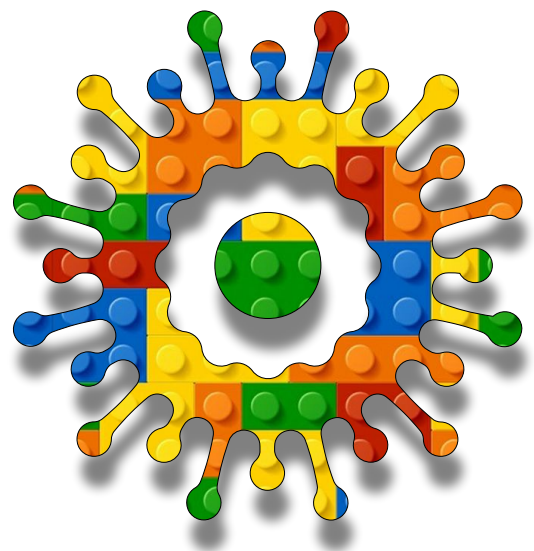


Algorithmic
Intelligence
Module A
Unit #9
cluster
classification





Module A Unit #9 cluster classification

Introduction to cluster classification

The index.html file

Sketch A9.1	starting sketch
Sketch A9.2	creating the model
Sketch A9.3	adding the options
Sketch A9.4	training data
Sketch A9.5	target label
Sketch A9.6	collect training data
Sketch A9.7	button
Sketch A9.8	training
Sketch A9.9	normalisation
Sketch A9.10	adding a bit of colour
Sketch A9.11	a state of mind
Sketch A9.12	classifying



Introduction to cluster classification with ml5.js

This is a supervised learning model. We are going to click on the canvas in four areas. In each quadrant, we will create a set of ten circles clustered together; each set of circles will be labelled **A**, **B**, **C**, and **D**. We will then train it with ml5.js, and when the training is complete, we will click on the canvas, and it will predict which lettered circle should be at that point on the canvas.

- ☞ This is a **classification** task.
- ☞ It will classify each circle as **A**, **B**, **C**, or **D**.



The index.html file

The `index.html` file needs the `ml5.js` library.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.11.1/p5.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.11.1/addons/p5.sound.min.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>
  </body>
</html>
```



Sketch A9.1 starting sketch

We have our bog-standard starting sketch.

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

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



Sketch A9.2 creating the model

Getting our neural network set up. We are going to use the `ml5.js` default `hyperparameter` settings for the moment.

```
let nn

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork()
}

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



Notes

This just checks that everything is working as it should when you run it.



Challenge

Think about giving your model a different name.



Code Explanation

<code>let nn</code>	We declare a variable name for our neural network
<code>ml5.setBackend("webgl")</code>	The <code>setBackend()</code> function allows this to run in all browsers, you could try "cpu" or even "gpu" depending on your machine
<code>nn = ml5.neuralNetwork()</code>	We define our neural network



Sketch A9.3 adding the options

We use the options to clarify what we are doing and what the inputs and outputs are. We tell it what task it is going to perform, **classification** or **regression**. We will define the inputs and outputs.

```
let nn

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")

  let options = {
    inputs: ['x', 'y'],
    outputs: ['label'],
    task: 'classification'
  }
  nn = ml5.neuralNetwork(options)
}

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



Notes

The **options** give you the possibility of fine-tuning the neural network. In this case, we will have two inputs, the **x** and **y** co-ordinates and four outputs, **A**, **B**, **C**, and **D**, which are the **labels**. Sometimes you don't even need to specify the inputs and outputs because the library can work it out anyway. It is important to specify the task, though. In this case, it is a **classification** task.



Code Explanation

<code>let options = {...}</code>	Defining the options
<code>inputs: ['x', 'y']</code>	The inputs are an array of x and y values
<code>outputs: ['label']</code>	The outputs are labels A, B, C or D
<code>task: 'classification'</code>	The task is a classification
<code>nn = ml5.neuralNetwork(options)</code>	The options are attributed to the neural network



Sketch A9.4 training data

We are going to create a `mousePressed()` function to collect training data. We need to move the `background()` into `setup()` and remove the `draw()` function because we will do all the drawing in the `mousePressed()` function. We will start with a default letter **A** and place it in a blue circle.

```
let nn

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    inputs: ['x', 'y'],
    outputs: ['label'],
    task: 'classification'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
}

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

function mousePressed()
{
  fill(0, 0, 255, 100)
  circle(mouseX, mouseY, 25)
  textAlign(CENTER, CENTER)
  stroke(0)
  text('A', mouseX, mouseY)
}
```

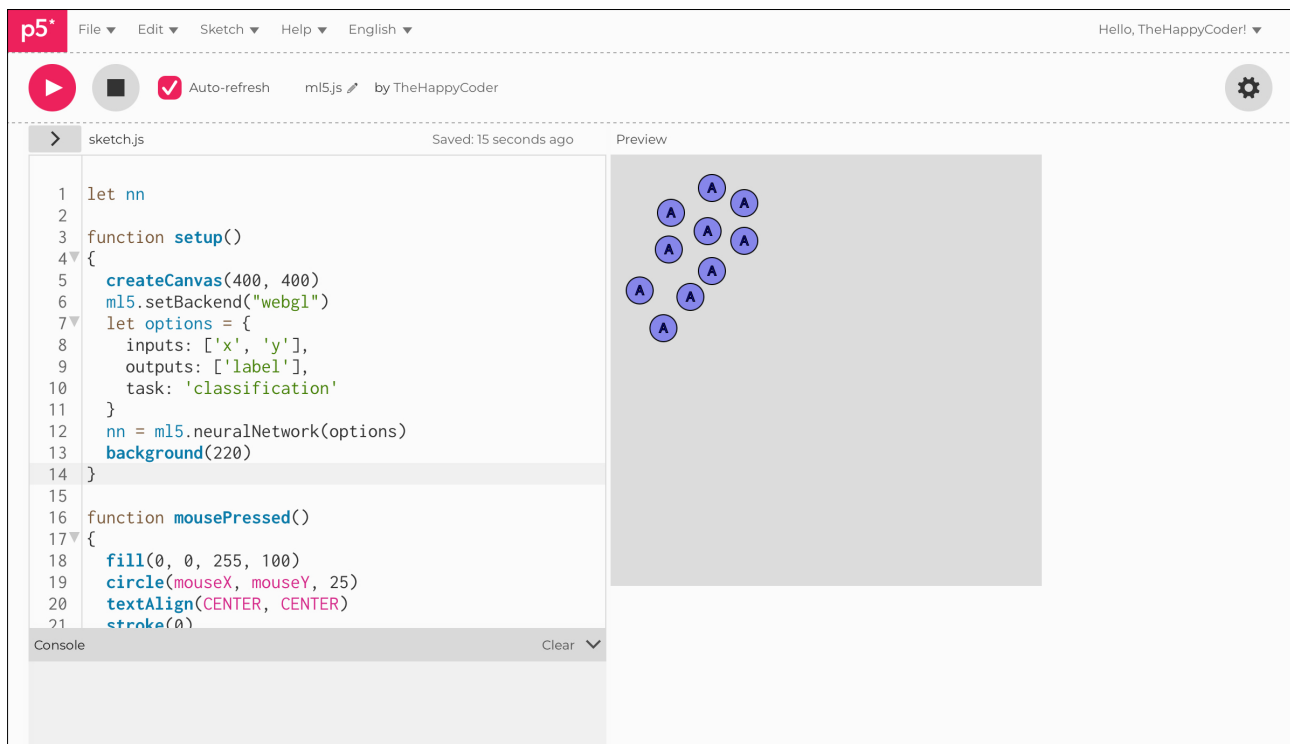
Notes

If you click on the canvas, you should get a circle with the letter **A** in the centre of it, like this. This will form part of your data collection. For the future sketches, there will be no `draw()` function.

Code Explanation

<code>function mousePressed()</code>	This function waits for the mouse to be pressed and then actions the code in the function.
<code>fill(0, 0, 255, 100)</code>	Gives a blue colour and the fourth argument is the alpha making it a little transparent
<code>textAlign(CENTER, CENTER)</code>	This puts the letter in the very centre of the circle
<code>text('A', mouseX, mouseY)</code>	We place the text at the same co-ordinates as the circle, at <code>mouseX</code> and <code>mouseY</code>

Figure A9.4



The screenshot shows the p5.js IDE interface. The top bar includes the p5 logo, menu items (File, Edit, Sketch, Help, English), and the user's name 'Hello, TheHappyCoder!'. Below the top bar, there are control buttons for play, stop, and auto-refresh, along with the file name 'ml5.js' and the author 'by TheHappyCoder'. The main workspace is split into two panes: 'sketch.js' on the left and 'Preview' on the right. The 'sketch.js' pane contains the following code:

```
1 let nn
2
3 function setup()
4 {
5   createCanvas(400, 400)
6   ml5.setBackend("webgl")
7   let options = {
8     inputs: ['x', 'y'],
9     outputs: ['label'],
10    task: 'classification'
11  }
12  nn = ml5.neuralNetwork(options)
13  background(220)
14 }
15
16 function mousePressed()
17 {
18   fill(0, 0, 255, 100)
19   circle(mouseX, mouseY, 25)
20   textAlign(CENTER, CENTER)
21   stroke(0)
```

The 'Preview' pane shows a gray canvas with several blue circles, each containing the letter 'A' in the center. The circles are arranged in a roughly circular pattern. At the bottom of the IDE, there is a 'Console' pane with a 'Clear' button.



Sketch A9.5 target label

We already have the letter **A** by default, but we want to collect three more data points with target labels **B**, **C**, and **D**, for this, we will need a keyboard input and the `keyPressed()` function. When you press the letter on the keyboard (**A**, **B**, **C**, or **D**), you activate that target `label` letter. The function `toUpperCase()` will make it uppercase for you, so you don't need to set Caps Lock; the function does it for you.

```
let nn
let targetLabel = 'A'

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    inputs: ['x', 'y'],
    outputs: ['label'],
    task: 'classification'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
}

function keyPressed()
{
  targetLabel = key.toUpperCase()
}

function mousePressed()
{
  fill(0, 0, 255, 100)
  circle(mouseX, mouseY, 25)
  textAlign(CENTER, CENTER)
  stroke(0)
  text(targetLabel, mouseX, mouseY)
}
```

Notes

Have about **10** (data points) per letter and try to cluster them together as illustrated in the image below. The target label has the default letter **A**, but to get the other target labels, press the **B**, **C**, or **D** keys first before clicking on the canvas.

Challenges

1. You could have any labels for your targets.
2. You could have more or less than the four suggested.

Code Explanation

<code>let targetLabel = 'A'</code>	Defines a variable for the target label and initialises it to A
<code>function keyPressed()</code>	This function waits for a key to be pressed before actioning the code in the function
<code>targetLabel = key.toUpperCase()</code>	Whatever key is pressed that is the new target label until another key is pressed, it capitalises any key pressed
<code>text(targetLabel, mouseX, mouseY)</code>	Now it puts the default key A unless another is pressed

Figure A9.5

The screenshot shows a p5.js IDE interface. The code editor on the left contains the following code:

```
1 let nn
2 let targetLabel = 'A'
3
4 function setup()
5 {
6   createCanvas(400, 400)
7   ml5.setBackend("webgl")
8   let options = {
9     inputs: ['x', 'y'],
10    outputs: ['label'],
11    task: 'classification'
12  }
13  nn = ml5.neuralNetwork(options)
14  background(220)
15 }
16
17 function keyPressed()
18 {
19   targetLabel = key.toUpperCase()
20 }
21
```

The preview window on the right displays a 400x400 canvas with a light gray background. It shows four distinct clusters of points, each represented by a small blue circle with a white letter inside. The top-left cluster contains 10 'A's, the top-right cluster contains 10 'B's, the bottom-left cluster contains 10 'C's, and the bottom-right cluster contains 10 'D's. The IDE interface includes a top menu bar with 'File', 'Edit', 'Sketch', 'Help', and 'English'. The top right corner shows the user name 'Hello, TheHappyCoder!'. The bottom left corner has a 'Console' area with a 'Clear' button.



Sketch A9.6 collect training data

Now we are going to do a bit of rearranging because we want to collect the data points every time we click on the canvas. To do that, we move the inputs and outputs object arrays into the `mousePressed()` function and then add the data to the neural network each time the mouse is clicked, inputting the target `label`. We use the ml5.js function `nn.addData()`.

```
let nn
let targetLabel = 'A'

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    // inputs: ['x', 'y'],
    // outputs: ['label'],
    task: 'classification'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
}

function keyPressed()
{
  targetLabel = key.toUpperCase()
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  let target = {
    label: targetLabel
  }
  nn.addData(inputs, target)
```

```
fill(0, 0, 255, 100)
circle(mouseX, mouseY, 25)
textAlign(CENTER, CENTER)
stroke(0)
text(targetLabel, mouseX, mouseY)
}
```



Notes

Just to clarify, every time you click and add a circle (with a target label letter) to the canvas, that data is passed to the neural network. You may notice that we didn't call the outputs by that name. This is because we created a new variable name (**target**) for the output.



Challenge

If you want to see the data produced, then add this line of code: **console.log(inputs, targetLabel)** at the end of the **mousePressed()** function. This is not essential but interesting (do one for each letter).



Code Explanation

```
nn.addData(inputs, target)
```

This function adds the data to the neural network. Inputs are the x and y co-ordinates and the output is called target which is the label (letter)



Sketch A9.7 button

We are going to use the data to train our model, but first we are going to create a button that, when we press it, we `train()` the model. But first, we will just get a console message to make sure the button is working.

```
let nn
let targetLabel = 'A'
let button

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    task: 'classification'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}

function keyPressed()
{
  targetLabel = key.toUpperCase()
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  let target = {
    label: targetLabel
  }
}
```

```

nn.addData(inputs, target)
fill(0, 0, 255, 100)
circle(mouseX, mouseY, 25)
textAlign(CENTER, CENTER)
stroke(0)
text(targetLabel, mouseX, mouseY)
}

```

```

function train()
{
  console.log('button working')
}

```

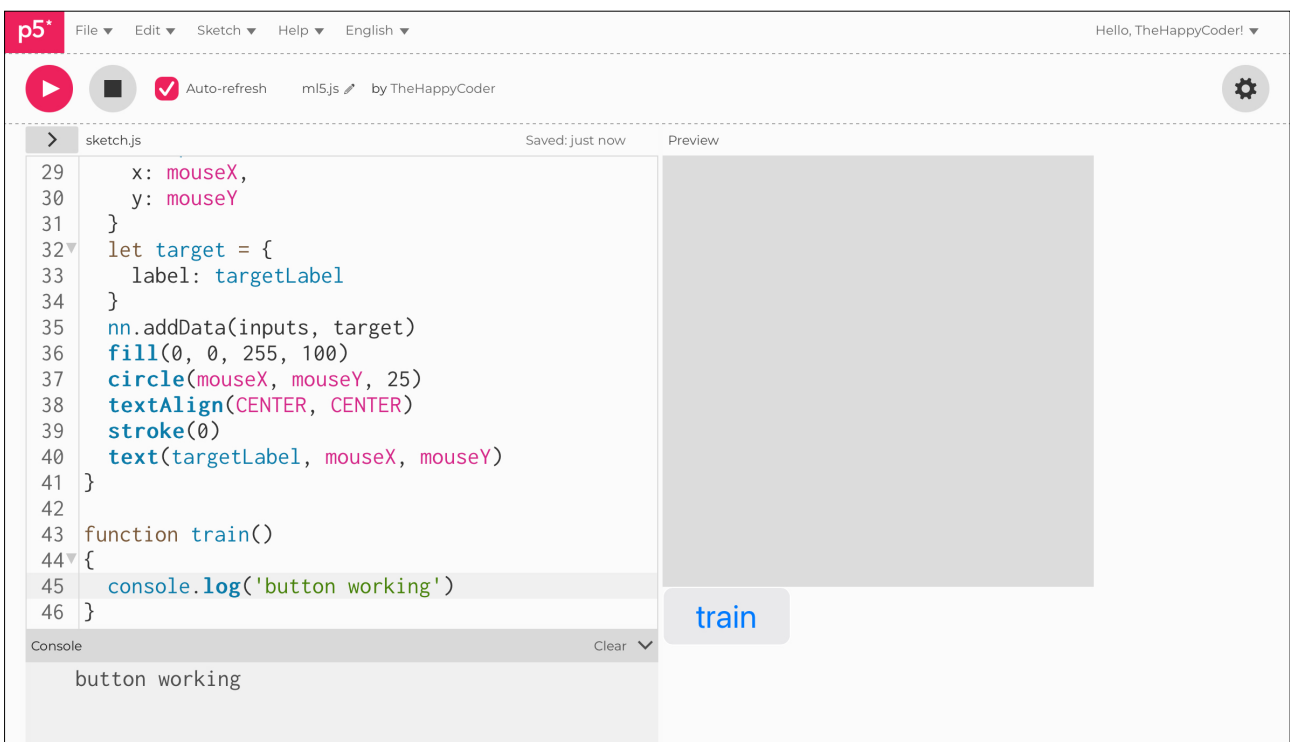
Notes

This just creates the button and checks to see if it is working, so you should get a message to that effect in the console. This is just temporary until we use the button to start the training process after we have our four sets of ten data points (target labels).

Code Explanation

<code>button.mousePressed(train)</code>	We activate the train() function when we click the button
<code>console.log('button working')</code>	This is where we are going to train our neural network

Figure A9.7





Sketch A9.8 training

If the button works OK, then remove the console log and add the `debug`. In the `train()` function, we start training the model. Add the data points (circles) for `A`, `B`, `C`, and `D` before clicking on the train button.

! We need to add a comma (,) after the word `'classification'`.

```
let nn
let targetLabel = 'A'
let button

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    task: 'classification',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}

function keyPressed()
{
  targetLabel = key.toUpperCase()
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  let target = {
```

```

    label: targetLabel
  }
  nn.addData(inputs, target)
  fill(0, 0, 255, 100)
  circle(mouseX, mouseY, 25)
  textAlign(CENTER, CENTER)
  stroke(0)
  text(targetLabel, mouseX, mouseY)
}

function train()
{
  nn.train()
}

```

Notes

When we do this, we get the following results, which are extremely poor. In this example, it was around 6 or 7, not good at all. There is a very good reason for this, and it is all to do with the size of the data points we have collected. The model works on the basis that all the values (including the inputs) are small; this is called **normalisation**. This was deliberate to illustrate what difference it makes and how essential it is. The neural network only really works with values between **0** and **1** (or **-1** and **+1**). We have **x** and **y** values in the region of up to **400**. We use the normalisation function **normalizeData()**.

Code Explanation

```
nn.train()
```

Instead of a console log we train the neural network instead

Figure A9.8

The screenshot shows a p5.js IDE interface. The top bar includes the p5 logo, menu items (File, Edit, Sketch, Help, English), and a user profile 'Hello, TheHappyCoder!'. Below the top bar are control buttons: a play button, a square button, a checked 'Auto-refresh' button, the filename 'ml5.js', and the author 'by TheHappyCoder'. A settings gear icon is on the right.

The main workspace is split into two panes. The left pane, titled 'sketch.js', contains the following code:

```
24
25 function mousePressed()
26 {
27   let inputs = {
28     x: mouseX,
29     y: mouseY
30   }
31   let target = {
32     label: targetLabel
33   }
34   nn.addData(inputs, target)
35   fill(0, 0, 255, 100)
36   circle(mouseX, mouseY, 25)
37   textAlign(CENTER, CENTER)
38   stroke(0)
39   text(targetLabel, mouseX, mouseY)
40 }
41
42 function train()
43 {
44   nn.train()
45 }
```

The right pane, titled 'Preview', contains a 'Visor' section with 'Maximize' and 'Hide' buttons. Below this is a 'Training Performance' graph. The graph has a title 'onEpochEnd' and shows a single data series named 'loss'. The y-axis is labeled 'Value' and ranges from 0 to 6. The x-axis is labeled 'Epoch' and ranges from 0 to 9. The 'loss' series is a horizontal blue line at a value of 6 across all epochs.



Sketch A9.9 normalisation

We need to improve things a little bit. Two things we are going to do: one is to normalise the data, which means putting the values between 0 and 1; for that, we use the `normalizeData()` function. The other thing is to specify the number of epochs. In the example from the training performance above, there were only ten epochs; we are increasing the number to 100. We also need a callback function: `finishedTraining()`.

```
let nn
let targetLabel = 'A'
let button

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    task: 'classification',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}

function keyPressed()
{
  targetLabel = key.toUpperCase()
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
}
```

```

let target = {
  label: targetLabel
}
nn.addData(inputs, target)
fill(0, 0, 255, 100)
circle(mouseX, mouseY, 25)
textAlign(CENTER, CENTER)
stroke(0)
text(targetLabel, mouseX, mouseY)
}

function train()
{
  nn.normalizeData()
  let options = {
    epochs: 100
  }
  nn.train(options, finishedTraining)
}

function finishedTraining()
{
  console.log('finished training')
}

```



Notes

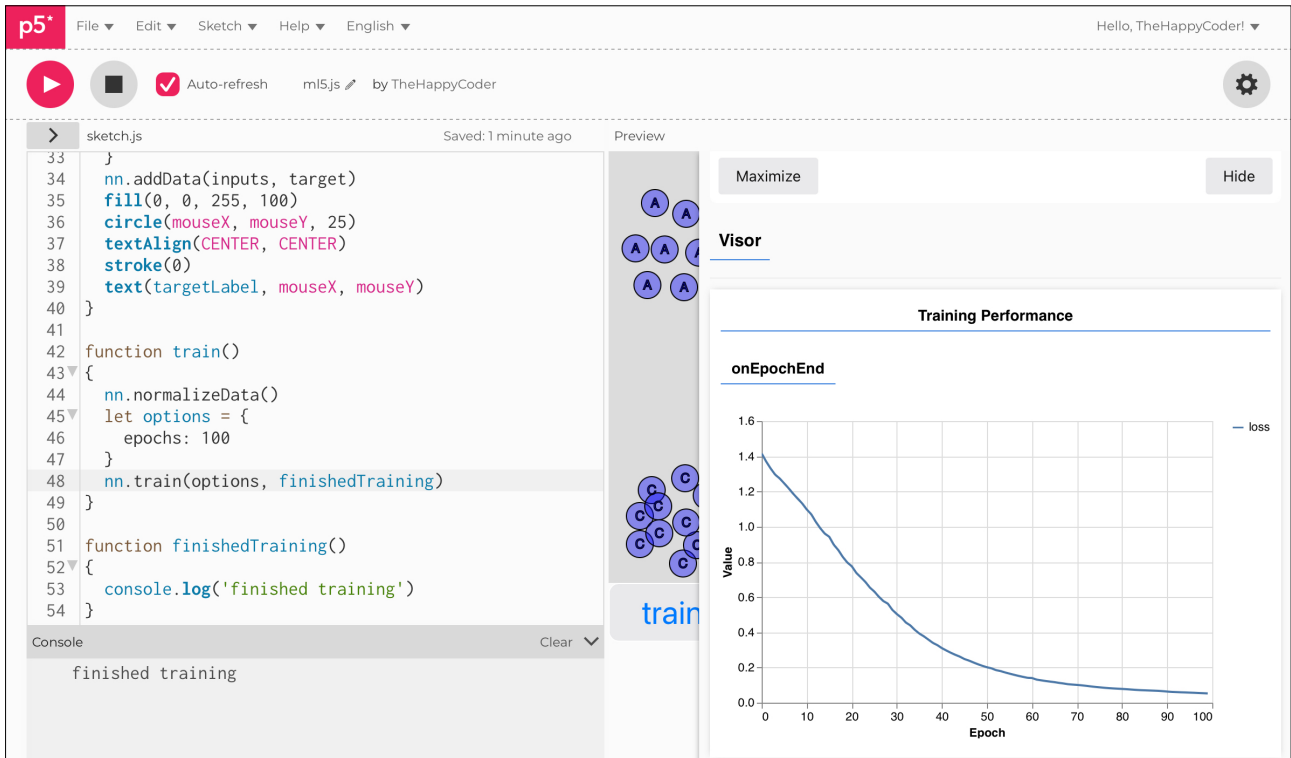
You will get a **finished training** comment in the console; also, the graph will look considerably better, very close to zero.



Code Explanation

<code>nn.normalizeData()</code>	Normalises the data between 0 and 1
<code>epochs: 100</code>	Go through the whole training data 100 times rather than just the 10

Figure A9.9





Sketch A9.10 adding a bit of colour

Before we go any further, I want to add a bit of colour to the proceedings. Each letter will have a given colour:

☐ A is blue

☐ B is red

☐ C is green

☐ D is yellow

This will give a more visual prediction later.

```
let nn
let targetLabel = 'A'
let button

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    task: 'classification',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}

function keyPressed()
{
  targetLabel = key.toUpperCase()
}

function mousePressed()
{
  let inputs = {
```

```

    x: mouseX,
    y: mouseY
  }
  let target = {
    label: targetLabel
  }
  nn.addData(inputs, target)
  if (targetLabel === 'A')
  {
    fill(0, 0, 255, 100)
  }
  if (targetLabel === 'B')
  {
    fill(255, 0, 0, 100)
  }
  if (targetLabel === 'C')
  {
    fill(0, 255, 0, 100)
  }
  if (targetLabel === 'D')
  {
    fill(255, 255, 0, 100)
  }

  circle(mouseX, mouseY, 25)
  textAlign(CENTER, CENTER)
  stroke(0)
  text(targetLabel, mouseX, mouseY)
}

function train()
{
  nn.normalizeData()
  let options = {
    epochs: 100
  }
  nn.train(options, finishedTraining)
}

```

```
function finishedTraining()
{
  console.log('finished training')
}
```

Notes

We have a separate colour for each cluster of data points; we just use several `if()` statements. The `===` means it has to be exactly that string.

Code Explanation

```
if (targetLabel === 'A')
```

If the label is A then use the following colour fill

Figure A9.10

The screenshot shows a p5.js IDE interface. The code editor on the left contains the following code:

```
49 fill(255, 255, 0, 100)
50 }
51 stroke(0)
52 circle(mouseX, mouseY, 25)
53 textAlign(CENTER, CENTER)
54 stroke(0)
55 text(targetLabel, mouseX, mouseY)
56 }
57
58 function train()
59 {
60   nn.normalizeData()
61   let options = {
62     epochs: 100
63   }
64   nn.train(options, finishedTraining)
65 }
66
67 function finishedTraining()
68 {
69   console.log('finished training')
70 }
```

The preview window on the right displays four clusters of data points, each labeled with a letter and colored accordingly: Cluster A (blue), Cluster B (red), Cluster C (green), and Cluster D (yellow). A 'train' button is located below the preview window.



Sketch A9.11 a state of mind

We create a simple variable expression called `state`; it helps us keep track of where we are in the process. We will have three states. The first state is to collect the data (`collection`), we then train the model on that data (`training`), and finally we predict what happens when we click somewhere on the canvas (`predicting`). In `mousePressed()`, we move a whole block of code inside an `if()` statement.

```
let nn
let targetLabel = 'A'
let button
let state = 'collection'

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    task: 'classification',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}

function keyPressed()
{
  targetLabel = key.toUpperCase()
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
}
```

```

if (state == 'collection')
{
  let target = {
    label: targetLabel
  }
  nn.addData(inputs, target)
  if (targetLabel === 'A')
  {
    fill(0, 0, 255, 100)
  }
  if (targetLabel === 'B')
  {
    fill(255, 0, 0, 100)
  }
  if (targetLabel === 'C')
  {
    fill(0, 255, 0, 100)
  }
  if (targetLabel === 'D')
  {
    fill(255, 255, 0, 100)
  }
  stroke(0)
  circle(mouseX, mouseY, 25)
  textAlign(CENTER, CENTER)
  stroke(0)
  text(targetLabel, mouseX, mouseY)
}
}

function train()
{
  state = 'training'
  nn.normalizeData()
  let options = {
    epochs: 100
  }
  nn.train(options, finishedTraining)
}

```

```
}  
  
function finishedTraining()  
{  
  console.log('finished training')  
  state = 'prediction'  
}
```



Notes

A bit of refactoring, not really adding lots of new code. The change of state is something like changing the mode setting on a piece of equipment.



Code Explanation

<code>let state = 'collection'</code>	Define and initialise variable string to collection
<code>state = 'training'</code>	Change the string to training
<code>state = 'prediction'</code>	Change again to prediction



Sketch A9.12 classifying

The final stage is the prediction or classification. We want to predict what every pixel of the canvas would be if you were to create a new data point. We create a new callback function called `gotResults()` but only after we have run the classifier.

When you have trained the model, click on the canvas (anywhere) and as you move the mouse around, you will start colouring in the predicted colour for that particular coordinate on the canvas. We are filling the circle with the predicted colour at that coordinate on the canvas.

```
let nn
let targetLabel = 'A'
let button
let state = 'collection'

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    task: 'classification',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}

function keyPressed()
{
  targetLabel = key.toUpperCase()
}

function mousePressed()
{
  let inputs = {
```

```

    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let target = {
      label: targetLabel
    }
    nn.addData(inputs, target)
    if (targetLabel === 'A')
    {
      fill(0, 0, 255, 100)
    }
    if (targetLabel === 'B')
    {
      fill(255, 0, 0, 100)
    }
    if (targetLabel === 'C')
    {
      fill(0, 255, 0, 100)
    }
    if (targetLabel === 'D')
    {
      fill(255, 255, 0, 100)
    }
    stroke(0)
    circle(mouseX, mouseY, 25)
    textAlign(CENTER, CENTER)
    stroke(0)
    text(targetLabel, mouseX, mouseY)
  }
  else if (state == 'prediction')
  {
    nn.classify(inputs, gotResults)
  }
}

function train()

```

```
{
  state = 'training'
  nn.normalizeData()
  let options = {
    epochs: 100
  }
  nn.train(options, finishedTraining)
}
```

```
function finishedTraining()
{
  console.log('finished training')
  state = 'prediction'
}
```

```
function gotResults(results)
{
  mousePressed()
  if (results[0].label === 'A')
  {
    fill(0, 0, 255, 100)
  }
  if (results[0].label === 'B')
  {
    fill(255, 0, 0, 100)
  }
  if (results[0].label === 'C')
  {
    fill(0, 255, 0, 100)
  }
  if (results[0].label === 'D')
  {
    fill(255, 255, 0, 100)
  }
  noStroke()
  circle(mouseX, mouseY, 25)
}
```

Notes

Each coordinate will have a predicted colour value depending on where that particular pixel point is. The results will classify it and draw the corresponding coloured circle. Think of it as filling in all the blanks where you had no data; it has interpreted the rest of the canvas for you.

Challenge

Try different shapes of data points e.g a swirl, concentric circles (you may need a lot more data points). Also, try drawing pixels instead of circles (although it might take longer to fill the canvas).

Code Explanation

<pre>nn.classify(inputs, gotResults)</pre>	The classify() function takes the inputs and returns the results whether it is A, B, C or D
<pre>if (results[0].label === 'A')</pre>	It makes the prediction for that x, y value input and checks to see whether it is A, B, C or D and colours appropriately

Figure A9.12

```
82 function gotResults(results)
83 {
84   mousePressed()
85   if (results[0].label === 'A')
86   {
87     fill(0, 0, 255, 100)
88   }
89   if (results[0].label === 'B')
90   {
91     fill(255, 0, 0, 100)
92   }
93   if (results[0].label === 'C')
94   {
95     fill(0, 255, 0, 100)
96   }
97   if (results[0].label === 'D')
98   {
99     fill(255, 255, 0, 100)
100  }
101  noStroke()
102  circle(mouseX, mouseY, 25)
103 }
```

finished training