err)
{
  alert("Serial port error: " + err)
}

function portConnect()
{
  console.log("port connected")
  serial.getPorts()
}

function portDisconnect()
{
  serial.close()
  console.log("port disconnected")
}

function closePort()
{
  serial.close()
}

function serialEvent()
{
  inString = serial.readStringUntil("\r\n")
  if (inString)
  {
```

```
    list = splitTokens(inString, ",")
    if (list.length > 2)
    {
      x = float(list[0])
      y = float(list[1])
      z = float(list[2])
    }
  }
}
```

Notes

Just the same as before.

Sketch C2.1 basic code

This is our bog-standard code for the Arduino.

Arduino sketch

```
#include "Arduino_BMI270_BMM150.h"
float x;
float y;
float z;

void setup()
{
  Serial.begin(9600);
  if (!IMU.begin())
  {
    Serial.println("Failed to initialise IMU");
    while (true);
  }
}

void loop()
{
  if (IMU.accelerationAvailable())
  {
    IMU.readAcceleration(x, y, z);
  }
    Serial.print(x);
    Serial.print(",");
    Serial.print(y);
    Serial.print(",");
    Serial.println(z);
}
```

Sketch C2.2 mouse gesture model

❗ In sketch.js

From the first module on AI, we have the gesture example of the classification task with the ml5.js library. The original sketch is shown below. To run this, you will also need to add the ml5.js library to the index.html file. In this example below, we used synthetic data (we made up the values).

For our Arduino IMU prediction, we need to use real data, and to do that, we need to collect it and then add it into the sketch below with some other minor changes.

❗ I have highlighted the lines of code we will not be needing (you may delete).

sketch.js

```
let nn
let status = "training"
let start
let end

let data = [
  { x: 1, y: 0.1, label: "right"},
  { x: 1, y: -0.1, label: "right"},
  { x: -1, y: 0.1, label: "left"},
  { x: -1, y: -0.1, label: "left"},
  { x: 0.1, y: 1, label: "down"},
  { x: -0.1, y: 1, label: "down"},
  { x: 0.1, y: -1, label: "up"},
  { x: -0.1, y: -1, label: "up"}
]

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    task: “classification",
    debug: false
  }
  nn = ml5.neuralNetwork(options)
  for (let item of data)
```

```
  {
    let inputs = [item.x, item.y]
    let outputs = [item.label]
    nn.addData(inputs, outputs)
  }
  nn.train({epochs: 250}, finishedTraining)
}

function finishedTraining()
{
  status = "ready"
}

function draw()
{
  background(220)
  textAlign(CENTER, CENTER)
  textSize(64)
  text(status, width/2, height/2)
  if (start && end)
  {
    strokeWeight(8)
    line(start.x, start.y, end.x, end.y)
  }
}

function mousePressed()
{
  start = createVector(mouseX, mouseY)
}

function mouseDragged()
{
  end = createVector(mouseX, mouseY)
}

function mouseReleased()
{
```

```
  let dir = p5.Vector.sub(end, start)
  dir.normalize()
  let inputs = [dir.x, dir.y]
  nn.classify(inputs, gotResults)
}

function gotResults(results)
{
  status = results[0].label
  console.log(status)
}
```

Notes

This code is something you would've come across if you had followed the AI tutorials. It predicts the movement of the mouse up, down, left, and right. We are going to use this with our new data for none, vertical, horizontal, and circular. If you want to know more in detail, then I recommend that you work through the tutorial.

Sketch C2.3 movement model

We are now ready to add the data we collected in the previous unit. This is why we reformatted the data so that it matches the data used in the gesture model.

sketch.js

```
let nn
let status = "training"

let data = [
  {x: -0.02, y: 0.02, label: "none"},
  {x: -0.02, y: 0.02, label: "none"},
  {x: -0.02, y: 0.02, label: "none"},
  {x: -0.02, y: 0.02, label: "none"},
  {x: -0.02, y: 0.00, label: "none"},
  {x: -0.02, y: 0.01, label: "none"},
  {x: -0.02, y: 0.00, label: "none"},
  {x: -0.02, y: 0.00, label: "none"},
  {x: -0.02, y: 0.00, label: "none"},
  {x: -0.02, y: 0.01, label: "none"},
  {x: 0.14, y: 0.29, label: "horizontal"},
  {x: 0.10, y: 0.28, label: "horizontal"},
  {x: 0.06, y: 0.14, label: "horizontal"},
  {x: 0.03, y: -0.06, label: "horizontal"},
  {x: -0.01, y: -0.11, label: "horizontal"},
  {x: -0.02, y: -0.22, label: "horizontal"},
  {x: -0.02, y: -0.24, label: "horizontal"},
  {x: -0.05, y: -0.28, label: "horizontal"},
  {x: -0.08, y: -0.34, label: "horizontal"},
  {x: -0.13, y: -0.48, label: "horizontal"},
  {x: 0.20, y:0.07, label: "vertical"},
  {x: 0.13, y:-0.04, label: "vertical"},
  {x: 0.25, y:-0.01, label: "vertical"},
  {x: 0.15, y:-0.03, label: "vertical"},
  {x: 0.04, y:-0.02, label: "vertical"},
  {x: 0.01, y:0.02, label: "vertical"},
  {x: -0.04, y:0.03, label: "vertical"},
  {x: -0.02, y:0.01, label: "vertical"},
  {x: -0.03, y:0, label: "vertical"},
```

```
  {x: 0.20, y:0.07, label: "vertical"},
  {x: 0.19, y: -0.29, label: "circular"},
  {x: 0.16, y: -0.37, label: "circular"},
  {x: 0.05, y: -0.36, label: "circular"},
  {x: -0.02, y: -0.37, label: "circular"},
  {x: -0.11, y: -0.45, label: "circular"},
  {x: -0.12, y: -0.42, label: "circular"},
  {x: -0.27, y: -0.36, label: "circular"},
  {x: -0.32, y: -0.31, label: "circular"},
  {x: -0.37, y: -0.19, label: "circular"},
  {x: -0.44, y: -0.07, label: "circular"}
]

function setup()
{
  createCanvas(400, 400)
  navigation()
  ml5.setBackend("webgl")
  let options = {
    task: "classification",
    debug: false
  }
  nn = ml5.neuralNetwork(options)
  for (let item of data)
  {
    let inputs = [item.x, item.y]
    let outputs = [item.label]
    nn.addData(inputs, outputs)
  }
  nn.train({epochs: 250}, finishedTraining)
}

function finishedTraining()
{
  status = "ready"
  console.log(status)
}
```

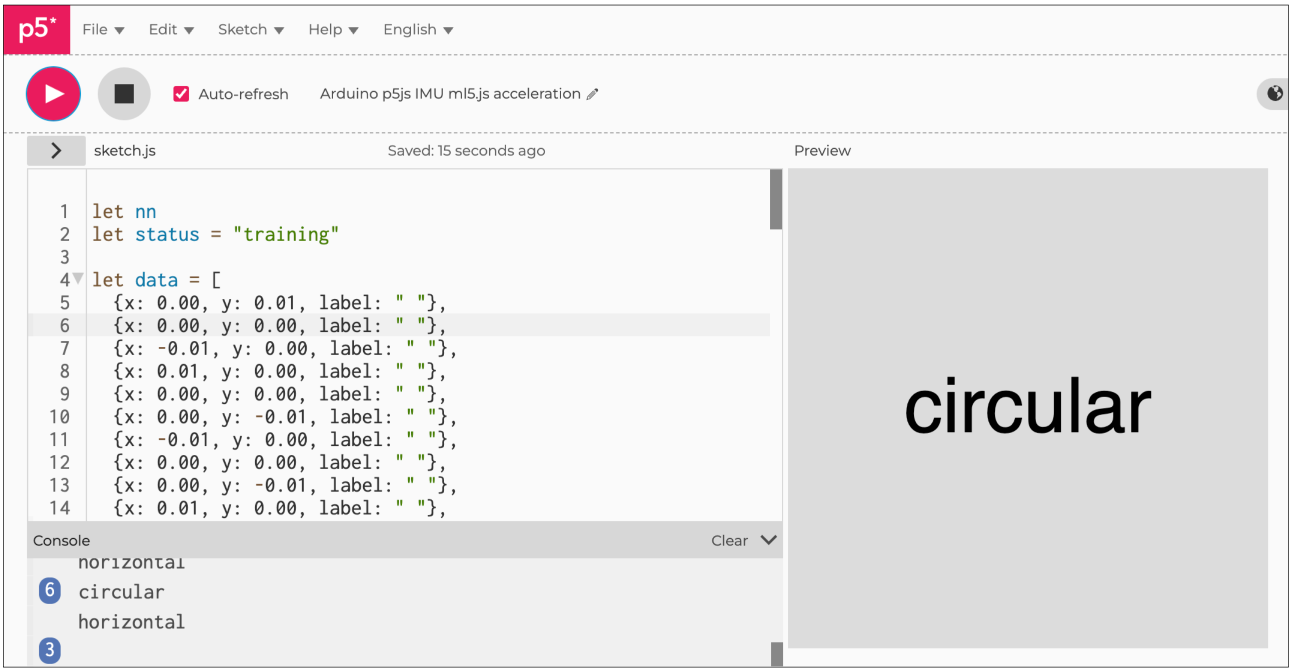
```
function draw()
{
  background(220)
  frameRate(20)
  textAlign(CENTER, CENTER)
  textSize(64)
  text(status, width/2, height/2)
  inputs = [x, y]
  nn.classify(inputs, gotResults)
}

function gotResults(results)
{
  status = results[0].label
  console.log(status)
}
```

Notes

A simpler sketch in many ways, but with more data.

Predicting the Accelerometer

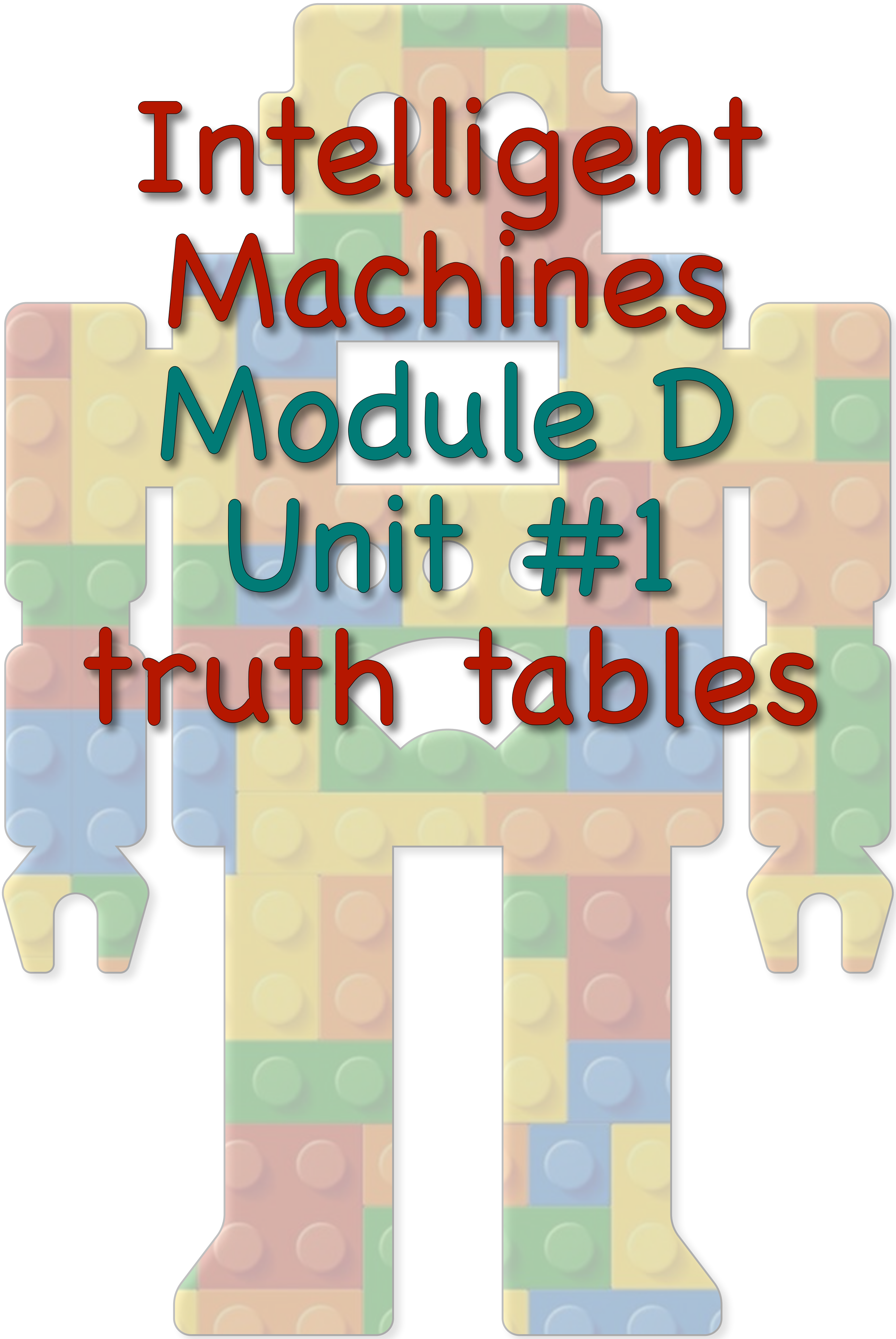


Intelligent Machines

Module D

Unit #1

truth tables



Module D Unit #1: OR and XOR

We will add a second button to our Arduino Nano 33 BLE. It will be attached to digital pin D3. We will use the same built-in LED on pin 13.

We are going to explore a very simple premise. How to use AI to control an LED with inputs from 2 buttons. We will look at two examples. The first…

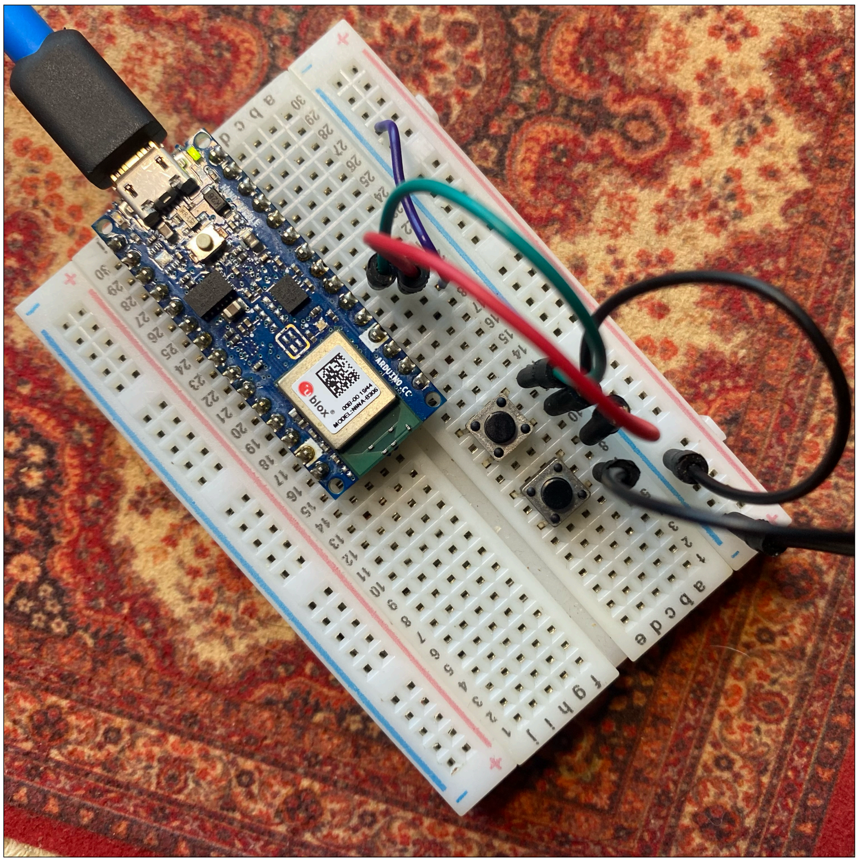
What you will need

The full list is here. Look at the image below and the wiring diagram (fig.5).

1 x Arduino Nano 33 BLE
1 x breadboard
2 x buttons
5 x male-to-male jumper leads

You could add a resistor in series with the button, but the Arduino Nano 33 BLE has a built-in resistor that you can pull up. This saves you the bother. The downside is that the logic (HIGH = off and LOW = on) is reversed.

Figure 1: component setup



Circuit Diagram for the button

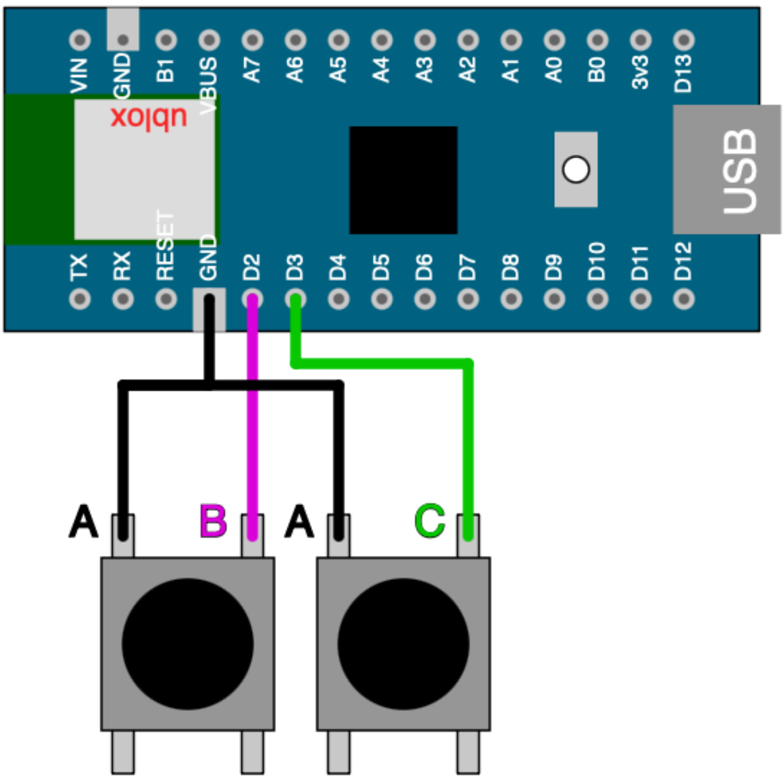
You will need two wires to connect the button to the device.

Button Pins	Arduino Pins
A *(both)*	GND
B	2 *(digital pin D2)*
C	3 *(digital pin D3)*

We are going to add the button to the breadboard. You can put it anywhere away from the board itself.

See Fig.2 below. The wiring diagram shows the connections.

Figure 2: wiring diagram



The OR truth table

A truth table is a way of representing the relationship between inputs and outputs. This helps to clarify what we expect to see. One of the simplest is the OR truth table, which has two inputs and one output. Our two inputs are button 1 and button 2, and our output is the LED, which is either on or off.

We use a simple notion for describing the state of the inputs: 1 or 0. It is quite logical: we give it 1 if the button is pressed and 0 if it is not, and we give the LED 1 if it is on and 0 when it is off. Simple.

OR Truth Table		
Button 1	Button 2	LED
0	0	0
1	0	1
0	1	1
1	1	1

Hopefully, this seems quite logical. If button 1 is pressed, the LED is on; if button 2 is pressed, the LED is also on; if both buttons are pressed, again the LED is on; if no button is pressed, the LED is off. I know I am labouring this point, but a sketch is coming to you soon. We will look at an XOR truth table later.

This is easy to hard-code; we don't really need an AI model to do this, so we will start with hard coding it anyway. We will see if we can use an AI model to do this job for us to illustrate the concept with a simple challenge.

Sketch D1.1 hard coding OR truth table

This is a simple OR configuration using two buttons (inputs) and one LED (output). The LED is on if button 1 or button 2 or both are pressed.

Arduino sketch

```
int ledPin = 13;
int buttonPin1 = 2;
int buttonPin2 = 3;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
}

void loop()
{
  buttonPin1 = digitalRead(2);
  buttonPin2 = digitalRead(3);
  if (buttonPin1 == LOW || buttonPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

Now we have solved the OR truth table with two buttons and an LED, let's look at the XOR problem.

The XOR truth table

This looks almost exactly the same, but there is one significant difference, which I have highlighted in blue. The LED only comes on if only one button is pressed, but not both.

XOR Truth Table		
Button 1	Button 2	LED
0	0	0
1	0	1
0	1	1
1	1	0

Let's go through the logic. If button 1 is pressed, the LED is on; if button 2 is pressed, the LED is also on; but if both buttons are pressed, the LED is off; if no button is pressed, the LED is also off.

This is the XOR truth table, and the difference is significant from a machine learning point of view. The OR truth table is linearly separable and could be solved (probably) by a single perceptron. But the XOR truth table needs a neural network to solve. This is also easy to hard-code, as was the OR example.

Sketch D1.2 hard coding XOR truth table

We just need to change one line of code and add in some more conditional statements.

Arduino sketch

```
int ledPin = 13;
int buttonPin1 = 2;
int buttonPin2 = 3;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
}

void loop()
{
  buttonPin1 = digitalRead(2);
  buttonPin2 = digitalRead(3);
  if ((buttonPin1 == LOW || buttonPin2 == LOW) && (!(buttonPin1 == LOW &&
buttonPin2 == LOW)))
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

This should work according to the truth table for the XOR problem.

Code Explanation

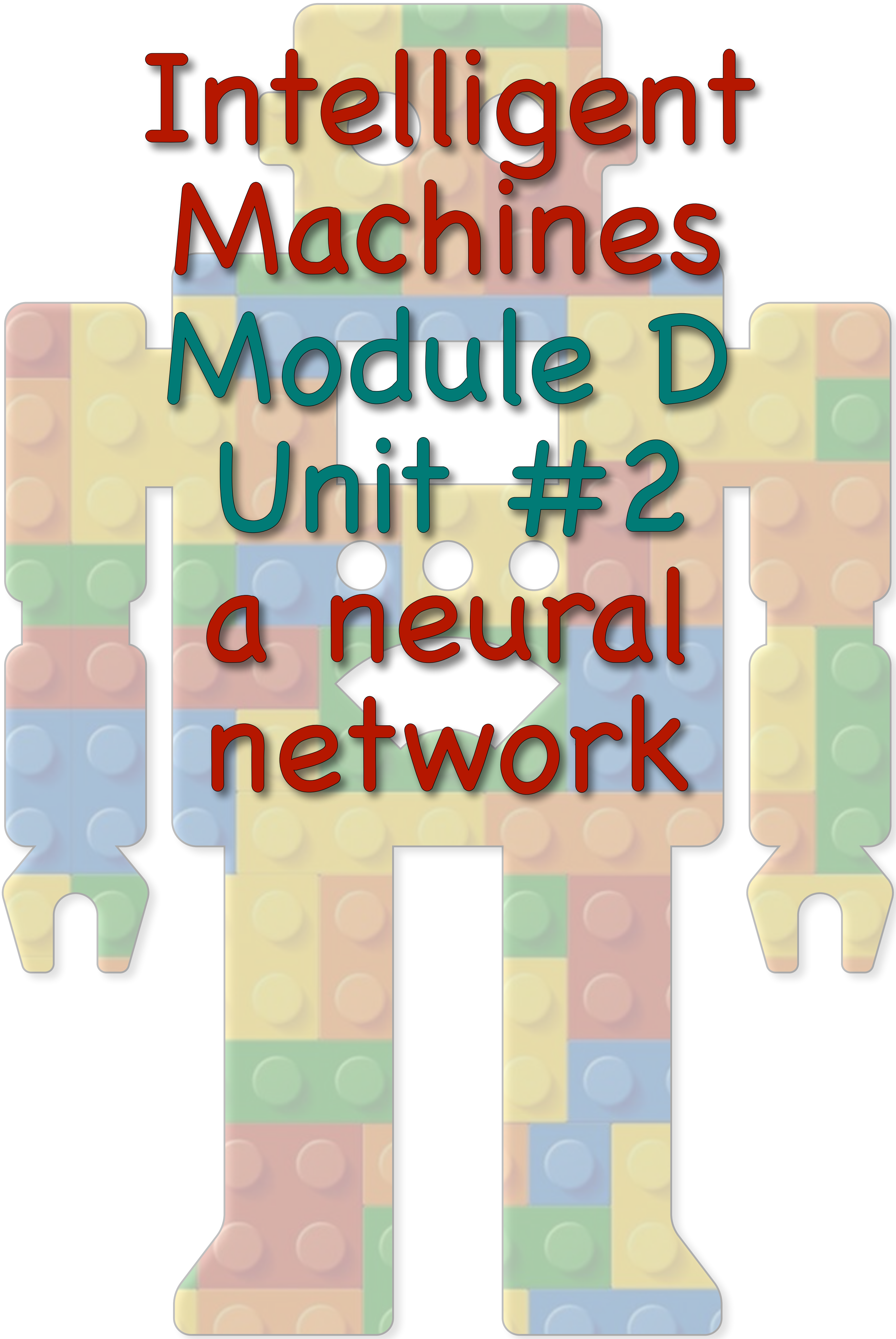
`&& (!(buttonPin1 == LOW && buttonPin2 == LOW)))`	Returns true as long as button 1 and button 2 are not pressed at the same time

Intelligent Machines

Module D

Unit #2

a neural network

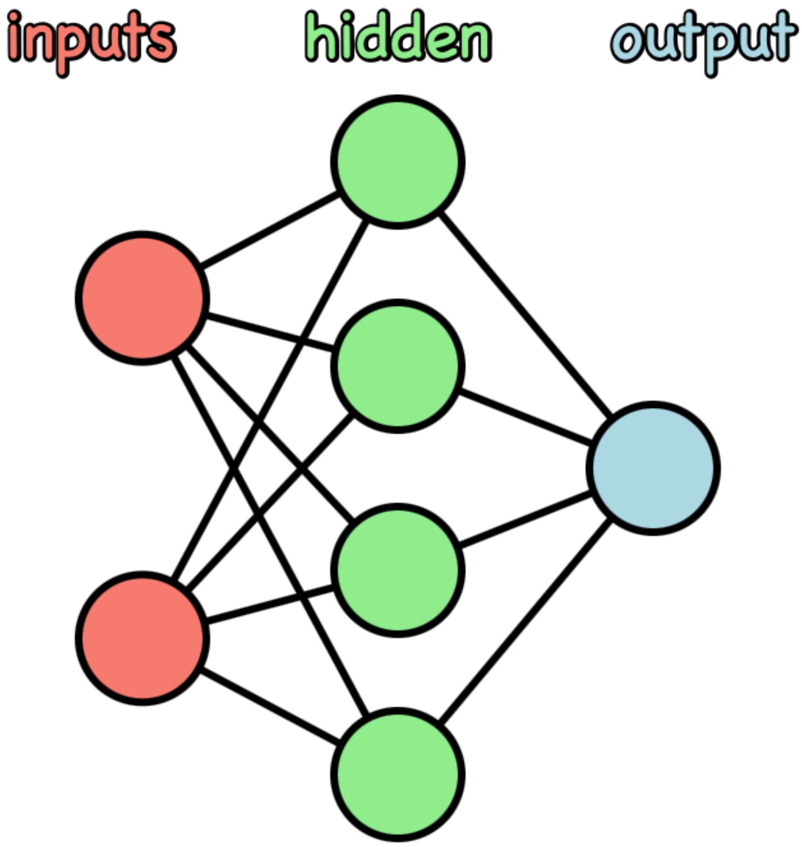


Module D Unit #2: a neural network

If you have followed along with the previous modules, you may be expecting a nice library we could use to do the work for us. The good news is there are libraries we could use; the bad news is that they can be quite complex, and for the moment, we can create our own neural network right inside the `Arduino Nano 33 BLE`.

We will build a neural network with two input neurons (nodes), button 1 and button 2, four hidden neurons (nodes), and one output neuron (node), the LED. See the diagram below:

Figure 1: neural network



The parameters and hyperparameters

Learning Rate: The learning rate determines the size of the step taken during each update. It plays a crucial role in both standard gradient descent and momentum-based optimisers.

Momentum: This controls how much of the past gradients are remembered in the current update. A value close to 1 means the optimiser will have more inertia, while a value closer to 0 means less reliance on past gradients.

Bias: Although I have drawn two input neurons, four hidden neurons, and one output neuron. We need to add an extra neuron to the input layer and an extra neuron to the hidden layer. They fire one each time just in case we have a situation where the neurons all become zero, which is definitely not what we want. The added benefit of adding bias neurons is stability.

Backpropagation: Once the data has flowed through the neural network and the error calculated, it needs to work back through the neural network, tweaking the weights so that when it goes through again, the error is less. This is called backpropagation.

Activation function: For this, we use the sigmoid function. This is perfectly fine for such a small neural network: `y = 1 / ( 1 + e^(-x))`

Error: To calculate the error, we use a form of Mean Squared Error (MSE). Which looks like: `error = 0.5 * (target - output)^2`

Weights: We have two sets of weights, those between the input and hidden neurons, and those between the hidden and output neurons. We will call them hiddenWeights and outputWeights, respectively.

Sketch D2.1 let us begin

❗ Our starting sketch.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
}

void loop()
{
  buttons();
}

void buttons()
{
  // this is where we activate the buttons
}
```

Notes

We have started with our basic sketch, plus added an extra function called `buttons()`. This is where the prediction takes place after we have trained the neural network. We use const int rather than just int because these are fixed variables (a bit contradictory, I know). We aren't going to change these, so a const will mitigate accidentally changing them.

Sketch D2.2 the buttons function

We add two more variables, inputPin1 and inputPin2, and these are changeable. We will use them to read the values on the button pins.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
}

void loop()
{
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

When you press the buttons, the LED should light up.

Sketch D2.3 input and target data

We have our truth table. We need to convert that into an array that can be used. We have four rows and two input columns, and one output column. This is a very, very small dataset. We can augment that by randomising the dataset, and later on we can add more (multiple repeats).

Here we create an input array and a target array. They are const integers as we are using 1's and 0's, and also we don't want to alter the arrays. I am using a generic neural network, so I am treating this as if it might have more than one target.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;

const int input[4][2] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[4][1] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
}
```

```
void loop()
{
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

A 4x2 array for the input and a 4x1 array for the target.

Sketch D2.4 naming key variables

Give the variables meaningful names. They are constants because they will not change, and we don't want to accidentally change them.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
}

void loop()
```

```
{
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

This is a generic neural network that will work with any set of inputs and targets.

Sketch D2.5 training is true

We want a boolean expression to indicate when training is to take place and when it is finished. We create a conditional statement in the `void loop()` function.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
}
```

```
void loop()
{
  if (training == true)
  {
    // all the training is going to happen here until...
    training = false;
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

Once training is finished, we jump out of the training loop by changing the boolean value.

Sketch D2.6 filling the weights

The weights are random values between 0 and 1 to two decimal places.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
}
```

```
void loop()
{
  if (training == true)
  {
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
        hiddenWeights[j][i] = float(random(100))/100;
      }
    }

    for (int i = 0; i < outputNodes; i++)
    {
      for (int j = 0; j <= hiddenNodes; j++)
      {
        outputWeights[j][i] = float(random(100))/100;
      }
    }
    training = false;
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

To get the kind of weights we want, we have to do it this way in C/C++.

Sketch D2.7 creating an array

We want to randomise the dataset for the truth table, but first we create an array that we can randomise and then use to jumble the data to make it easier to train. We add a global variable, a, so we don't have to keep declaring it.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
```

```
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
        hiddenWeights[j][i] = float(random(100))/100;
      }
    }

    for (int i = 0; i < outputNodes; i++)
    {
      for (int j = 0; j <= hiddenNodes; j++)
      {
        outputWeights[j][i] = float(random(100))/100;
      }
    }
    training = false;
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
```

```
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

This hasn't randomised the array just yet, just setting the scene.

Sketch D2.8 randomising the truth table

The randomised truth table will be done during the actual training process once we have initialised the weights. We will want to cycle through the randomised data a number of times because it is such a tiny dataset. So we will do this 10000 times.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};
```

```
void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
        hiddenWeights[j][i] = float(random(100))/100;
      }
    }

    for (int i = 0; i < outputNodes; i++)
    {
      for (int j = 0; j <= hiddenNodes; j++)
      {
        outputWeights[j][i] = float(random(100))/100;
      }
    }

    for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
    {
      for (a = 0; a < truthTable; a++)
      {
        b = random(truthTable);
```

```
        c = randomisedTT[a];
        randomisedTT[a] = randomisedTT[b];
        randomisedTT[b] = c;
      }
    }
    training = false;
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

We give the `random()` function a seed value drawn from a random value on `pin 4`. It means every time we run the sketch, we get a different random series of numbers.

Sketch D2.9 cycle through

We are going to cycle through each training pattern in the randomised order.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
```

```
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
        hiddenWeights[j][i] = float(random(100))/100;
      }
    }

    for (int i = 0; i < outputNodes; i++)
    {
      for (int j = 0; j <= hiddenNodes; j++)
      {
        outputWeights[j][i] = float(random(100))/100;
      }
    }

    for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
    {
      for (a = 0; a < truthTable; a++)
      {
        b = random(truthTable);
        c = randomisedTT[a];
        randomisedTT[a] = randomisedTT[b];
```

```
        randomisedTT[b] = c;
      }

      for (b = 0; b < truthTable; b++)
      {
        a = randomisedTT[b];
      }
    }
    training = false;
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

The order of the data set will change on each iteration.

Sketch D2.10 introducing the sum

We are going to compute the hidden layer activations. For this, we need to add up all the weighted sums, which also includes the bias (the extra neuron). The bias value is 1. We create a global variable for i and the sum.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;
int i;
float sum;

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
```

```
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
        hiddenWeights[j][i] = float(random(100))/100;
      }
    }

    for (int i = 0; i < outputNodes; i++)
    {
      for (int j = 0; j <= hiddenNodes; j++)
      {
        outputWeights[j][i] = float(random(100))/100;
      }
    }

    for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
    {
      for (a = 0; a < truthTable; a++)
```

```
      {
        b = random(truthTable);
        c = randomisedTT[a];
        randomisedTT[a] = randomisedTT[b];
        randomisedTT[b] = c;
      }

      for (b = 0; b < truthTable; b++)
      {
        a = randomisedTT[b];
        for (i = 0; i < hiddenNodes; i++)
        {
          sum = hiddenWeights[inputNodes][i];
        }
      }
    }
    training = false;
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

The variable a is redundant here but is used in the next sketch. Building up slowly.

Sketch D2.11 the weighted sum

Now, to add all the weights and inputs to the hidden layer, hence sum +=. For each hidden neuron, we work out the weighted sums and add them together before we pass it through the activation function.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;
int i;
float sum;
int j;

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
```

```
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
        hiddenWeights[j][i] = float(random(100))/100;
      }
    }

    for (int i = 0; i < outputNodes; i++)
    {
      for (int j = 0; j <= hiddenNodes; j++)
      {
        outputWeights[j][i] = float(random(100))/100;
      }
    }

    for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
    {
```

```
      for (a = 0; a < truthTable; a++)
      {
        b = random(truthTable);
        c = randomisedTT[a];
        randomisedTT[a] = randomisedTT[b];
        randomisedTT[b] = c;
      }

      for (b = 0; b < truthTable; b++)
      {
        a = randomisedTT[b];
        for (i = 0; i < hiddenNodes; i++)
        {
          sum = hiddenWeights[inputNodes][i];
          for (j = 0; j < inputNodes; j++)
          {
            sum += input[a][j] * hiddenWeights[j][i];
          }
        }
      }
    }
    training = false;
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
```

```
}
```

Notes

We are accumulating the weighted sums for each neuron.

Sketch D2.12 activation function

Once all the weighted sums have been added to each neuron, then it needs to pass through the activation function, which is the sigmoid function.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;
int i;
float sum;
int j;
float hidden[hiddenNodes];

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
```

```
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
        hiddenWeights[j][i] = float(random(100))/100;
      }
    }

    for (int i = 0; i < outputNodes; i++)
    {
      for (int j = 0; j <= hiddenNodes; j++)
      {
        outputWeights[j][i] = float(random(100))/100;
      }
    }

    for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
    {
```

```
      for (a = 0; a < truthTable; a++)
      {
        b = random(truthTable);
        c = randomisedTT[a];
        randomisedTT[a] = randomisedTT[b];
        randomisedTT[b] = c;
      }

      for (b = 0; b < truthTable; b++)
      {
        a = randomisedTT[b];
        for (i = 0; i < hiddenNodes; i++)
        {
          sum = hiddenWeights[inputNodes][i];
          for (j = 0; j < inputNodes; j++)
          {
            sum += input[a][j] * hiddenWeights[j][i];
          }
          hidden[i] = 1 / (1 + exp(-sum));
        }
      }
    }
    training = false;
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
```

```
  }
}
```

Notes

We could use another `activation function`.

Sketch D2.13 more weighted sums

Next, we calculate the weighted sums from the hidden layer to the output layer and pass the sum through the activation function (sigmoid).

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;
int i;
float sum;
int j;
float hidden[hiddenNodes];
float output[outputNodes];

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
```

```
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
        hiddenWeights[j][i] = float(random(100))/100;
      }
    }

    for (int i = 0; i < outputNodes; i++)
    {
      for (int j = 0; j <= hiddenNodes; j++)
      {
        outputWeights[j][i] = float(random(100))/100;
      }
    }

    for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
```

```
    {
      for (a = 0; a < truthTable; a++)
      {
        b = random(truthTable);
        c = randomisedTT[a];
        randomisedTT[a] = randomisedTT[b];
        randomisedTT[b] = c;
      }

      for (b = 0; b < truthTable; b++)
      {
        a = randomisedTT[b];
        for (i = 0; i < hiddenNodes; i++)
        {
          sum = hiddenWeights[inputNodes][i];
          for (j = 0; j < inputNodes; j++)
          {
            sum += input[a][j] * hiddenWeights[j][i];
          }
          hidden[i] = 1 / (1 + exp(-sum));
        }

        for (i = 0; i < outputNodes; i++)
        {
          sum = outputWeights[hiddenNodes][i];
          for (j = 0; j < hiddenNodes; j++)
          {
            sum += hidden[j] * outputWeights[j][i];
          }
          output[i] = 1 / (1 + exp(-sum));
        }
      }
    }
    training = false;
  }
  buttons();
}
```

```
void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Sketch D2.14 error

We now have an output from our neural network. We have our target values and we can compare them. The difference will be our error; we will use MSE; the code is highlighted below. We initialised the error to zero before we iterate through the training.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;
int i;
float sum;
int j;
float hidden[hiddenNodes];
float output[outputNodes];
float error;

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};
```

```
const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
        hiddenWeights[j][i] = float(random(100))/100;
      }
    }

    for (int i = 0; i < outputNodes; i++)
    {
      for (int j = 0; j <= hiddenNodes; j++)
      {
        outputWeights[j][i] = float(random(100))/100;
      }
```

```
    }

    for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
    {
      for (a = 0; a < truthTable; a++)
      {
        b = random(truthTable);
        c = randomisedTT[a];
        randomisedTT[a] = randomisedTT[b];
        randomisedTT[b] = c;
      }
      error = 0;

      for (b = 0; b < truthTable; b++)
      {
        a = randomisedTT[b];
        for (i = 0; i < hiddenNodes; i++)
        {
          sum = hiddenWeights[inputNodes][i];
          for (j = 0; j < inputNodes; j++)
          {
            sum += input[a][j] * hiddenWeights[j][i];
          }
          hidden[i] = 1 / (1 + exp(-sum));
        }

        for (i = 0; i < outputNodes; i++)
        {
          sum = outputWeights[hiddenNodes][i];
          for (j = 0; j < hiddenNodes; j++)
          {
            sum += hidden[j] * outputWeights[j][i];
          }
          output[i] = 1 / (1 + exp(-sum));
          error += 0.5 * (target[a][i] - output[i]) * (target[a][i] -
output[i]);
        }
      }
    }
```

```
    training = false;
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

We are still at the `feedforward` stage of the process and are not changing the `weights`. All we are doing is calculating the `error` at this stage.

Sketch D2.15 delta

Now we need to go back through the neural network and tweak the weights. First off are the weights between the output and the hidden layer. The question is by how much. So we calculate the delta error for the output. The equation is `∂ (delta) = (target − actual) ∗ actual ∗ (1 − actual)`. We can do this easily for the `output` as we have the `target` and `actual`.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;
int i;
float sum;
int j;
float hidden[hiddenNodes];
float output[outputNodes];
float error;
float outputDelta[outputNodes];

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};
```

```
const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
        hiddenWeights[j][i] = float(random(100))/100;
      }
    }

    for (int i = 0; i < outputNodes; i++)
    {
      for (int j = 0; j <= hiddenNodes; j++)
      {
        outputWeights[j][i] = float(random(100))/100;
```

```
      }
    }

    for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
    {
      for (a = 0; a < truthTable; a++)
      {
        b = random(truthTable);
        c = randomisedTT[a];
        randomisedTT[a] = randomisedTT[b];
        randomisedTT[b] = c;
      }
      error = 0;

      for (b = 0; b < truthTable; b++)
      {
        a = randomisedTT[b];
        for (i = 0; i < hiddenNodes; i++)
        {
          sum = hiddenWeights[inputNodes][i];
          for (j = 0; j < inputNodes; j++)
          {
            sum += input[a][j] * hiddenWeights[j][i];
          }
          hidden[i] = 1 / (1 + exp(-sum));
        }

        for (i = 0; i < outputNodes; i++)
        {
          sum = outputWeights[hiddenNodes][i];
          for (j = 0; j < hiddenNodes; j++)
          {
            sum += hidden[j] * outputWeights[j][i];
          }
          output[i] = 1 / (1 + exp(-sum));
          outputDelta[i] = (target[a][i] - output[i]) * output[i] * (1 -
output[i]);
          error += 0.5 * (target[a][i] - output[i]) * (target[a][i] -
output[i]);
```

```
        }
      }
    }
    training = false;
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

The first step in gradient descent is to calculate a value called the delta for each neuron. The delta reflects the magnitude of the error; the greater the difference between the target value for the neuron and its actual output, the smaller the change `delta = (target - output) * output * (1 - output)`.

Sketch D2.16 delta to hidden

Here we will calculate the hidden delta, which is a bit more tricky as we have no target to compare with. We keep a running total (`sum`) of the `output weights` and `output delta` (which we have already calculated).

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;
int i;
float sum;
int j;
float hidden[hiddenNodes];
float output[outputNodes];
float error;
float outputDelta[outputNodes];
float hiddenDelta[hiddenNodes];

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};
```

```
const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
        hiddenWeights[j][i] = float(random(100))/100;
      }
    }

    for (int i = 0; i < outputNodes; i++)
    {
      for (int j = 0; j <= hiddenNodes; j++)
      {
        outputWeights[j][i] = float(random(100))/100;
```

```
      }
    }

    for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
    {
      for (a = 0; a < truthTable; a++)
      {
        b = random(truthTable);
        c = randomisedTT[a];
        randomisedTT[a] = randomisedTT[b];
        randomisedTT[b] = c;
      }
      error = 0;

      for (b = 0; b < truthTable; b++)
      {
        a = randomisedTT[b];
        for (i = 0; i < hiddenNodes; i++)
        {
          sum = hiddenWeights[inputNodes][i];
          for (j = 0; j < inputNodes; j++)
          {
            sum += input[a][j] * hiddenWeights[j][i];
          }
          hidden[i] = 1 / (1 + exp(-sum));
        }

        for (i = 0; i < outputNodes; i++)
        {
          sum = outputWeights[hiddenNodes][i];
          for (j = 0; j < hiddenNodes; j++)
          {
            sum += hidden[j] * outputWeights[j][i];
          }
          output[i] = 1 / (1 + exp(-sum));
          outputDelta[i] = (target[a][i] - output[i]) * output[i] * (1 -
output[i]);
          error += 0.5 * (target[a][i] - output[i]) * (target[a][i] -
output[i]);
```

```
      }

      for (i = 0; i < hiddenNodes; i++)
      {
        sum = 0;
        for (j = 0; j < outputNodes; j++)
        {
          sum += outputWeights[i][j] * outputDelta[j];
        }
        hiddenDelta[i] = sum * hidden[i] * (1 - hidden[i]);
      }
    }
  }
  training = false;
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

We now have the hidden as well as the output delta difference or error. We want to update the weights between the input through to the output layer.

Calculating the delta at the hidden layer becomes slightly more involved as there is no target to measure against. Instead, the magnitude of the error for each hidden neuron is derived from the relationship between the weights and the delta that was calculated for the output layer. For each hidden neuron, the code steps through all of the output connections, multiplying the weights by the deltas and keeping a running total: `sum += output weights * output delta`.

Sketch D2.17 updating the hidden weights

Update the input to hidden weights. We introduce the learning rate so that we can manage how much we change these weights. Also, add in the momentum; otherwise, it may well get stuck on some local minimum.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;
int i;
float sum;
int j;
float hidden[hiddenNodes];
float output[outputNodes];
float error;
float outputDelta[outputNodes];
float hiddenDelta[hiddenNodes];
float changeHiddenWeights[inputNodes + 1][hiddenNodes];
float learningRate = 0.5;
float momentum = 0.9;

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
```

```
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
        hiddenWeights[j][i] = float(random(100))/100;
      }
    }

    for (int i = 0; i < outputNodes; i++)
    {
```

```
    for (int j = 0; j <= hiddenNodes; j++)
    {
      outputWeights[j][i] = float(random(100))/100;
    }
  }

  for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
  {
    for (a = 0; a < truthTable; a++)
    {
      b = random(truthTable);
      c = randomisedTT[a];
      randomisedTT[a] = randomisedTT[b];
      randomisedTT[b] = c;
    }
    error = 0;

    for (b = 0; b < truthTable; b++)
    {
      a = randomisedTT[b];
      for (i = 0; i < hiddenNodes; i++)
      {
        sum = hiddenWeights[inputNodes][i];
        for (j = 0; j < inputNodes; j++)
        {
          sum += input[a][j] * hiddenWeights[j][i];
        }
        hidden[i] = 1 / (1 + exp(-sum));
      }

      for (i = 0; i < outputNodes; i++)
      {
        sum = outputWeights[hiddenNodes][i];
        for (j = 0; j < hiddenNodes; j++)
        {
          sum += hidden[j] * outputWeights[j][i];
        }
        output[i] = 1 / (1 + exp(-sum));
```

```
          outputDelta[i] = (target[a][i] - output[i]) * output[i] * (1 - output[i]);
          error += 0.5 * (target[a][i] - output[i]) * (target[a][i] - output[i]);
        }

        for (i = 0; i < hiddenNodes; i++)
        {
          sum = 0;
          for (j = 0; j < outputNodes; j++)
          {
            sum += outputWeights[i][j] * outputDelta[j];
          }
          hiddenDelta[i] = sum * hidden[i] * (1 - hidden[i]);
        }

        for (i = 0; i < hiddenNodes; i++)
        {
          changeHiddenWeights[inputNodes][i] = learningRate * hiddenDelta[i] + momentum * changeHiddenWeights[inputNodes][i];
          hiddenWeights[inputNodes][i] += changeHiddenWeights[inputNodes][i];
          for (j = 0; j < inputNodes; j++)
          {
            changeHiddenWeights[j][i] = learningRate * input[a][j] * hiddenDelta[i] + momentum * changeHiddenWeights[j][i];
            hiddenWeights[j][i] += changeHiddenWeights[j][i];
          }
        }
      }
    }
    training = false;
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
```

```
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

There is a whole lot of code going on.

Sketch D2.18 change the output weights

Now to do the same for the hidden to output weights.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;
int i;
float sum;
int j;
float hidden[hiddenNodes];
float output[outputNodes];
float error;
float outputDelta[outputNodes];
float hiddenDelta[hiddenNodes];
float changeHiddenWeights[inputNodes + 1][hiddenNodes];
float learningRate = 0.5;
float momentum = 0.9;
float changeOutputWeights[hiddenNodes + 1][outputNodes];

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
```

```
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
        hiddenWeights[j][i] = float(random(100))/100;
      }
    }

    for (int i = 0; i < outputNodes; i++)
    {
      for (int j = 0; j <= hiddenNodes; j++)
```

```
      {
        outputWeights[j][i] = float(random(100))/100;
      }
    }

    for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
    {
      for (a = 0; a < truthTable; a++)
      {
        b = random(truthTable);
        c = randomisedTT[a];
        randomisedTT[a] = randomisedTT[b];
        randomisedTT[b] = c;
      }
      error = 0;

      for (b = 0; b < truthTable; b++)
      {
        a = randomisedTT[b];
        for (i = 0; i < hiddenNodes; i++)
        {
          sum = hiddenWeights[inputNodes][i];
          for (j = 0; j < inputNodes; j++)
          {
            sum += input[a][j] * hiddenWeights[j][i];
          }
          hidden[i] = 1 / (1 + exp(-sum));
        }

        for (i = 0; i < outputNodes; i++)
        {
          sum = outputWeights[hiddenNodes][i];
          for (j = 0; j < hiddenNodes; j++)
          {
            sum += hidden[j] * outputWeights[j][i];
          }
          output[i] = 1 / (1 + exp(-sum));
          outputDelta[i] = (target[a][i] - output[i]) * output[i] * (1 -
output[i]);
```

```
          error += 0.5 * (target[a][i] - output[i]) * (target[a][i] -
output[i]);
        }

        for (i = 0; i < hiddenNodes; i++)
        {
          sum = 0;
          for (j = 0; j < outputNodes; j++)
          {
            sum += outputWeights[i][j] * outputDelta[j];
          }
          hiddenDelta[i] = sum * hidden[i] * (1 - hidden[i]);
        }

        for (i = 0; i < hiddenNodes; i++)
        {
          changeHiddenWeights[inputNodes][i] = learningRate * hiddenDelta[i] +
momentum * changeHiddenWeights[inputNodes][i];
          hiddenWeights[inputNodes][i] += changeHiddenWeights[inputNodes][i];
          for (j = 0; j < inputNodes; j++)
          {
            changeHiddenWeights[j][i] = learningRate * input[a][j] *
hiddenDelta[i] + momentum * changeHiddenWeights[j][i];
            hiddenWeights[j][i] += changeHiddenWeights[j][i];
          }
        }

        for (i = 0; i < outputNodes; i++)
        {
          changeOutputWeights[hiddenNodes][i] = learningRate * outputDelta[i]
+ momentum * changeOutputWeights[hiddenNodes][i];
          outputWeights[hiddenNodes][i] += changeOutputWeights[hiddenNodes]
[i];
          for (j = 0; j < hiddenNodes; j++)
          {
            changeOutputWeights[j][i] = learningRate * hidden[j] *
outputDelta[i] + momentum * changeOutputWeights[j][i];
            outputWeights[j][i] += changeOutputWeights[j][i];
          }
        }
```

```
      }
    }
    training = false;
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

This may take some working through.

Sketch D2.19 seeing the error

What is the error? We can send the final error to the serial monitor. This is the final error after 10000 iterations. Press the reset button to see it train a number of times.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;
int i;
float sum;
int j;
float hidden[hiddenNodes];
float output[outputNodes];
float error;
float outputDelta[outputNodes];
float hiddenDelta[hiddenNodes];
float changeHiddenWeights[inputNodes + 1][hiddenNodes];
float learningRate = 0.5;
float momentum = 0.9;
float changeOutputWeights[hiddenNodes + 1][outputNodes];

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
```

```
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
        hiddenWeights[j][i] = float(random(100))/100;
      }
    }

    for (int i = 0; i < outputNodes; i++)
    {
```

```
    for (int j = 0; j <= hiddenNodes; j++)
    {
      outputWeights[j][i] = float(random(100))/100;
    }
  }

  for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
  {
    for (a = 0; a < truthTable; a++)
    {
      b = random(truthTable);
      c = randomisedTT[a];
      randomisedTT[a] = randomisedTT[b];
      randomisedTT[b] = c;
    }
    error = 0;

    for (b = 0; b < truthTable; b++)
    {
      a = randomisedTT[b];
      for (i = 0; i < hiddenNodes; i++)
      {
        sum = hiddenWeights[inputNodes][i];
        for (j = 0; j < inputNodes; j++)
        {
          sum += input[a][j] * hiddenWeights[j][i];
        }
        hidden[i] = 1 / (1 + exp(-sum));
      }

      for (i = 0; i < outputNodes; i++)
      {
        sum = outputWeights[hiddenNodes][i];
        for (j = 0; j < hiddenNodes; j++)
        {
          sum += hidden[j] * outputWeights[j][i];
        }
        output[i] = 1 / (1 + exp(-sum));
```

```
          outputDelta[i] = (target[a][i] - output[i]) * output[i] * (1 - output[i]);
          error += 0.5 * (target[a][i] - output[i]) * (target[a][i] - output[i]);
        }

        for (i = 0; i < hiddenNodes; i++)
        {
          sum = 0;
          for (j = 0; j < outputNodes; j++)
          {
            sum += outputWeights[i][j] * outputDelta[j];
          }
          hiddenDelta[i] = sum * hidden[i] * (1 - hidden[i]);
        }

        for (i = 0; i < hiddenNodes; i++)
        {
          changeHiddenWeights[inputNodes][i] = learningRate * hiddenDelta[i] + momentum * changeHiddenWeights[inputNodes][i];
          hiddenWeights[inputNodes][i] += changeHiddenWeights[inputNodes][i];
          for (j = 0; j < inputNodes; j++)
          {
            changeHiddenWeights[j][i] = learningRate * input[a][j] * hiddenDelta[i] + momentum * changeHiddenWeights[j][i];
            hiddenWeights[j][i] += changeHiddenWeights[j][i];
          }
        }

        for (i = 0; i < outputNodes; i++)
        {
          changeOutputWeights[hiddenNodes][i] = learningRate * outputDelta[i] + momentum * changeOutputWeights[hiddenNodes][i];
          outputWeights[hiddenNodes][i] += changeOutputWeights[hiddenNodes][i];
          for (j = 0; j < hiddenNodes; j++)
          {
            changeOutputWeights[j][i] = learningRate * hidden[j] * outputDelta[i] + momentum * changeOutputWeights[j][i];
            outputWeights[j][i] += changeOutputWeights[j][i];
```

```
          }
        }
      }
    }
    training = false;
    Serial.print(" error = ");
    Serial.println(error, 5);
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

The final error is around `0.00004` and `0.00005`, which is plenty low enough.

Figure D2.19

ANN.ino

```
          hiddenWeights[j][i] += changeHiddenWeights[j][i];
        }
      }

      for (i = 0; i < outputNodes; i++)
      {
        changeOutputWeights[hiddenNodes][i] = learningRate * outputDelta[i] + momentum * changeOutputWeights[hiddenNodes][i];
        outputWeights[hiddenNodes][i] += changeOutputWeights[hiddenNodes][i];
        for (j = 0; j < hiddenNodes; j++)
        {
          changeOutputWeights[j][i] = learningRate * hidden[j] * outputDelta[i] + momentum * changeOutputWeights[j][i];
          outputWeights[j][i] += changeOutputWeights[j][i];
        }
      }
    }
  }
  training = false;
  Serial.print(" error = ");
  Serial.println(error, 5);
}
```

Output | Serial Monitor

Message (Enter to send message to 'Arduino Nano 33 BLE' on '/dev/cu.usbmodem1101') | No Line Ending | 9600 baud

```
error = 0.00005
error = 0.00004
error = 0.00004
error = 0.00005
```

Sketch D2.20 RGB LED training finished

One problem we have is that the training takes a few seconds, but we have no indication as to when the training is finished. We can do something about that by using the RGB LED that is built into the board.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;
int i;
float sum;
int j;
float hidden[hiddenNodes];
float output[outputNodes];
float error;
float outputDelta[outputNodes];
float hiddenDelta[hiddenNodes];
float changeHiddenWeights[inputNodes + 1][hiddenNodes];
float learningRate = 0.5;
float momentum = 0.9;
float changeOutputWeights[hiddenNodes + 1][outputNodes];

const int input[truthTable][inputNodes] = {
  { 0, 0 },
```

```
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    digitalWrite(LEDR, LOW);
    digitalWrite(LEDG, HIGH);
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
        hiddenWeights[j][i] = float(random(100))/100;
```

```
    }
  }

  for (int i = 0; i < outputNodes; i++)
  {
    for (int j = 0; j <= hiddenNodes; j++)
    {
      outputWeights[j][i] = float(random(100))/100;
    }
  }

  for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
  {
    for (a = 0; a < truthTable; a++)
    {
      b = random(truthTable);
      c = randomisedTT[a];
      randomisedTT[a] = randomisedTT[b];
      randomisedTT[b] = c;
    }
    error = 0;

    for (b = 0; b < truthTable; b++)
    {
      a = randomisedTT[b];
      for (i = 0; i < hiddenNodes; i++)
      {
        sum = hiddenWeights[inputNodes][i];
        for (j = 0; j < inputNodes; j++)
        {
          sum += input[a][j] * hiddenWeights[j][i];
        }
        hidden[i] = 1 / (1 + exp(-sum));
      }

      for (i = 0; i < outputNodes; i++)
      {
        sum = outputWeights[hiddenNodes][i];
```

```
          for (j = 0; j < hiddenNodes; j++)
          {
            sum += hidden[j] * outputWeights[j][i];
          }
          output[i] = 1 / (1 + exp(-sum));
          outputDelta[i] = (target[a][i] - output[i]) * output[i] * (1 -
output[i]);
          error += 0.5 * (target[a][i] - output[i]) * (target[a][i] -
output[i]);
        }

        for (i = 0; i < hiddenNodes; i++)
        {
          sum = 0;
          for (j = 0; j < outputNodes; j++)
          {
            sum += outputWeights[i][j] * outputDelta[j];
          }
          hiddenDelta[i] = sum * hidden[i] * (1 - hidden[i]);
        }

        for (i = 0; i < hiddenNodes; i++)
        {
          changeHiddenWeights[inputNodes][i] = learningRate * hiddenDelta[i] +
momentum * changeHiddenWeights[inputNodes][i];
          hiddenWeights[inputNodes][i] += changeHiddenWeights[inputNodes][i];
          for (j = 0; j < inputNodes; j++)
          {
            changeHiddenWeights[j][i] = learningRate * input[a][j] *
hiddenDelta[i] + momentum * changeHiddenWeights[j][i];
            hiddenWeights[j][i] += changeHiddenWeights[j][i];
          }
        }

        for (i = 0; i < outputNodes; i++)
        {
          changeOutputWeights[hiddenNodes][i] = learningRate * outputDelta[i]
+ momentum * changeOutputWeights[hiddenNodes][i];
          outputWeights[hiddenNodes][i] += changeOutputWeights[hiddenNodes]
[i];
```

```
        for (j = 0; j < hiddenNodes; j++)
        {
          changeOutputWeights[j][i] = learningRate * hidden[j] *
outputDelta[i] + momentum * changeOutputWeights[j][i];
          outputWeights[j][i] += changeOutputWeights[j][i];
        }
      }
    }
  }
    training = false;
    digitalWrite(LEDR, HIGH);
    digitalWrite(LEDG, LOW);
    Serial.print(" error = ");
    Serial.println(error, 5);
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW || inputPin2 == LOW)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

Notes

It gives you a visual clue when training is finished.

Sketch D2.21 truth table buttons

Now that the training has finished, we can predict the outcome for each row of the truth table. We have the final weights (hidden and output), so we can determine the final output for a particular set of inputs. We will use the variable a to keep track of which set of inputs we want. Before we launch into the predicting business, we need to determine which combination of buttons relate to which part of the input[] array.

❗ Remove:

```
if (inputPin1 == LOW || inputPin2 == LOW)
 {
   digitalWrite(ledPin, HIGH);
 }
 else
 {
   digitalWrite(ledPin, LOW);
 }
```

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;
int i;
float sum;
int j;
float hidden[hiddenNodes];
float output[outputNodes];
float error;
```

```
float outputDelta[outputNodes];
float hiddenDelta[hiddenNodes];
float changeHiddenWeights[inputNodes + 1][hiddenNodes];
float learningRate = 0.5;
float momentum = 0.9;
float changeOutputWeights[hiddenNodes + 1][outputNodes];

const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
```

```
if (training == true)
{
  digitalWrite(LEDR, LOW);
  digitalWrite(LEDG, HIGH);
  for (int i = 0; i < hiddenNodes; i++)
  {
    for (int j = 0; j <= inputNodes; j++)
    {
      hiddenWeights[j][i] = float(random(100))/100;
    }
  }

  for (int i = 0; i < outputNodes; i++)
  {
    for (int j = 0; j <= hiddenNodes; j++)
    {
      outputWeights[j][i] = float(random(100))/100;
    }
  }

  for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
  {
    for (a = 0; a < truthTable; a++)
    {
      b = random(truthTable);
      c = randomisedTT[a];
      randomisedTT[a] = randomisedTT[b];
      randomisedTT[b] = c;
    }
    error = 0;

    for (b = 0; b < truthTable; b++)
    {
      a = randomisedTT[b];
      for (i = 0; i < hiddenNodes; i++)
      {
        sum = hiddenWeights[inputNodes][i];
        for (j = 0; j < inputNodes; j++)
```

```
          {
            sum += input[a][j] * hiddenWeights[j][i];
          }
          hidden[i] = 1 / (1 + exp(-sum));
        }

        for (i = 0; i < outputNodes; i++)
        {
          sum = outputWeights[hiddenNodes][i];
          for (j = 0; j < hiddenNodes; j++)
          {
            sum += hidden[j] * outputWeights[j][i];
          }
          output[i] = 1 / (1 + exp(-sum));
          outputDelta[i] = (target[a][i] - output[i]) * output[i] * (1 -
output[i]);
          error += 0.5 * (target[a][i] - output[i]) * (target[a][i] -
output[i]);
        }

        for (i = 0; i < hiddenNodes; i++)
        {
          sum = 0;
          for (j = 0; j < outputNodes; j++)
          {
            sum += outputWeights[i][j] * outputDelta[j];
          }
          hiddenDelta[i] = sum * hidden[i] * (1 - hidden[i]);
        }

        for (i = 0; i < hiddenNodes; i++)
        {
          changeHiddenWeights[inputNodes][i] = learningRate * hiddenDelta[i] +
momentum * changeHiddenWeights[inputNodes][i];
          hiddenWeights[inputNodes][i] += changeHiddenWeights[inputNodes][i];
          for (j = 0; j < inputNodes; j++)
          {
            changeHiddenWeights[j][i] = learningRate * input[a][j] *
hiddenDelta[i] + momentum * changeHiddenWeights[j][i];
```

```
            hiddenWeights[j][i] += changeHiddenWeights[j][i];
          }
        }

        for (i = 0; i < outputNodes; i++)
        {
          changeOutputWeights[hiddenNodes][i] = learningRate * outputDelta[i]
+ momentum * changeOutputWeights[hiddenNodes][i];
          outputWeights[hiddenNodes][i] += changeOutputWeights[hiddenNodes]
[i];
          for (j = 0; j < hiddenNodes; j++)
          {
            changeOutputWeights[j][i] = learningRate * hidden[j] *
outputDelta[i] + momentum * changeOutputWeights[j][i];
            outputWeights[j][i] += changeOutputWeights[j][i];
          }
        }
      }
    }
    training = false;
    digitalWrite(LEDR, HIGH);
    digitalWrite(LEDG, LOW);
    Serial.print(" error = ");
    Serial.println(error, 5);
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW && inputPin2 == HIGH)
  {
    a = 1;
  }
  else if (inputPin1 == HIGH && inputPin2 == LOW)
  {
    a = 2;
```

```
  }
  else if (inputPin1 == LOW && inputPin2 == LOW)
  {
    a = 3;
  }
  else
  {
    a = 0;
  }
}
```

Notes

The buttons aren't connected to the LED anymore.

Sketch D2.22 inputs to hidden

We can now predict the results by running the input array through the neural network. The input array passes through the hidden layer, then through to the output layer, at which point it passes through the sigmoid activation function to give us a value between `0` and `1`.

First inputs to the hidden layer.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;
int i;
float sum;
int j;
float hidden[hiddenNodes];
float output[outputNodes];
float error;
float outputDelta[outputNodes];
float hiddenDelta[hiddenNodes];
float changeHiddenWeights[inputNodes + 1][hiddenNodes];
float learningRate = 0.5;
float momentum = 0.9;
float changeOutputWeights[hiddenNodes + 1][outputNodes];
```

```
const int input[truthTable][inputNodes] = {
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    digitalWrite(LEDR, LOW);
    digitalWrite(LEDG, HIGH);
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
```

```
      {
        hiddenWeights[j][i] = float(random(100))/100;
      }
    }

    for (int i = 0; i < outputNodes; i++)
    {
      for (int j = 0; j <= hiddenNodes; j++)
      {
        outputWeights[j][i] = float(random(100))/100;
      }
    }

    for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
    {
      for (a = 0; a < truthTable; a++)
      {
        b = random(truthTable);
        c = randomisedTT[a];
        randomisedTT[a] = randomisedTT[b];
        randomisedTT[b] = c;
      }
      error = 0;

      for (b = 0; b < truthTable; b++)
      {
        a = randomisedTT[b];
        for (i = 0; i < hiddenNodes; i++)
        {
          sum = hiddenWeights[inputNodes][i];
          for (j = 0; j < inputNodes; j++)
          {
            sum += input[a][j] * hiddenWeights[j][i];
          }
          hidden[i] = 1 / (1 + exp(-sum));
        }

        for (i = 0; i < outputNodes; i++)
```

```
        {
          sum = outputWeights[hiddenNodes][i];
          for (j = 0; j < hiddenNodes; j++)
          {
            sum += hidden[j] * outputWeights[j][i];
          }
          output[i] = 1 / (1 + exp(-sum));
          outputDelta[i] = (target[a][i] - output[i]) * output[i] * (1 -
output[i]);
          error += 0.5 * (target[a][i] - output[i]) * (target[a][i] -
output[i]);
        }

        for (i = 0; i < hiddenNodes; i++)
        {
          sum = 0;
          for (j = 0; j < outputNodes; j++)
          {
            sum += outputWeights[i][j] * outputDelta[j];
          }
          hiddenDelta[i] = sum * hidden[i] * (1 - hidden[i]);
        }

        for (i = 0; i < hiddenNodes; i++)
        {
          changeHiddenWeights[inputNodes][i] = learningRate * hiddenDelta[i] +
momentum * changeHiddenWeights[inputNodes][i];
          hiddenWeights[inputNodes][i] += changeHiddenWeights[inputNodes][i];
          for (j = 0; j < inputNodes; j++)
          {
            changeHiddenWeights[j][i] = learningRate * input[a][j] *
hiddenDelta[i] + momentum * changeHiddenWeights[j][i];
            hiddenWeights[j][i] += changeHiddenWeights[j][i];
          }
        }

        for (i = 0; i < outputNodes; i++)
        {
          changeOutputWeights[hiddenNodes][i] = learningRate * outputDelta[i]
+ momentum * changeOutputWeights[hiddenNodes][i];
```

```
          outputWeights[hiddenNodes][i] += changeOutputWeights[hiddenNodes]
[i];
          for (j = 0; j < hiddenNodes; j++)
          {
            changeOutputWeights[j][i] = learningRate * hidden[j] *
outputDelta[i] + momentum * changeOutputWeights[j][i];
            outputWeights[j][i] += changeOutputWeights[j][i];
          }
        }
      }
    }
    training = false;
    digitalWrite(LEDR, HIGH);
    digitalWrite(LEDG, LOW);
    Serial.print(" error = ");
    Serial.println(error, 5);
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW && inputPin2 == HIGH)
  {
    a = 1;
  }
  else if (inputPin1 == HIGH && inputPin2 == LOW)
  {
    a = 2;
  }
  else if (inputPin1 == LOW && inputPin2 == LOW)
  {
    a = 3;
  }
  else
  {
    a = 0;
```

```
  }

  for (i = 0; i < hiddenNodes; i++)
  {
    sum = hiddenWeights[inputNodes][i];
    for (j = 0; j < inputNodes; j++)
    {
      sum += input[a][j] * hiddenWeights[j][i];
    }
    hidden[i] = 1 / (1 + exp(-sum));
  }
}
```

Notes

We are not training (no backpropagation), just feeding the data forward through the already trained model.

Sketch D2.23 hidden to output

Final piece of the jigsaw, we move the hidden outputs through to the output layer and calculate the final result (output). We will print them to the serial monitor to `five decimal places`. When you press one of the buttons, it should read nearly one; pressing both buttons simultaneously should bring it back to almost zero.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;
int i;
float sum;
int j;
float hidden[hiddenNodes];
float output[outputNodes];
float error;
float outputDelta[outputNodes];
float hiddenDelta[hiddenNodes];
float changeHiddenWeights[inputNodes + 1][hiddenNodes];
float learningRate = 0.5;
float momentum = 0.9;
float changeOutputWeights[hiddenNodes + 1][outputNodes];

const int input[truthTable][inputNodes] = {
```

```
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    digitalWrite(LEDR, LOW);
    digitalWrite(LEDG, HIGH);
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
```

```
      hiddenWeights[j][i] = float(random(100))/100;
    }
  }

  for (int i = 0; i < outputNodes; i++)
  {
    for (int j = 0; j <= hiddenNodes; j++)
    {
      outputWeights[j][i] = float(random(100))/100;
    }
  }

  for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
  {
    for (a = 0; a < truthTable; a++)
    {
      b = random(truthTable);
      c = randomisedTT[a];
      randomisedTT[a] = randomisedTT[b];
      randomisedTT[b] = c;
    }
    error = 0;

    for (b = 0; b < truthTable; b++)
    {
      a = randomisedTT[b];
      for (i = 0; i < hiddenNodes; i++)
      {
        sum = hiddenWeights[inputNodes][i];
        for (j = 0; j < inputNodes; j++)
        {
          sum += input[a][j] * hiddenWeights[j][i];
        }
        hidden[i] = 1 / (1 + exp(-sum));
      }

      for (i = 0; i < outputNodes; i++)
      {
```

```
            sum = outputWeights[hiddenNodes][i];
            for (j = 0; j < hiddenNodes; j++)
            {
              sum += hidden[j] * outputWeights[j][i];
            }
            output[i] = 1 / (1 + exp(-sum));
            outputDelta[i] = (target[a][i] - output[i]) * output[i] * (1 -
output[i]);
            error += 0.5 * (target[a][i] - output[i]) * (target[a][i] -
output[i]);
          }

          for (i = 0; i < hiddenNodes; i++)
          {
            sum = 0;
            for (j = 0; j < outputNodes; j++)
            {
              sum += outputWeights[i][j] * outputDelta[j];
            }
            hiddenDelta[i] = sum * hidden[i] * (1 - hidden[i]);
          }

          for (i = 0; i < hiddenNodes; i++)
          {
            changeHiddenWeights[inputNodes][i] = learningRate * hiddenDelta[i] +
momentum * changeHiddenWeights[inputNodes][i];
            hiddenWeights[inputNodes][i] += changeHiddenWeights[inputNodes][i];
            for (j = 0; j < inputNodes; j++)
            {
              changeHiddenWeights[j][i] = learningRate * input[a][j] *
hiddenDelta[i] + momentum * changeHiddenWeights[j][i];
              hiddenWeights[j][i] += changeHiddenWeights[j][i];
            }
          }

          for (i = 0; i < outputNodes; i++)
          {
            changeOutputWeights[hiddenNodes][i] = learningRate * outputDelta[i]
+ momentum * changeOutputWeights[hiddenNodes][i];
```

```
          outputWeights[hiddenNodes][i] += changeOutputWeights[hiddenNodes]
[i];
          for (j = 0; j < hiddenNodes; j++)
          {
            changeOutputWeights[j][i] = learningRate * hidden[j] *
outputDelta[i] + momentum * changeOutputWeights[j][i];
            outputWeights[j][i] += changeOutputWeights[j][i];
          }
        }
      }
    }
    training = false;
    digitalWrite(LEDR, HIGH);
    digitalWrite(LEDG, LOW);
    Serial.print(" error = ");
    Serial.println(error, 5);
  }
  buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW && inputPin2 == HIGH)
  {
    a = 1;
  }
  else if (inputPin1 == HIGH && inputPin2 == LOW)
  {
    a = 2;
  }
  else if (inputPin1 == LOW && inputPin2 == LOW)
  {
    a = 3;
  }
  else
  {
    a = 0;
```

```
  }

  for (i = 0; i < hiddenNodes; i++)
  {
    sum = hiddenWeights[inputNodes][i];
    for (j = 0; j < inputNodes; j++)
    {
      sum += input[a][j] * hiddenWeights[j][i];
    }
    hidden[i] = 1 / (1 + exp(-sum));
  }

  for (i = 0; i < outputNodes; i++)
  {
    sum = outputWeights[hiddenNodes][i];
    for (j = 0; j < hiddenNodes; j++)
    {
      sum += hidden[j] * outputWeights[j][i];
    }
    output[i] = 1 / (1 + exp(-sum));
    Serial.println(output[i], 5);
  }
}
```

Notes

The feed forward reaches the final output neuron. Figure D2.23a shows the results for no button pressed (`LED off`). Figure D2.23b is one button pressed (`LED on`). You should get an equally low number when both are pressed (`LED effectively off`).

Figure D2.23a

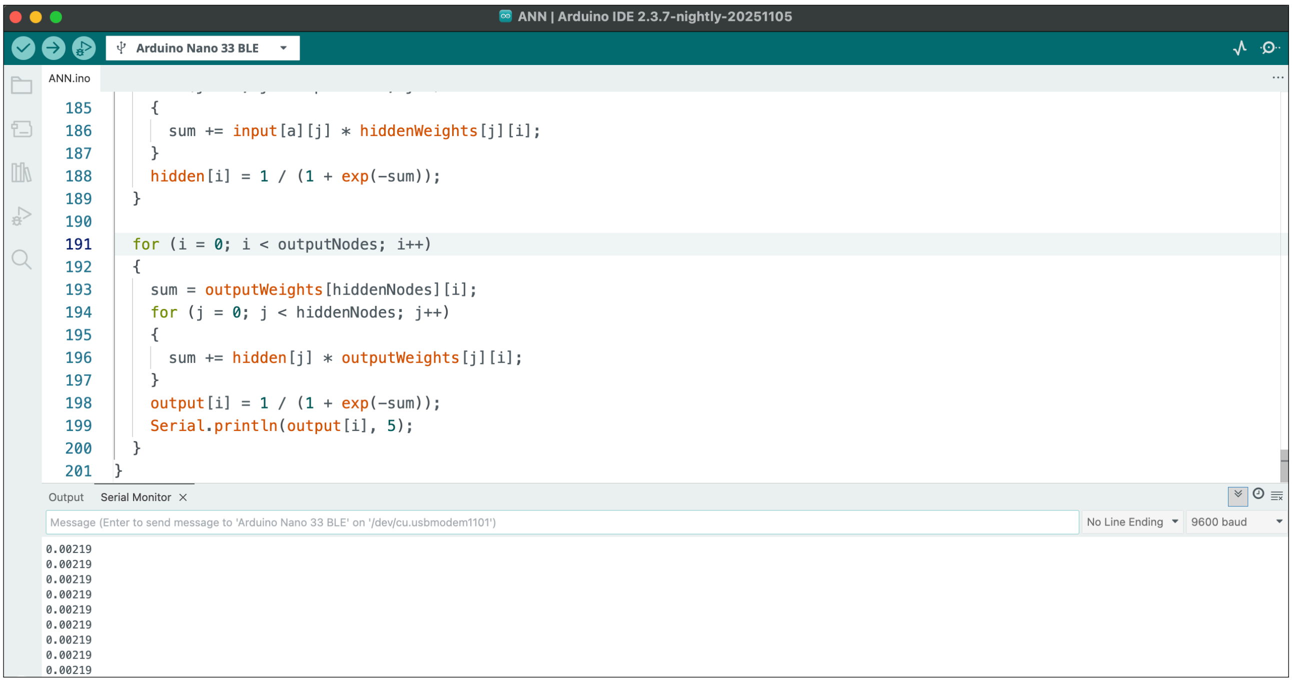
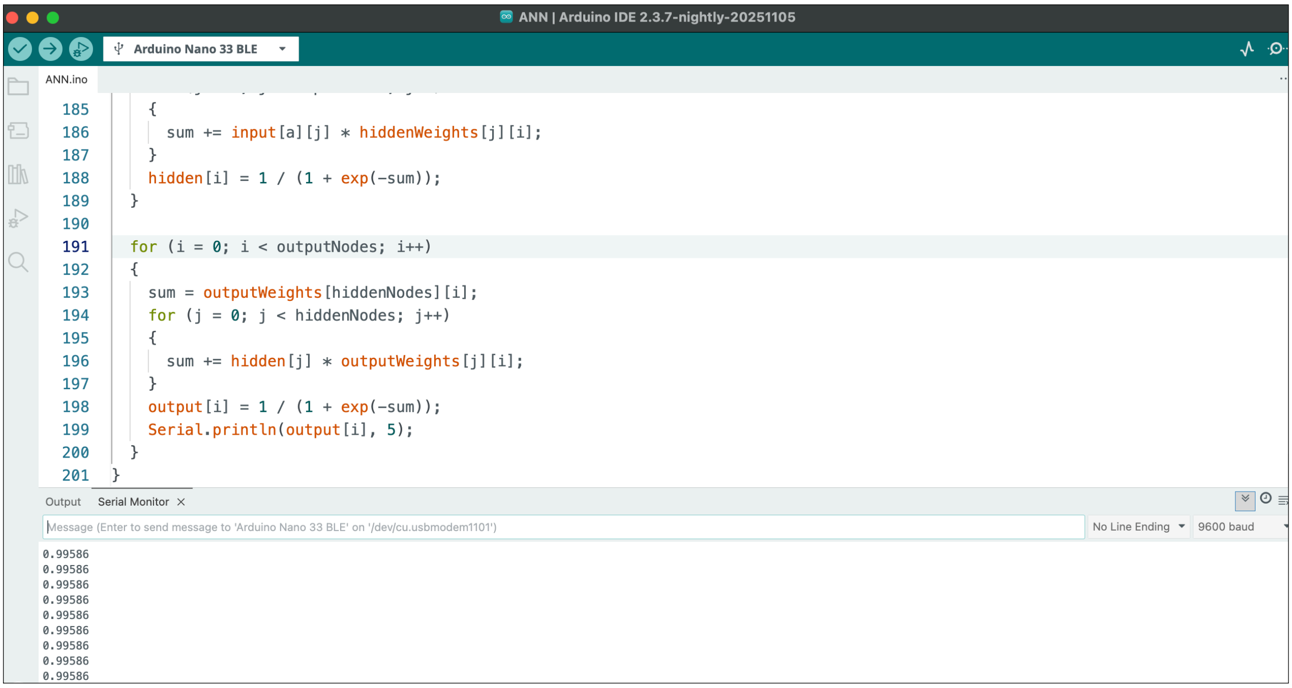


Figure D2.23b



Sketch D2.24 output to LED

We are going to take the prediction and convert it into an LED. We want it in the format of an integer between `0` and `255`. To do that, we convert the `float` to an `integer` and multiply it by `255`. For some instances, it might glow very slightly when it should be off.

Arduino sketch

```
const int ledPin = 13;
const int buttonPin1 = 2;
const int buttonPin2 = 3;
int inputPin1;
int inputPin2;
const int truthTable = 4;
const int inputNodes = 2;
const int hiddenNodes = 4;
const int outputNodes = 1;
bool training = true;
float hiddenWeights[inputNodes + 1][hiddenNodes];
float outputWeights[hiddenNodes + 1][outputNodes];
int randomisedTT[truthTable];
int a;
long trainingCycle;
int b;
int c;
int i;
float sum;
int j;
float hidden[hiddenNodes];
float output[outputNodes];
float error;
float outputDelta[outputNodes];
float hiddenDelta[hiddenNodes];
float changeHiddenWeights[inputNodes + 1][hiddenNodes];
float learningRate = 0.5;
float momentum = 0.9;
float changeOutputWeights[hiddenNodes + 1][outputNodes];

const int input[truthTable][inputNodes] = {
```

```
  { 0, 0 },
  { 1, 0 },
  { 0, 1 },
  { 1, 1 }
};

const int target[truthTable][outputNodes] = {
  { 0 },
  { 1 },
  { 1 },
  { 0 }
};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(buttonPin1, INPUT_PULLUP);
  pinMode(buttonPin2, INPUT_PULLUP);
  randomSeed(analogRead(4));
  for (a = 0; a < truthTable; a++)
  {
    randomisedTT[a] = a;
  }
}

void loop()
{
  if (training == true)
  {
    digitalWrite(LEDR, LOW);
    digitalWrite(LEDG, HIGH);
    for (int i = 0; i < hiddenNodes; i++)
    {
      for (int j = 0; j <= inputNodes; j++)
      {
```

```
      hiddenWeights[j][i] = float(random(100))/100;
    }
  }

  for (int i = 0; i < outputNodes; i++)
  {
    for (int j = 0; j <= hiddenNodes; j++)
    {
      outputWeights[j][i] = float(random(100))/100;
    }
  }

  for (trainingCycle = 0; trainingCycle < 10000; trainingCycle++)
  {
    for (a = 0; a < truthTable; a++)
    {
      b = random(truthTable);
      c = randomisedTT[a];
      randomisedTT[a] = randomisedTT[b];
      randomisedTT[b] = c;
    }
    error = 0;

    for (b = 0; b < truthTable; b++)
    {
      a = randomisedTT[b];
      for (i = 0; i < hiddenNodes; i++)
      {
        sum = hiddenWeights[inputNodes][i];
        for (j = 0; j < inputNodes; j++)
        {
          sum += input[a][j] * hiddenWeights[j][i];
        }
        hidden[i] = 1 / (1 + exp(-sum));
      }

      for (i = 0; i < outputNodes; i++)
      {
```

```
          sum = outputWeights[hiddenNodes][i];
          for (j = 0; j < hiddenNodes; j++)
          {
            sum += hidden[j] * outputWeights[j][i];
          }
          output[i] = 1 / (1 + exp(-sum));
          outputDelta[i] = (target[a][i] - output[i]) * output[i] * (1 -
output[i]);
          error += 0.5 * (target[a][i] - output[i]) * (target[a][i] -
output[i]);
        }

        for (i = 0; i < hiddenNodes; i++)
        {
          sum = 0;
          for (j = 0; j < outputNodes; j++)
          {
            sum += outputWeights[i][j] * outputDelta[j];
          }
          hiddenDelta[i] = sum * hidden[i] * (1 - hidden[i]);
        }

        for (i = 0; i < hiddenNodes; i++)
        {
          changeHiddenWeights[inputNodes][i] = learningRate * hiddenDelta[i] +
momentum * changeHiddenWeights[inputNodes][i];
          hiddenWeights[inputNodes][i] += changeHiddenWeights[inputNodes][i];
          for (j = 0; j < inputNodes; j++)
          {
            changeHiddenWeights[j][i] = learningRate * input[a][j] *
hiddenDelta[i] + momentum * changeHiddenWeights[j][i];
            hiddenWeights[j][i] += changeHiddenWeights[j][i];
          }
        }

        for (i = 0; i < outputNodes; i++)
        {
          changeOutputWeights[hiddenNodes][i] = learningRate * outputDelta[i]
+ momentum * changeOutputWeights[hiddenNodes][i];
```

```
        outputWeights[hiddenNodes][i] += changeOutputWeights[hiddenNodes]
[i];
        for (j = 0; j < hiddenNodes; j++)
        {
          changeOutputWeights[j][i] = learningRate * hidden[j] *
outputDelta[i] + momentum * changeOutputWeights[j][i];
          outputWeights[j][i] += changeOutputWeights[j][i];
        }
      }
    }
  }
  training = false;
  digitalWrite(LEDR, HIGH);
  digitalWrite(LEDG, LOW);
  Serial.print(" error = ");
  Serial.println(error, 5);
 }
 buttons();
}

void buttons()
{
  inputPin1 = digitalRead(buttonPin1);
  inputPin2 = digitalRead(buttonPin2);
  if (inputPin1 == LOW && inputPin2 == HIGH)
  {
    a = 1;
  }
  else if (inputPin1 == HIGH && inputPin2 == LOW)
  {
    a = 2;
  }
  else if (inputPin1 == LOW && inputPin2 == LOW)
  {
    a = 3;
  }
  else
  {
    a = 0;
```

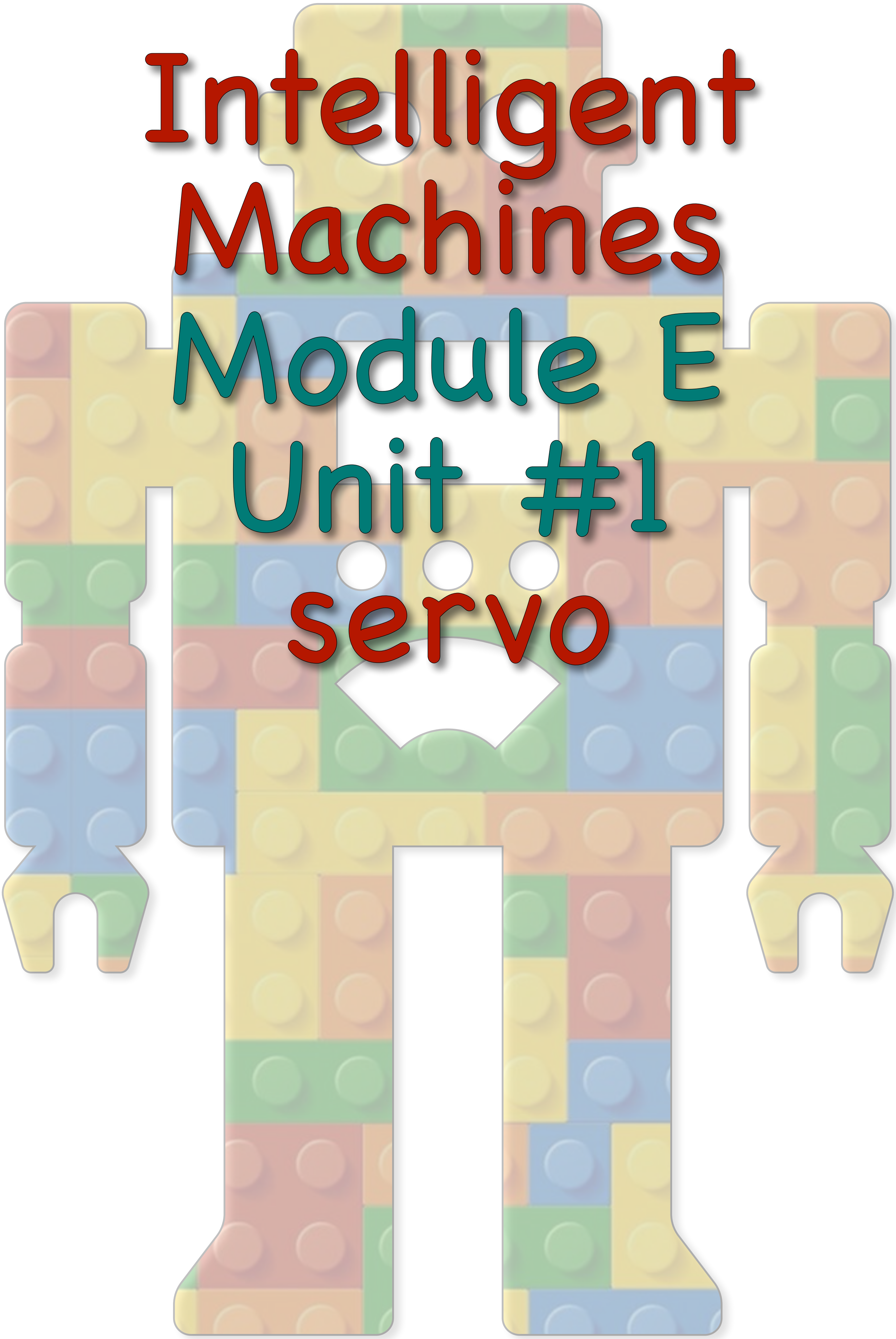
```
  }

  for (i = 0; i < hiddenNodes; i++)
  {
    sum = hiddenWeights[inputNodes][i];
    for (j = 0; j < inputNodes; j++)
    {
      sum += input[a][j] * hiddenWeights[j][i];
    }
    hidden[i] = 1 / (1 + exp(-sum));
  }

  for (i = 0; i < outputNodes; i++)
  {
    sum = outputWeights[hiddenNodes][i];
    for (j = 0; j < hiddenNodes; j++)
    {
      sum += hidden[j] * outputWeights[j][i];
    }
    output[i] = 1 / (1 + exp(-sum));
    Serial.println(output[i], 5);
  }
  for (i = 0; i < outputNodes; i++)
  {
    analogWrite(ledPin, int(output[i] * 255));
  }
}
```

Notes

When you press one button, it should switch the LED on, and when you press both, the LED should switch off.



Intelligent Machines

Module E

Unit #1

servo

Module E Unit #1: the servo

The servo is a particular type of motor. It usually moves through a limited number of degrees. In this case, it is 180°. You can get similar ones that are continuous.

What you will need

In addition to the Arduino Nano 33 BLE and breadboard, you will need the following:

1 x breadboard power supply
1 x 9-volt battery
1 x 9V battery clip with 5.5mm/2.1mm plug
1 x TowerPro Servo Motor - SG92R Micro
4 x male-to-male jumper leads

The breadboard power supply

The power supply for the breadboard provides 5V or 3.3V along the side rails. There are a number of different types and methods of supplying power. We need to supply separate power to the servo because the Arduino Nano 33 BLE won't be able to handle the extra power needed if the servo is loaded (meets resistance). It could damage the Arduino.

The power supply module shown in the image below is a popular and very cheap one. When you place it on the breadboard, it is important that you share the ground with the board's ground.

You select whether you want 5V or 3.3V (or neither) by lifting the plastic slider and placing it over the relevant pins. The slider is shown in yellow on both sides of the power module. The 9V battery is connected by inserting the jack plug (black). The USB can be connected to the Arduino to power it (5V).

Figure 1: breadboard power supply

The 9v battery and connector

The power supply to the above module is a 9v battery with the appropriate cable. See fig.2 and fig.3.

Figure 2: 9v battery cable

Figure 3: 9v battery

The servo motor

The servo motor that is cheap and plentiful is the TowerPro Servo Motor - SG92R Micro or the TowerPro Servo Motor - SG90 Digital. They are both quite similar. The SG92R is a slight upgrade.

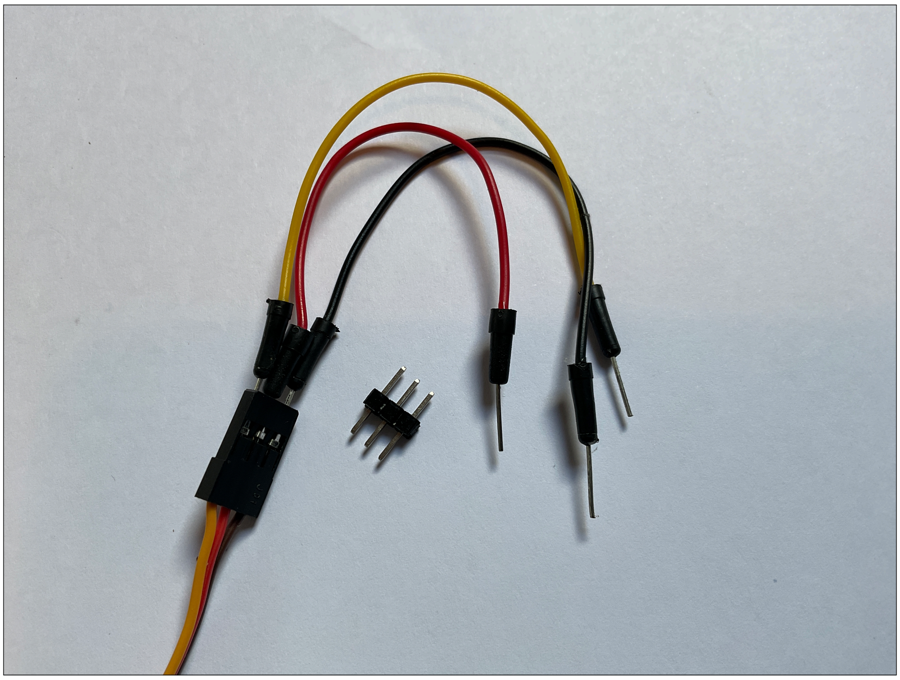
Both take a ≈5v supply from the rails on the breadboard. They have three leads. A red lead which is the 5v power supply, a brown lead which is the ground (GND), and an orange lead which will be attached to digital pin 2 in our examples.

Figure 4: servo motor

Alternative pin connectors

To make life a little simpler, you might want to get hold of a strip of pins that you can break off any number to use. I use them in this case on the breadboard to make connections between the servo and the components/breadboard. Either method is fine.

Figure 5: two ways to connect servo to Arduino



The circuit diagram

The circuit diagram is shown below. Notice the common ground.

Servo wires	Arduino Pins
Red	+ve (5V) red rail on breadboard
Brown	-ve (GND) blue rail **and** GND Arduino
Yellow	2 (digital pin 2)

Figure 6: circuit diagram

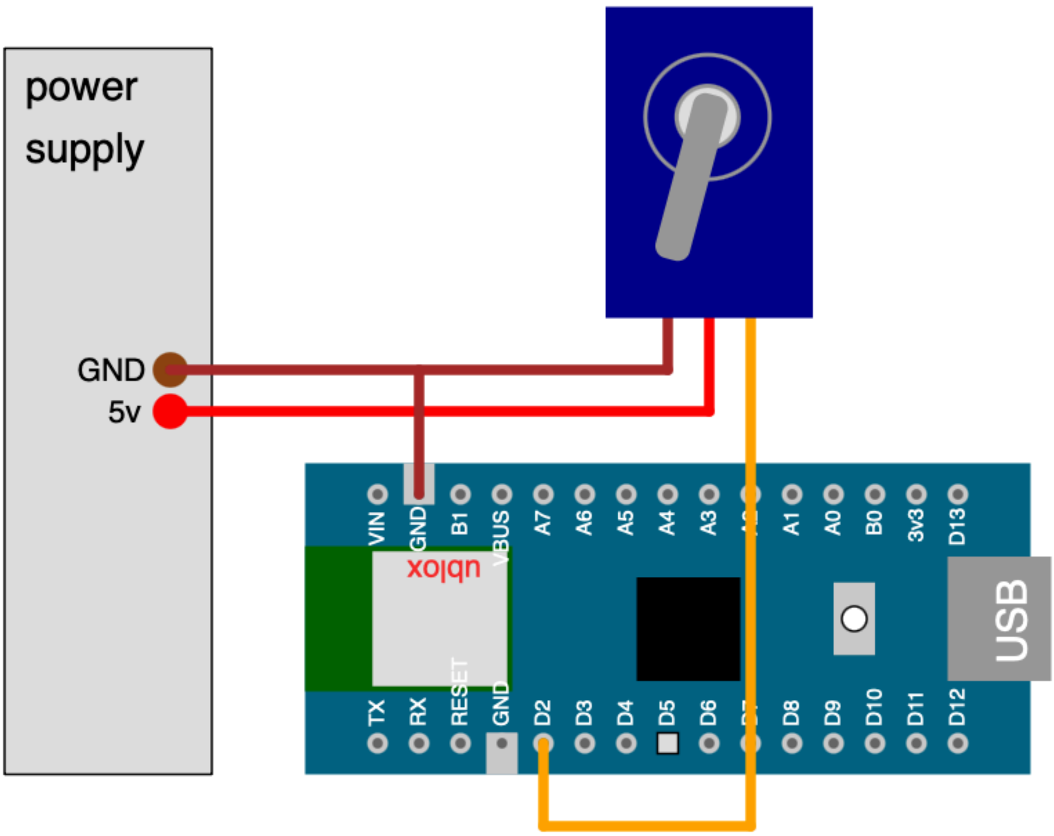
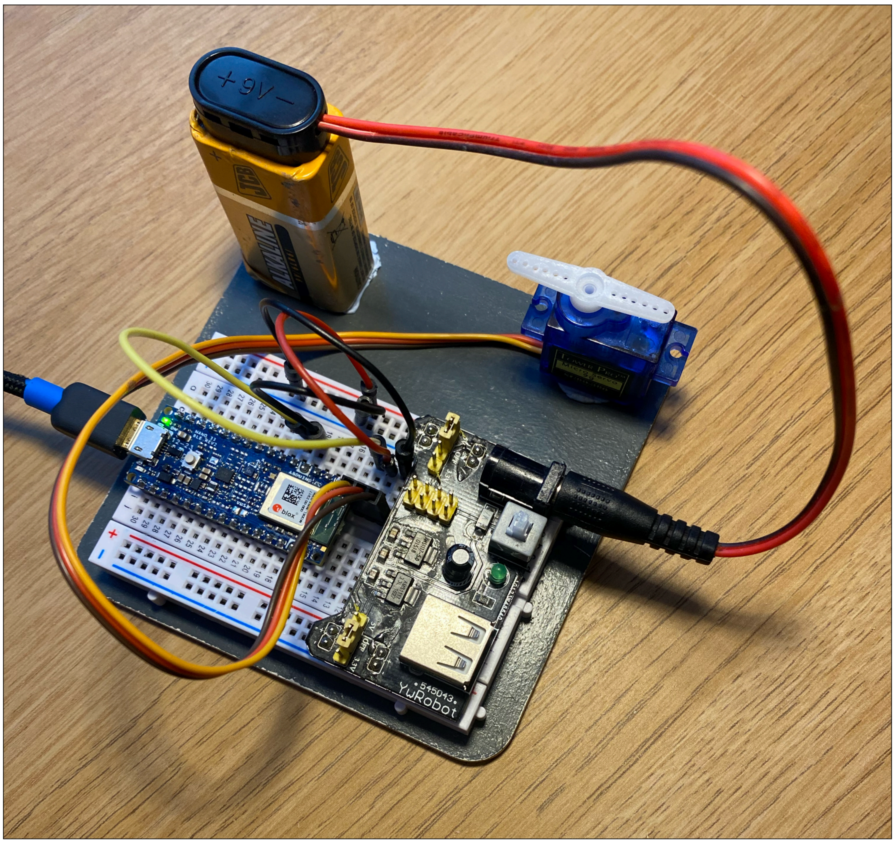


Figure 7: fully connected



Installing libraries

This is a critical part of learning to code with the Arduino and with using other coding languages like Python. Libraries are like little modules that contain a sketch or programme that does a certain function to control a complex component. It saves you having to write a lot of complicated code.

Some libraries are pre-installed, for instance, the servo library. Others you will need to install manually. This is very simple and quick to do. If you don't, you will soon get an error message, and no damage is done.

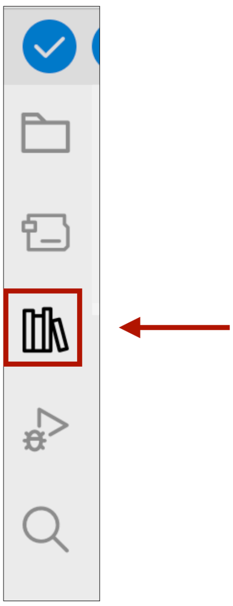
In the code, you will have a line like so...

```
#include <Servo.h>
```

This means that you are including a library called Servo.h. Sometimes you will only have to install one library, and at other times, you will install many. You can even write your own library, but that is beyond the scope of this book.

1️⃣ Click on the symbol that looks like a stack of books (this is where the libraries are, hence like a library shelf).

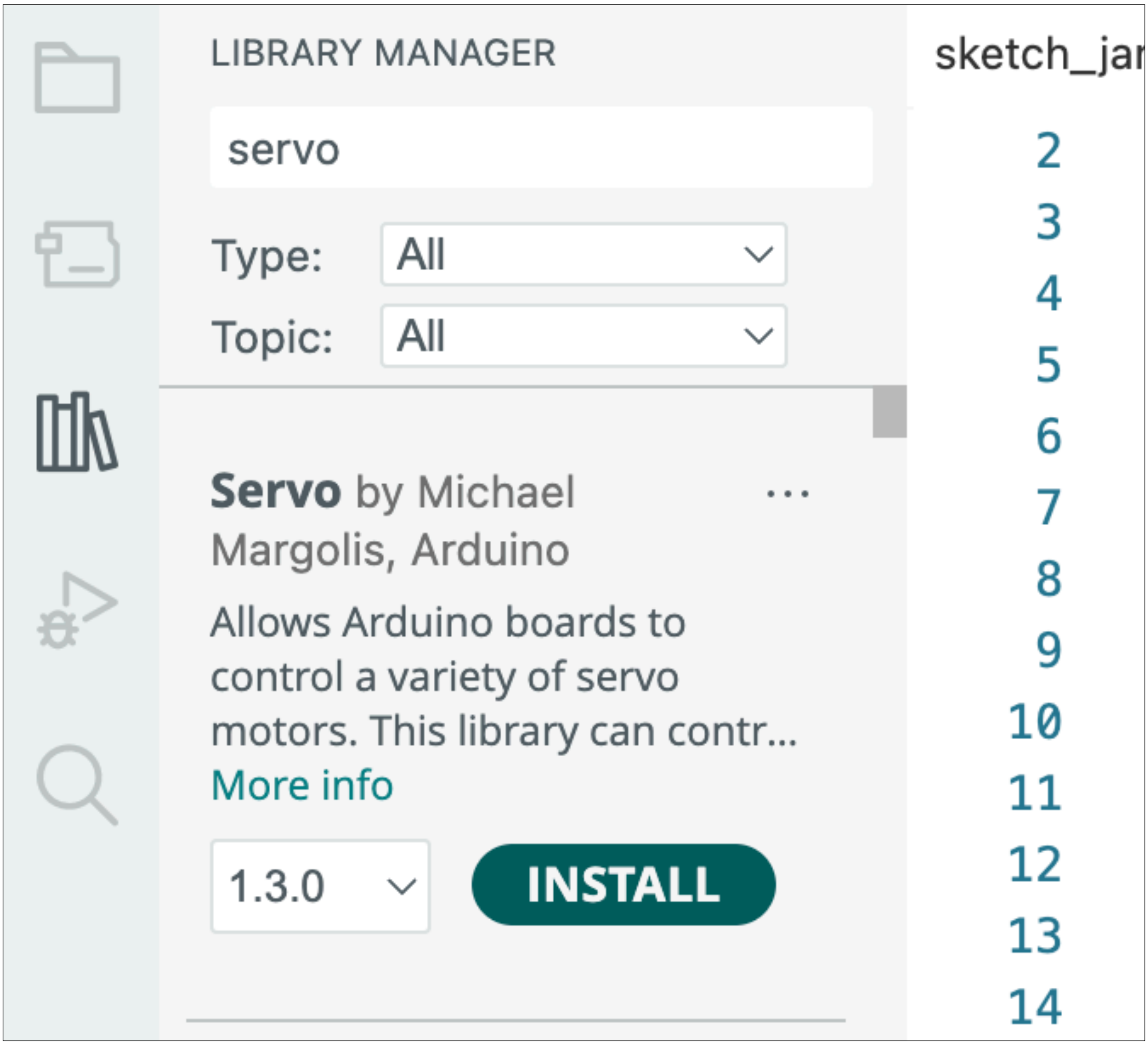
Figure 8: library



2️⃣ Type in the word servo in the box at the top and press Return. You should get a long list of possible libraries to install. The one that I recommend is highlighted here.

It is the one by Michael Margolis, which seems to be Arduino's own. Click INSTALL (latest version).

Figure 9: servo library



Sketch E1.1 position zero

Our first servo sketch. This positions the servo at an angle of 0°.

Arduino sketch

```
#include <Servo.h>
Servo myServo;

void setup()
{
  myServo.attach(2);
}

void loop()
{
  myServo.write(0);
  delay(15);
}
```

Notes

❗ You may find that the servo doesn't behave well. Mine didn't. So you could set it to 10° instead if it starts to rotate of its own accord.

Challenge

Try other angles.

Code Explanation

#include <Servo.h>	The servo library
Servo myServo;	We give the servo a name myServo, it can be any name and will need to be if you are operating more than one
myServo.attach(2);	The servo is attached to digital pin 2
myServo.write(0);	The angle is written to the servo in degrees
delay(15);	Gives the servo time to get to new position

Sketch E1.2 now 90 degrees

Next, we change the angle to 90°.

Arduino sketch

```
#include <Servo.h>
Servo myServo;

void setup()
{
  myServo.attach(2);
}

void loop()
{
  myServo.write(90);
  delay(15);
}
```

Notes

Fairly obvious, just to make the point. It should move round to a new position after you have uploaded.

Sketch E1.3 finally 180 degrees

Just to top it off.

Arduino sketch

```
#include <Servo.h>
Servo myServo;

void setup()
{
  myServo.attach(2);
}

void loop()
{
  myServo.write(180);
  delay(15);
}
```

Notes

And it should move to its final resting place.

Challenge

Have a think about how you would rotate it continuously from 0° to 180° and back again. The next sketch will show you.

Sketch E1.4 repeated rotation

We create two for() loops. The first one checks for the position of the servo arm. If it is zero, it increments 1° at a time until it reaches 180°. The second for() loop decreases the angle by −1° from 180° back to 0°. This is repeated.

Arduino sketch

```
#include <Servo.h>
Servo myServo;
int pos = 0;

void setup()
{
  myServo.attach(2);
}

void loop()
{
  for (pos = 0; pos < 180; pos += 1)
  {
    myServo.write(pos);
    delay(15);
  }
  for (pos = 180; pos >= 0; pos -= 1)
  {
    myServo.write(pos);
    delay(15);
  }
}
```

Notes

I have not broken it down into separate sketches because by now you should be able to understand how the `for()` loops work, the conditional statements, and the incremental steps.

Challenge

Try rotating it from and to different angles.

Code Explanation

`for (pos = 0; pos < 180; pos += 1)`	We cycle through from 0 to 180 in steps of 1° each iteration

Sketch E1.5 variation on a theme

Just to complete the picture, we could also do the following and watch it on the serial plotter.

Arduino sketch

```
#include <Servo.h>
Servo myServo;
int pos = 0;

void setup()
{
  myServo.attach(2);
  Serial.begin(9600);
}

void loop()
{
  for (pos = 0; pos < 180; pos++)
  {
    myServo.write(pos);
    delay(15);
    Serial.print(0);
    Serial.print(" ");
    Serial.print(180);
    Serial.print(" ");
    Serial.println(pos);
  }
  for (pos = 180; pos > 0; pos--)
  {
    myServo.write(pos);
    delay(15);
    Serial.print(0);
    Serial.print(" ");
    Serial.print(180);
    Serial.print(" ");
    Serial.println(pos);
  }
}
```

Notes

We have an extra two reference points to keep the plotter from going a bit berserk.

Challenge

Could you make the servo rotate according to a sine wave?

Figure E1.5

Intelligent Machines

Module E
Unit #2
slider servo

Module E Unit #2: slider servo

We are going to connect our Arduino Nano 33 BLE with p5.js. From there, we will control the servo with a slider.

Sketch E2.1 reading a value

Here we will be sending a value (val) through the serial connection with a p5.js sketch.

Arduino sketch

```
#include <Servo.h>
Servo myServo;
int val = 0;

void setup()
{
  myServo.attach(2);
  Serial.begin(9600);
}

void loop()
{
  if (Serial.available() > 0)
  {
    val = Serial.read();
    myServo.write(val);
  }
}
```

Notes

The Arduino is in listening mode, waiting for a value (val) to be sent from p5.js.

Sketch E2.2 the port sketch

Make sure that you have everything in the port.js sketch.

port.js

```
const serial = new p5.WebSerial()
let portButton

function navigation()
{
  if (!navigator.serial)
  {
    alert ("WebSerial is not supported. Try Chrome")
  }
  serial.getPorts()
  serial.on("noport", makePortButton)
  serial.on("portavailable", openPort)
  serial.on("requesterror", portError)
  serial.on("close", makePortButton)
}

function makePortButton()
{
  portButton = createButton("choose port")
  portButton.position(10, 10)
  portButton.mousePressed(choosePort)
}

function choosePort()
{
  if (portButton) portButton.show()
  serial.requestPort()
}

function openPort()
{
  serial.open().then(initiateSerial)
  function initiateSerial()
```

```
  {
    console.log("port open")
    if (portButton) portButton.hide()
  }
}

function portError(err)
{
  alert("Serial port error: " + err)
}

function portConnect()
{
  console.log("port connected")
  serial.getPorts()
}

function portDisconnect()
{
  serial.close()
  console.log("port disconnected")
}

function closePort()
{
  serial.close()
}
```

Notes

There is nothing new here.

Sketch E2.3 the main sketch

❗ In sketch.js.
Starting with a fairly basic sketch. We have a slider (remember the LED unit?).

sketch.js

```
let slider

function setup()
{
  createCanvas(400, 400)
  slider = createSlider(0, 180, 0)
  navigation()
}

function draw()
{
  background(220)
}
```

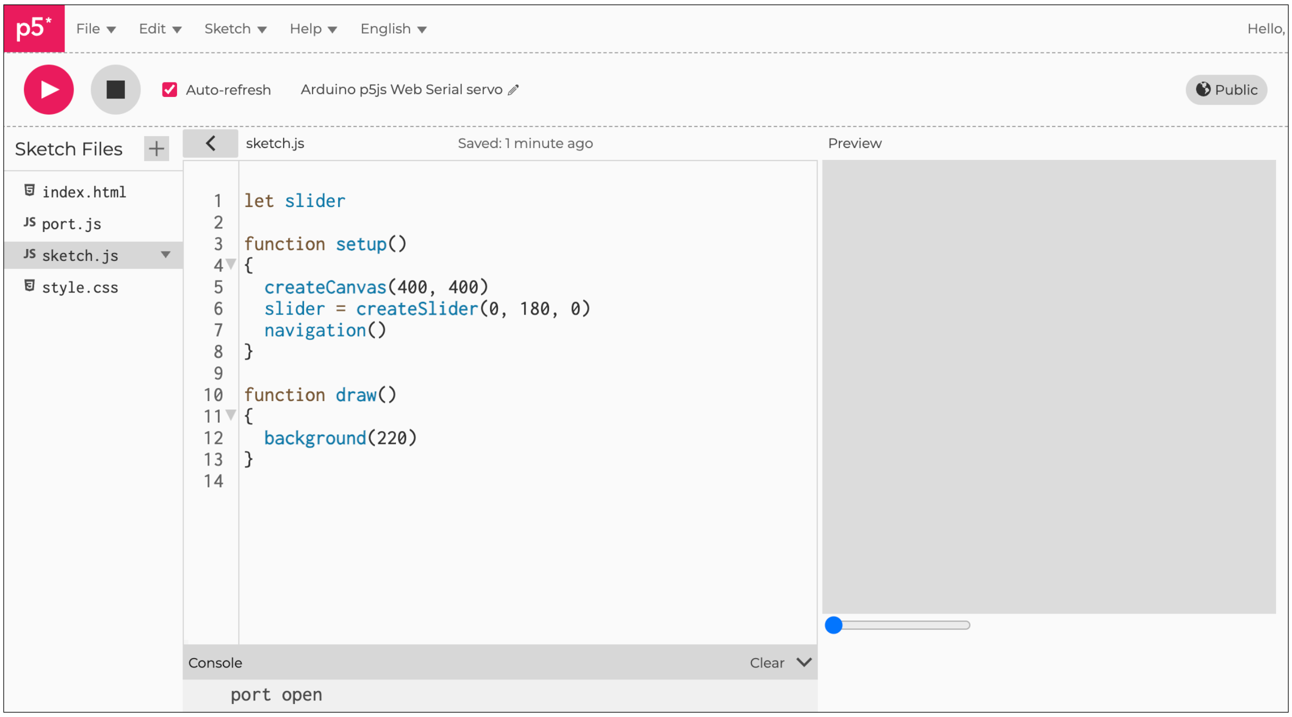
Notes
The slider doesn't do anything yet, we have not attached any functionality to it yet.

Challenge
You could place the slider on the canvas.

Code Explanation

`slider = createSlider(0, 180, 0)`	Creating a slider with value range of 0-180 and a starting value of zero.

Figure E2.3


p5*
File
Edit
Sketch
Help
English
Hello,
Auto-refresh
Arduino p5js Web Serial servo
Public
Sketch Files
index.html
port.js
sketch.js
style.css
sketch.js
Saved: 1 minute ago
Preview
let slider
function setup()
{
createCanvas(400, 400)
slider = createSlider(0, 180, 0)
navigation()
}
function draw()
{
background(220)
}
Console
Clear
port open

Sketch E2.4 attaching a value

We want to send the value of the slider to the Arduino Nano 33 BLE.

sketch.js

```
let slider
let val

function setup()
{
  createCanvas(400, 400)
  slider = createSlider(0, 180, 0)
  navigation()
}

function draw()
{
  background(220)
  val = slider.value()
  serial.write(val)
}
```

Notes

When you slide the slider you should see the servo copy the movement from 0° to 180°.

Code Explanation

`val = slider.value()`	Receives the value from the slider
`serial.write(val)`	Sends the value from the slider to the Arduino.

Sketch E2.5 drawing the servo arm

Rather than draw the whole thing I have just drawn the rotating arm showing that you can display the movement graphically, also have added the value as text.

sketch.js

```
let slider
let val

function setup()
{
  createCanvas(400, 400)
  slider = createSlider(0, 180, 0)
  navigation()
  angleMode(DEGREES)
}

function draw()
{
  background('darkred')
  val = slider.value()
  serial.write(val)
  translate(width/2, height/2)
  strokeWeight(20)
  stroke('yellow')
  line(0, 0, 100 * sin(val), 100 * cos(val))
  textSize(30)
  noStroke()
  fill('yellow')
  text('angle: ' + val, -150, -150)
}
```

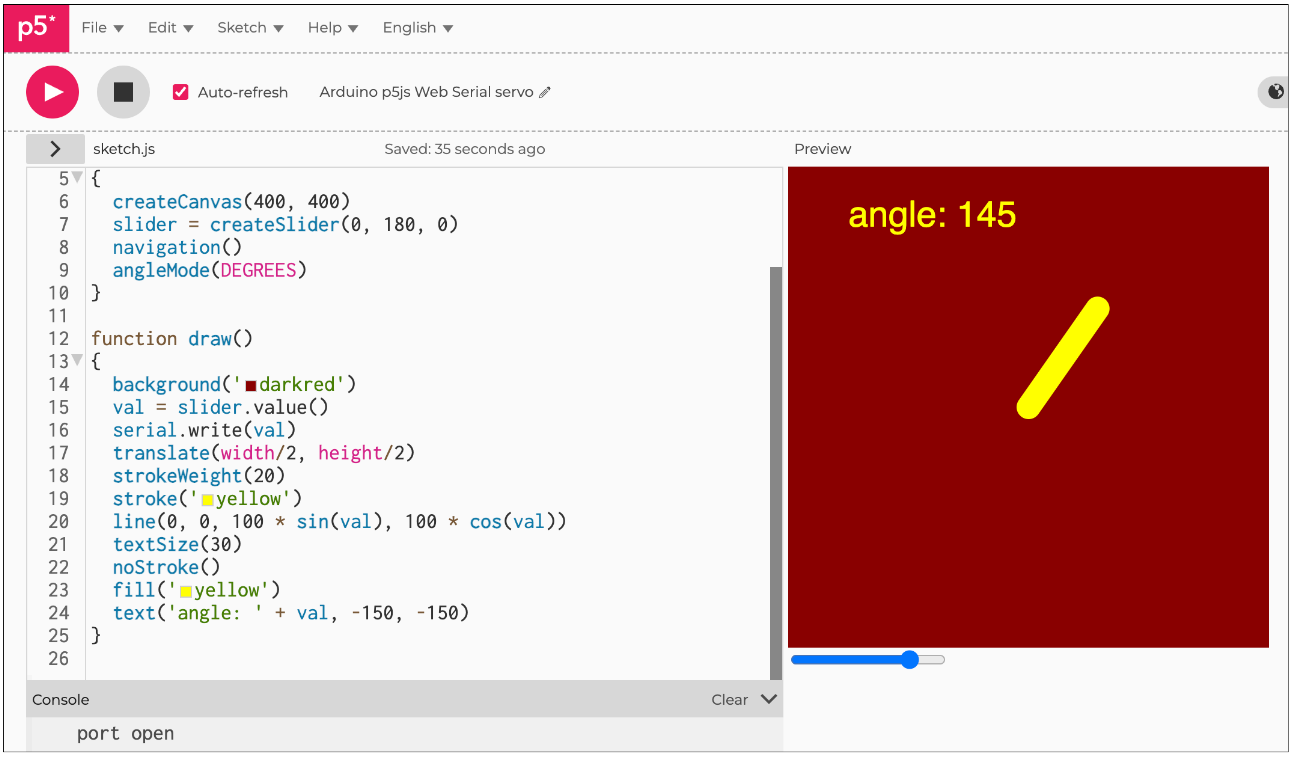
Notes

The arm follows the slider and the servo.

Challenge

1. Draw the rest of the servo.
2. Draw it in 3D.

Figure E2.6


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Arduino p5js Web Serial servo
sketch.js
Saved: 35 seconds ago
Preview
{
  createCanvas(400, 400)
  slider = createSlider(0, 180, 0)
  navigation()
  angleMode(DEGREES)
}

function draw()
{
  background('darkred')
  val = slider.value()
  serial.write(val)
  translate(width/2, height/2)
  strokeWeight(20)
  stroke('yellow')
  line(0, 0, 100 * sin(val), 100 * cos(val))
  textSize(30)
  noStroke()
  fill('yellow')
  text('angle: ' + val, -150, -150)
}
angle: 145
Console
Clear
port open

Intelligent Machines

Module E Unit #3

AI finger servo

Module E Unit #3: AI finger servo

In this unit we will control a servo using your index finger. This uses the pretrained model handPose to identify the index finger and map it across the canvas.

The position will be sent to the Arduino Nano 33 BLE which is connected to the servo. The servo rotates as your finger moves across the canvas.

❗ The port.js and the Arduino sketch remain the same.

Sketch E3.1 basic servo sketch

The same sketch as unit #2.

Arduino sketch

```
#include <Servo.h>
Servo myServo;
int val = 0;

void setup()
{
  myServo.attach(2);
  Serial.begin(9600);
}

void loop()
{
  if (Serial.available() > 0)
  {
    val = Serial.read();
    myServo.write(val);
  }
}
```

Notes

All things the same.

Sketch E3.2 index.html

We have covered the use of pretrained models in previous modules at length. So, we won't jumping into this deeply but will still build up gradually. To use a pretrained model we will need to add the library to our index.html file shown below.

index.html

```
<!DOCTYPE html>
<html lang="en"><head>
    <script src="https://cdn.jsdelivr.net/npm/p5@2.1.1/lib/p5.js"></script>
      <script src="https://unpkg.com/p5-webserial@0.1.1/build/
p5.webserial.js"></script>
   <script src="https://unpkg.com/ml5@1/dist/ml5.min.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8">
  </head>
  <body>
    <main>
    </main>
    <script src="sketch.js"></script>
    <script src="port.js"></script>
</body></html>
```

Notes

We have the three libraries: p5.js, ml5.js and p5.webserial.js.

```html
<!DOCTYPE html>
<html lang="en"><head>
    <script src="https://cdn.jsdelivr.net/npm/p5@2.1.1/lib/p5.js"></script>
      <script src="https://unpkg.com/p5-webserial@0.1.1/build/p5.webserial.js"></script>
   <script src="https://unpkg.com/ml5@1/dist/ml5.min.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8">
  </head>
  <body>
    <main>
    </main>
    <script src="sketch.js"></script>
    <script src="port.js"></script>
</body></html>
```

Console

```
port open
```

Sketch E3.3 in the beginning

Our starting sketch.

sketch.js

```
let val = 0

function setup()
{
  createCanvas(400, 400)
  navigation()
}

function draw()
{
  background(220)
  serial.write(val)
}
```

Sketch E3.4 getting the video

As I have said, I covered this bit in greater depth in the AI tutorial series. At this point we are using the webcam, taking the video, putting it in the canvas. The flipped part is to make it like a mirror rather than reversed.

sketch.js

```
let val = 0
let video

function setup()
{
  createCanvas(640, 480)
  video = createCapture(VIDEO, {flipped: true})
  video.hide()
  navigation()
}

function draw()
{
  background(220)
  image(video, 0, 0, width, height)
  serial.write(val)
}
```

Notes

You should have mirrored image of yourself on the canvas with a size of 640 x 480 to accommodate the dimensions of the webcam image.

Sketch E3.5 the handPose model

In version 2 of p5.js which I encourage you to start using (but not essential) we need to load pretrained model before we doing anything with it, so we add in async and await into the setup function. This means it is synced and waits for the model to load before doing anything else. This is crucial. The model we will be using is the handPose model.

sketch.js

```
let val = 0
let video
let handPose
let hands = []

async function setup()
{
  handPose = await ml5.handPose({flipped: true})
  createCanvas(640, 480)
  video = createCapture(VIDEO, {flipped: true})
  video.hide()
  handPose.detectStart(video, gotHands)
  navigation()
}

function draw()
{
  background(220)
  image(video, 0, 0, width, height)
  serial.write(val)
}

function gotHands(results)
{
  hands = results
}
```

Notes

The `handPose` model is part of `ml5.js`. We wait for it to be uploaded. We then use the video feed to detect `startDetect()` your hands. If it finds any then sends the information (`results`) to another function `gotHands()` as a callback. The results are stored in an array called `hands`. You won't see anything yet as we are only setting it up, we haven't used the information yet.

Sketch E3.6 now to see

Here we will now draw a circle where the tip of your index finger is. As you move your index finger the circle will follow it. We need the information from handPose (in the hands array) which identifies the tip of the index finger. In short this is keypoint 8 in the array. We get the x and y co-ordinates for that index value. Then map them as the co-ordinates of the circle. The x component is then sent to the Arduino.

sketch.js

```
let val = 0
let video
let handPose
let hands = []

async function setup()
{
  handPose = await ml5.handPose({flipped: true})
  createCanvas(640, 480)
  video = createCapture(VIDEO, {flipped: true})
  video.hide()
  handPose.detectStart(video, gotHands)
  navigation()
}

function draw()
{
  background(220)
  image(video, 0, 0, width, height)
  if (hands.length > 0)
  {
    fill('yellow')
    let index = hands[0].keypoints[8]
    circle(index.x, index.y, 20)
    val = map(index.x, 0, 640, 0, 180)
    serial.write(val)
  }
}

function gotHands(results)
{
```

```
  hands = results
}
```

Notes

You should see the circle follow your index finger. Make sure it can see your full hand (only one hand visible). Move it slowly from left to right and back again so that the circle goes from one edge of the canvas to the other.

Challenge

Use two servos and track the y value with the other servo.

Code Explanation

`if (hands.length > 0)`	Checks to see if you have any hands
`let index = hands[0].keypoints[8]`	If it finds a hand it then finds the index finger tip [8] and uses its co-ordinates [0]
`circle(index.x, index.y, 20)`	Draws the circle to the x, y co-ordinates
`val = map(index.x, 0, 640, 0, 180)`	Maps the x co-ordinate to the horizontal size of the canvas. 0 is 0, 640 is now 180
`serial.write(val)`	Sends the 0-180 value (val) to the Arduino