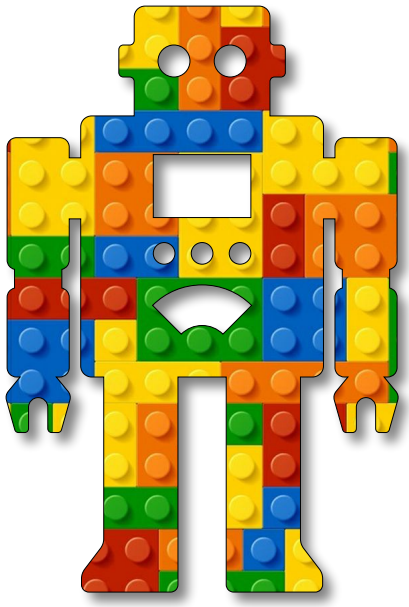
Intelligent Machines Module D Unit #2 neural network



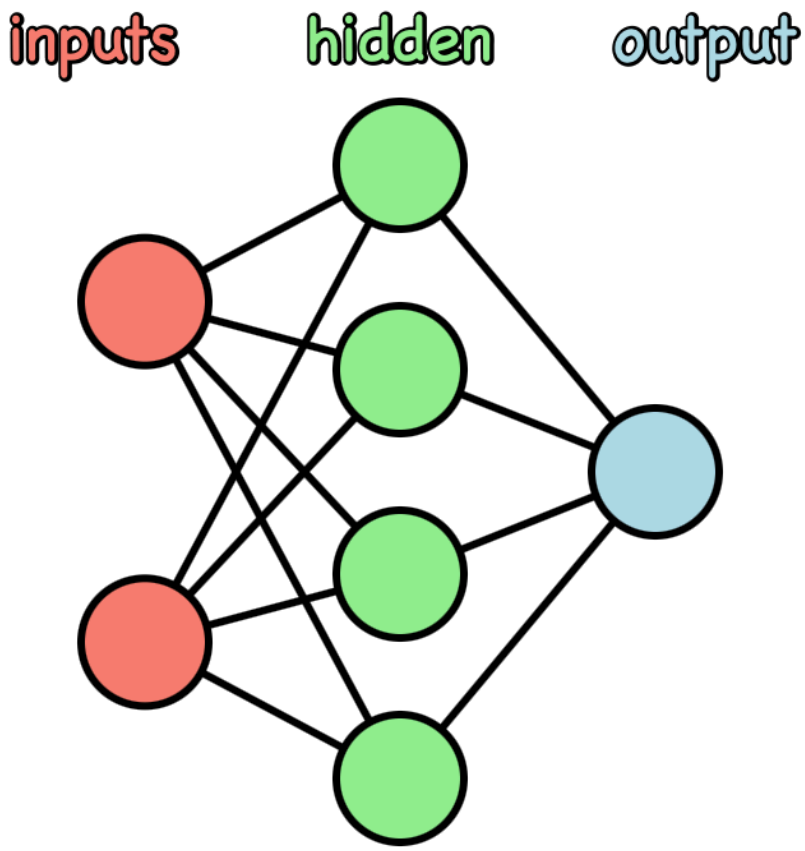
Content

Module D Unit #2 the XOR 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];
```

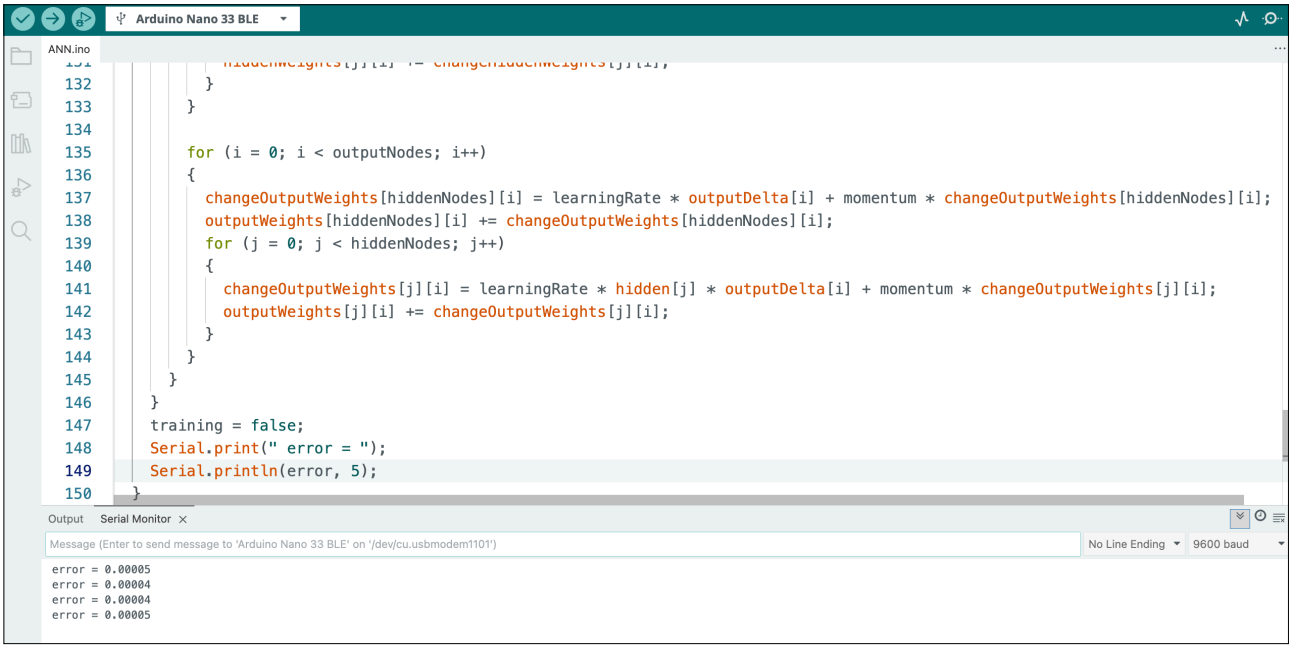
```
            outputWeights[j][i] += changeOutputWeights[j][i];
          }
        }
      }
    }
    training = false;
    Serial.print(" error = ");
    Serial.println(error, 5);
  }
  buttons();
}

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


Arduino Nano 33 BLE
ANN.ino
}
}
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

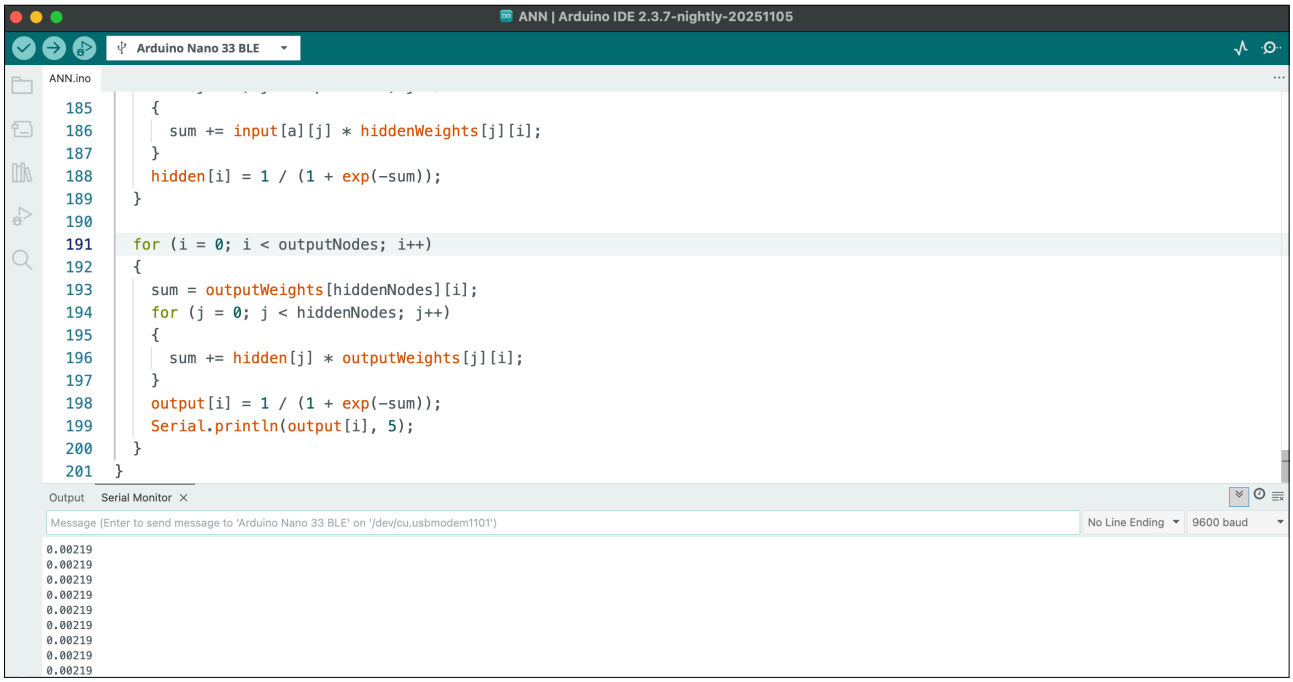
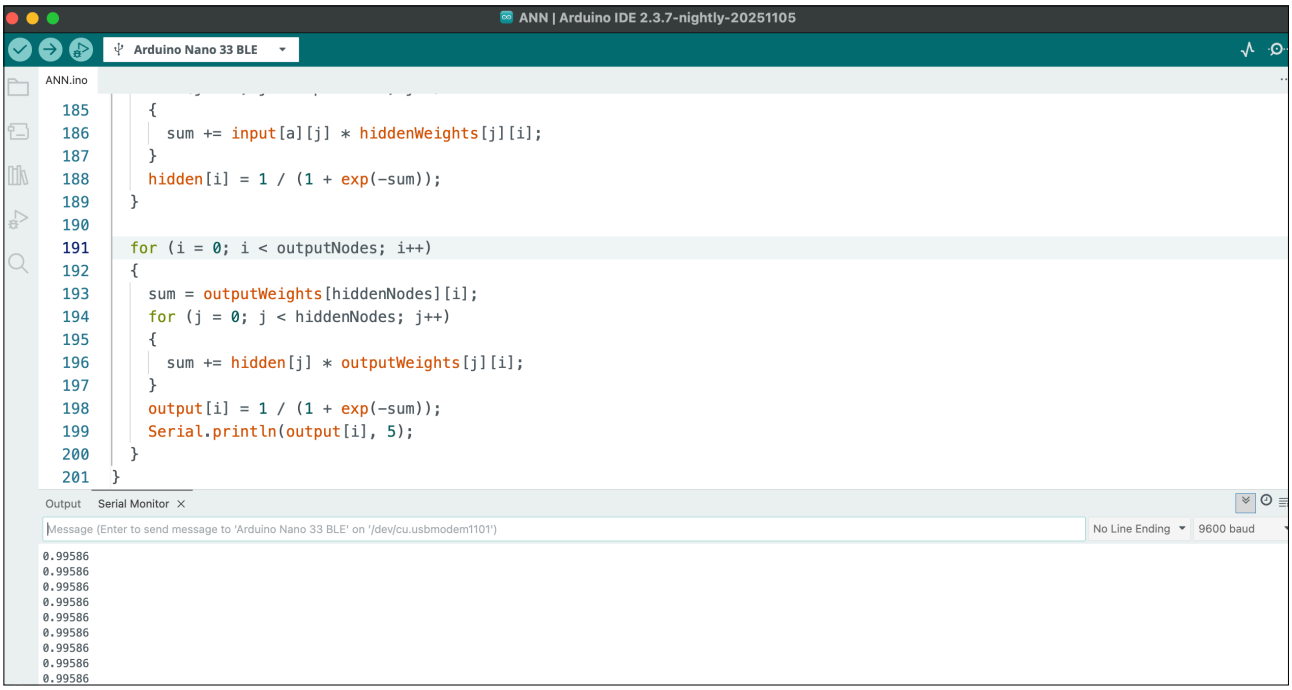


Figure D2.23b



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.