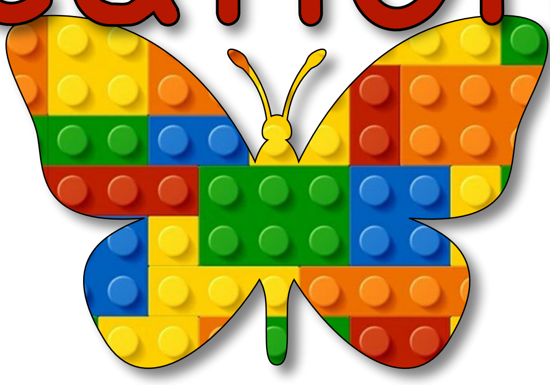
Artificial Intelligence Module A Unit #4 mouse gesture classification



Content

Module A Unit #4 mouse gesture classification

Introduction to mouse gesture with ml5.js

The two examples so far have been `regression` tasks so now we need a `classification` task. We will try to identify which way the mouse is moving either up, down, left or right. First we will create some synthetic data to train the model on.

The model is then trained on this data over a number of epochs until we are happy with the result. We are going to be careful that there is no underfitting or overfitting.

When the model is trained we can test it by moving the mouse in each direction to see how well it performs. Remember that this is a relatively simple example and has many drawbacks and omissions but it demonstrates a simple `classification` task. We will be using ml5.js as before so make sure that you have the line of code in the index.html file.

Sketch A4.1 our starting sketch

We start with our basic sketch

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

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

The data

We are using synthetic data once again. This time rather than generating it we will hard code it. You will notice that it is an `array` of `objects`. Each object has three elements:

- the x component of a vector,
- the y component of the vector and
- the label indicating which direction it is going in.

You may notice that the units are between `0` and `1`. We have effectively normalised the data already so no need to do it again. We have two sets of data for each movement `left`, `right`, `up` and `down`. This is a very small dataset but we will see how well it does once we start training it.

The data is a vector which is the amount the mouse has moved from the relative position of (`0, 0`).

Sketch A4.2 adding the data

Add in the data as shown below. I have kept it very simple and very obvious. Either +1, −1, +0.1, −0.1 depending on the relevant direction.

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

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

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

Notes

I hope this is seems fairly straight forward. The format is for a json type array.

🌻 Challenges

1. You could adjust some of the values, making them more random, so that they are not so obvious
2. Increase the size of the dataset
3. You could think about how would you collect and then save the data to be loaded (maybe for another time).

🛠️ Code Explanation

`let data = [`	Create an array of objects
`{x: 1, y: 0.1, label: "right"},`	This is an object with two vectors and a label, this moves the coordinates to the right and slightly downwards
`{x: -0.1, y: -1, label: "up"}`	This is an object with two vectors and a label, this moves the coordinates slightly to the left and upwards

Sketch A4.3 building the model

You should be familiar with building the model now, we are going to create a neural network model and call it nn. We will give the neural network the following options:

- It is a classification task
- Set debug to true which will show the progress of the training (you can set it to false later)

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

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    task: "classification",
    debug: true
  }
  nn = ml5.neuralNetwork(options)
}

function draw()
```

```
{
  background(220)
}
```

Notes

We are building the model just we have done before

Sketch A4.4 adding the data training the model

The `for()` loop (`let items of data`) will pull all the datapoints in the `data` array into another array called `items`. We can then create an array of inputs based on the `x` and `y` values. The output array then can collect all the labels that go with those input vectors. We then add this dataset to the neural network model `nn.addData(inputs, outputs)`.

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


function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    task: "classification",
    debug: true
  }
  nn = ml5.neuralNetwork(options)
  for (let item of data)
  {
    let inputs = [item.x, item.y]
    let outputs = [item.label]
```

```
    nn.addData(inputs, outputs)
  }
}

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

🗒️ Notes

It pulls all the x, y and label values from the array and adds this data to the neural network model (nn). This is similar to the regression tasks but there is a slightly different approach for classification tasks.

Sketch A4.5 training the model

After we have added the data we are going to train it with nn.train(), with a callback finishedTraining which will let us know when it has finished. The callback is a function, it will help us to keep track of what is happening. To help us at this stage we will console log the status. The default is training and when it has finished training the status will change to ready

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

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  let options = {
    task: "classification",
    debug: true
  }
  nn = ml5.neuralNetwork(options)
  for (let item of data)
  {
    let inputs = [item.x, item.y]
```

```
    let outputs = [item.label]
    nn.addData(inputs, outputs)
  }
  nn.train(finishedTraining)
}

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

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

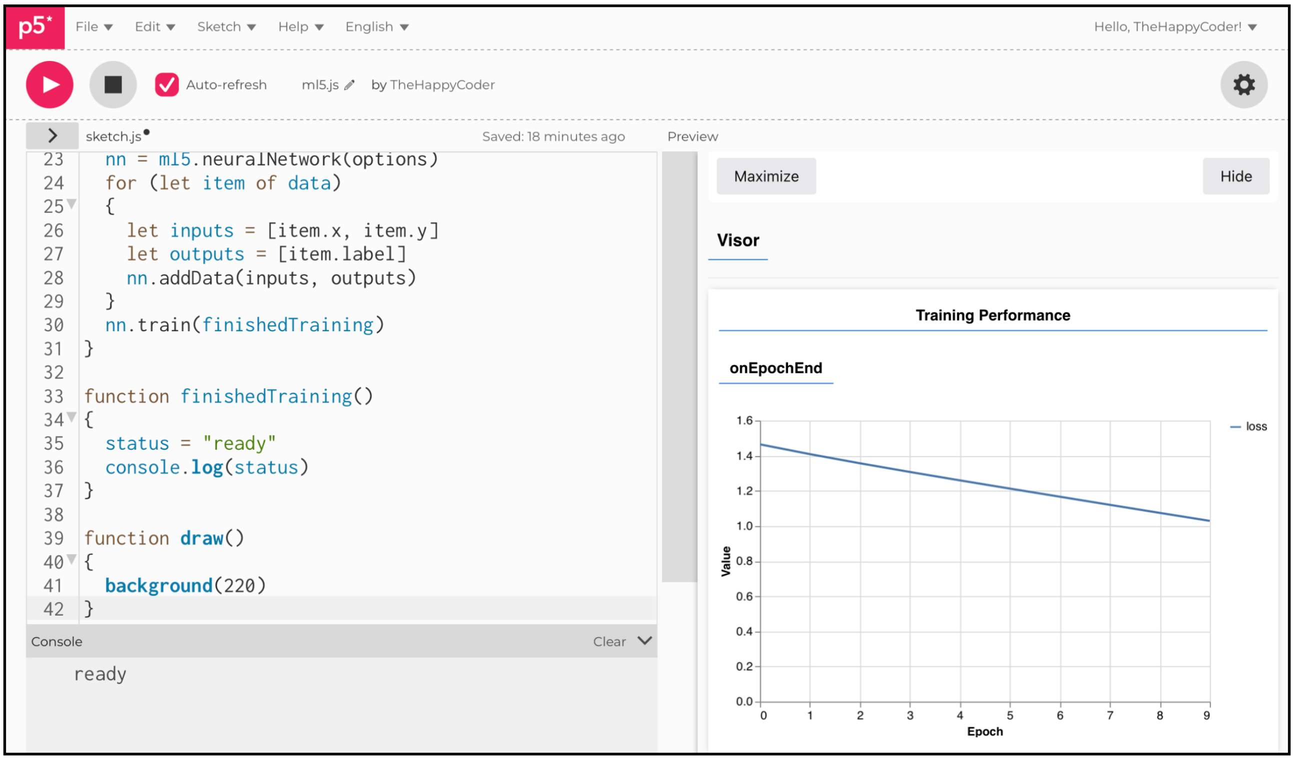
🗒️ Notes

The `status` should go from `training` to `ready` once it has finished training (you won't see the word training). We also can see that `10` epochs are nowhere near enough, se we need to increase that `hyperparameter`

🛠️ Code Explanation

`let status = "training"`	This is a string variable that is initialised to "training"
`status = "ready"`	The string variable value is now "ready"

Figure A4.5


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 18 minutes ago
Preview
nn = ml5.neuralNetwork(options)
for (let item of data)
{
let inputs = [item.x, item.y]
let outputs = [item.label]
nn.addData(inputs, outputs)
}
nn.train(finishedTraining)
}
function finishedTraining()
{
status = "ready"
console.log(status)
}
function draw()
{
background(220)
}
Console
Clear
ready
Maximize
Hide
Visor
Training Performance
onEpochEnd
loss
Value
Epoch
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
7
8
9

Sketch A4.6 epochs

Clearly the loss function was still going down so we will try 250 epochs and see if that works. We can just add it straight in to the nn.train() function, just a shorthand formatting version.

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

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

```
  }
  nn.train({epochs: 250}, finishedTraining)
}

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

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

🗒️ Notes

You will see that was still going down even after 250 epochs so may continue to reduce, however it is probably overfitting after say 100 epochs. The reason for such a high number of epochs compared to our other examples could be that it is a tiny dataset. If you move your mouse over the chart it gives you the value of the loss function. My effort was 0.009, which is pretty low.

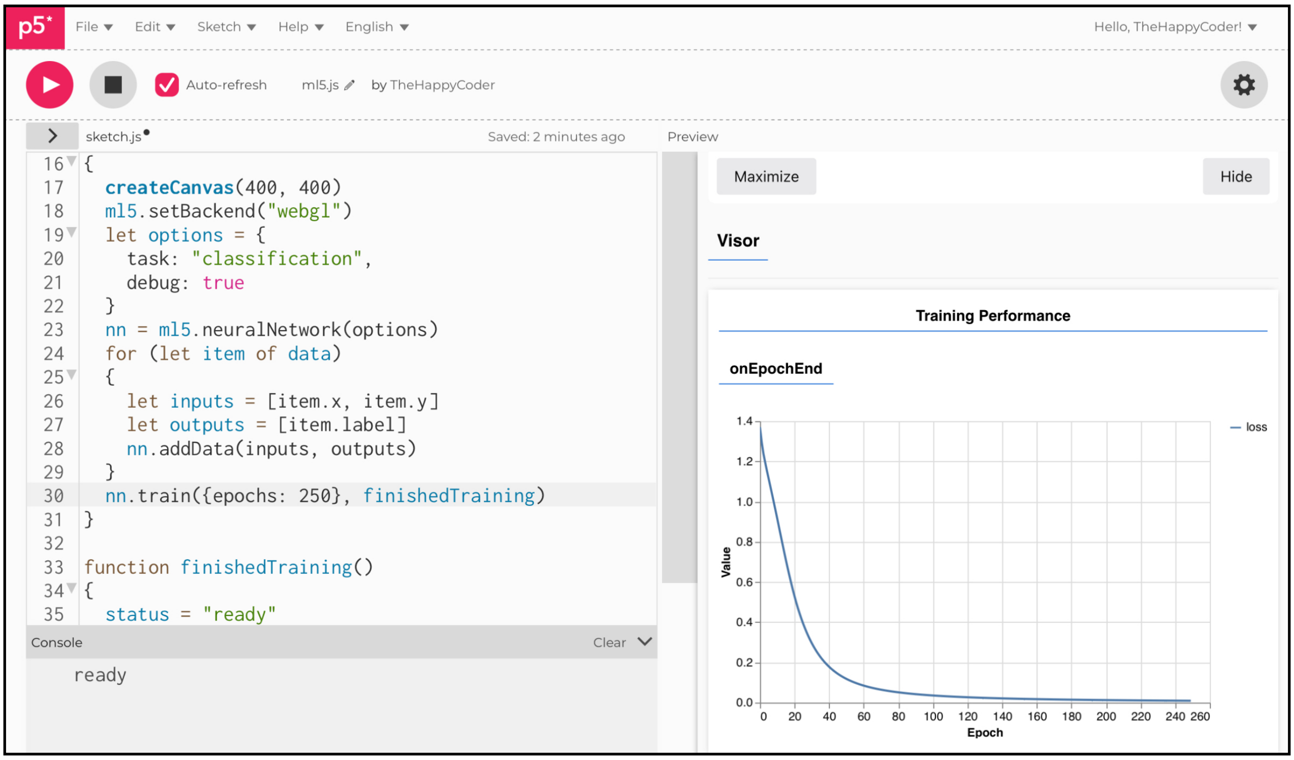
🌻 Challenge

Try an even larger number of epochs.

🛠️ Code Explanation

`nn.train({epochs: 250}, finishedTraining)`	Specifying the number of epochs within the training fucntion

Figure A4.6


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 2 minutes ago
Preview
{
createCanvas(400, 400)
ml5.setBackend("webgl")
let options = {
task: "classification",
debug: true
}
nn = ml5.neuralNetwork(options)
for (let item of data)
{
let inputs = [item.x, item.y]
let outputs = [item.label]
nn.addData(inputs, outputs)
}
nn.train({epochs: 250}, finishedTraining)
}
function finishedTraining()
{
status = "ready"
Console
Clear
ready
Maximize
Hide
Visor
Training Performance
onEpochEnd
loss
Value
Epoch
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0 20 40 60 80 100 120 140 160 180 200 220 240 260

Sketch A4.7 mouse vectors

What we want to do now is move the mouse in such a way that we can use the model to predict what movement it has made, either up, down, left or right. So we need two variables for the start and end of the mouse movement. The movement starts when the mouse is clicked and keeps going while it is dragged (and then stopped dragging). So we have two vectors for the start and for the end.

```
let nn
let status = "training"
let start
let end
let data = [
  { x: 1, y: 0.1, label: "right"},
  { x: 1, y: -0.1, label: "right"},
  { x: -1, y: 0.1, label: "left"},
  { x: -1, y: -0.1, label: "left"},
  { x: 0.1, y: 1, label: "down"},
  { x: -0.1, y: 1, label: "down"},
  { x: 0.1, y: -1, label: "up"},
  { x: -0.1, y: -1, label: "up"}
]


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

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

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

function draw()
{
  background(220)
}

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

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

Notes

Nothing will happen just yet, we just collecting data from the mouse

🛠️ Code Explanation

`start = createVector(mouseX, mouseY)`	We create a vector (called start) as soon as we click on the canvas
`end = createVector(mouseX, mouseY)`	A final vector is created as we drag the mouse across the canvas

Sketch A4.8 classifying the mouse

We create another function to input the data into the model (which has been trained on the synthetic dataset. This function is called when the mouse is released after it has finished dragging. The key elements are described below:

- The direction (dir) of the movement is done by subtracting the two vectors (end and start)
- We normalise them so their magnitudes are less than 1
- We then get the x and y components from the direction (dir.x, dir.y) vector
- We then have these as our new inputs to classify as either up, down, left or right and put them into the model.
- The classification takes two arguments, one is the inputs and the other gotResults is the output.

```
let nn
let status = "training"
let start
let end
let data = [
  { x: 1, y: 0.1, label: "right"},
  { x: 1, y: -0.1, label: "right"},
  { x: -1, y: 0.1, label: "left"},
  { x: -1, y: -0.1, label: "left"},
  { x: 0.1, y: 1, label: "down"},
  { x: -0.1, y: 1, label: "down"},
  { x: 0.1, y: -1, label: "up"},
  { x: -0.1, y: -1, label: "up"}
]


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

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

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

function draw()
{
  background(220)
}

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

function mouseDragged()
{
```

```
  end = createVector(mouseX, mouseY)
}

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

🗒️ Notes

❗ please note you will get a script error if you run this

When we `classify` the movement of the mouse we give it the inputs (`dir.x`, `dir.y`) plus a callback. This callback is a function which will carry the result, next we need to create a function called, you guessed it, `gotResults()` to make use of the result.

🛠️ Code Explanation

`let dir = p5.Vector.sub(end, start)`	We subtract the two vectors, end and start
`dir.normalize()`	The subtraction of those two vectors (called dir) is normalised to be between 0 and 1
`let inputs = [dir.x, dir.y]`	The inputs into the classify function are the x and y components of the vector
`nn.classify(inputs, gotResults)`	We give the inputs to the classify function and also give it a callback (gotResults)

Sketch A4.9 getting the results

As we create the callback function gotResults() we can see how well we are doing by putting the results in the console for now. The status changes from training to ready and now it is expressed as one of the labels left, right, up or down.

❗ we will change the debug to false or remove altogether.

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

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

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

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

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

function draw()
{
  background(220)
}

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

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

function mouseReleased()
{
  let dir = p5.Vector.sub(end, start)
```

```
  dir.normalize()
  let inputs = [dir.x, dir.y]
  nn.classify(inputs, gotResults)
}

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

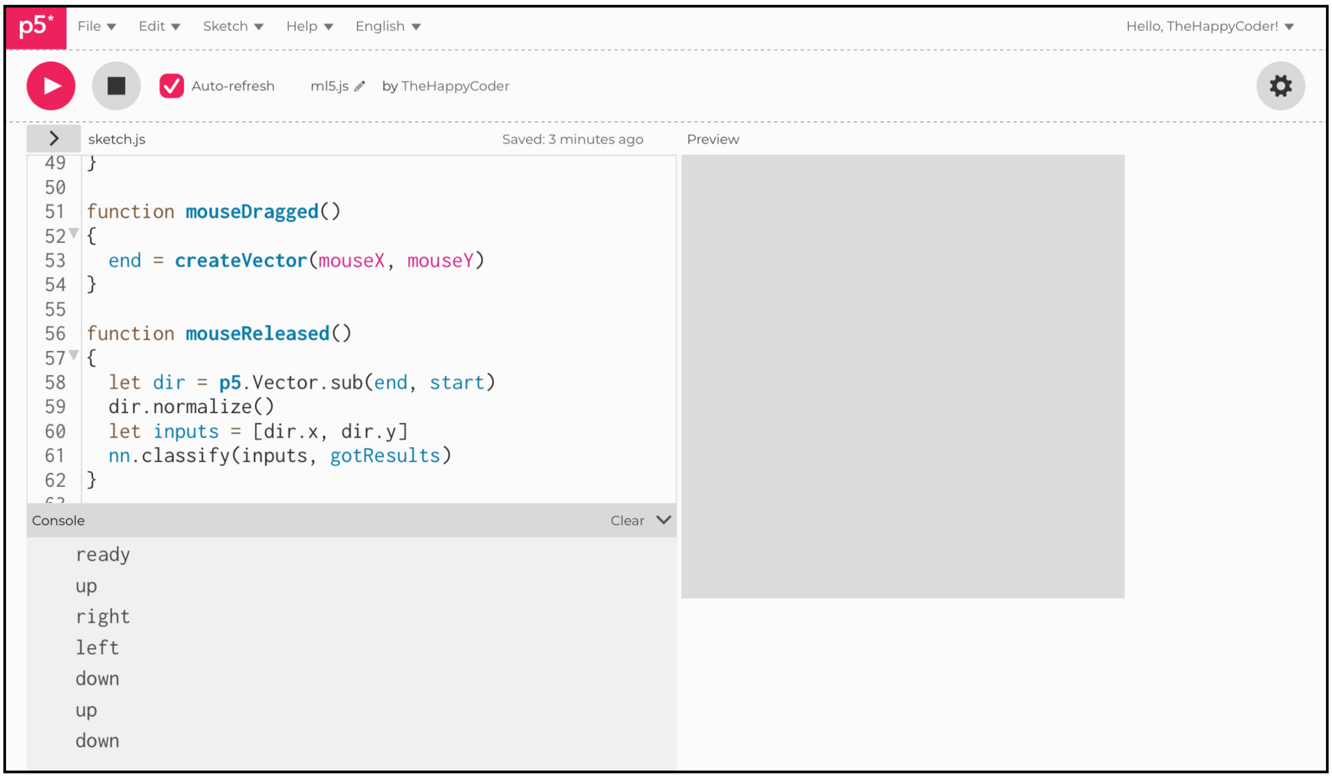
🗒️ Notes

This seems to work quite well, remember to hold the button down as you move the mouse and when you release the button you should get the correct movement.

🛠️ Code Explanation

`status = results[0].label`	We can now see the result of the mouse movement (drag) in the console

Figure A4.9



p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 3 minutes ago
Preview
}
function mouseDragged()
{
end = createVector(mouseX, mouseY)
}
function mouseReleased()
{
let dir = p5.Vector.sub(end, start)
dir.normalize()
let inputs = [dir.x, dir.y]
nn.classify(inputs, gotResults)
}
Console
Clear
ready
up
right
left
down
up
down

Sketch A4.10 displaying the results

We want to see this in action. We want the result on the canvas and also draw a line showing the movement of the mouse. We will do all this in the `draw()` function.

❗ Comment out the console logs

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

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

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

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

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

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

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

function mouseDragged()
```

```
{
  end = createVector(mouseX, mouseY)
}

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

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

🗒️ Notes

We don't need the console logs any more as we are writing straight to the canvas

🌻 Challenge

Add some colour or other event

🛠️ Code Explanation

`textAlign(CENTER, CENTER)`	Puts the complete text in the centre of the text co-ordinates
`textSize(64)`	Nice and big text
`text(status, width/2, height/2)`	The status has the string values
`if (start && end)`	Condition, draws the line when we have a start AND end vector
`line(start.x, start.y, end.x, end.y)`	Draw the line to those two vectors

Figure A4.10a we are training

Figure A4.10b we are ready (training complete)

Figure A4.10c move and release upwards

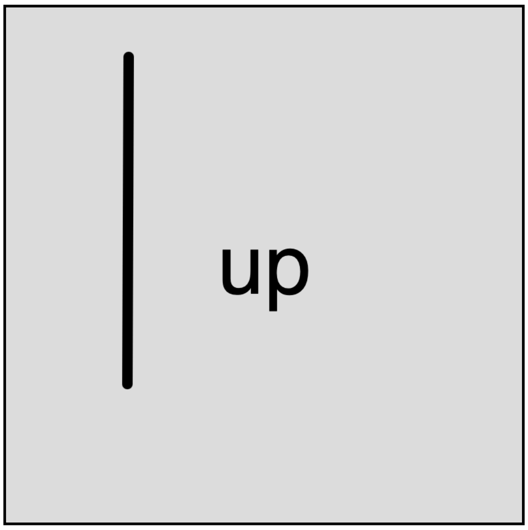


Figure A4.10d and then down

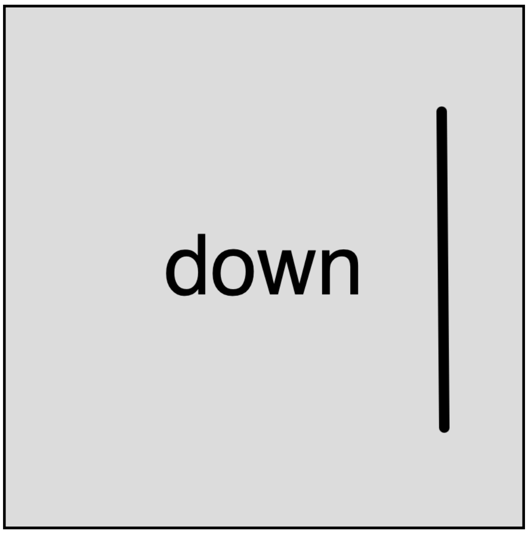


Figure A4.10e move to the left

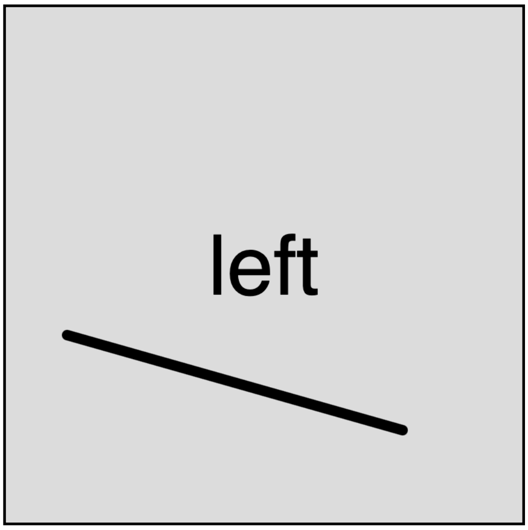


Figure A4.10f and now to the right