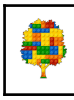
Module 3
Regression
ml5.js



Content

Module 3: Regression

Introduction to module 3 Regression

In this module we will describe a model for regression. This is very similar to the classification one you have just done. The main difference is that the outcome (output) is a value between 0 and 255, whereas the classification the result was either A, B, C or D (or colour equivalent)

It might just learn the data it is give to train on and be useless on data it hasn't seen before. Too small and feeble a model can produce poor results even if it is quick to learn. There has to be a compromise, a sweet spot and that is the skill of the AI engineer (you).

The problems are called 'under-fitting' and 'over-fitting'. They are always easy to spot but this is when AI Engineering is more of an art than a science which sounds silly saying it but true, there is quite a lot of trial and error.

This part looks at regression which has many similarities to classification except when it comes to the output, where it is a value rather than a label.

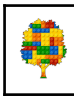
Sketch 3.1 index.html file

The index.html needs the extra line of code

<script src="https://unpkg.com/ml5@1/dist/ml5.min.js"></script>

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.10.0/p5.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.10.0/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 3.2 basic starting model

Starting a new sketch. We are creating the basic model, which is very similar to the classification previously. There are two inputs (x, y), and one output (val, which is the grey value between 0 and 255). The task is 'regression', not 'classification'. The debug will show us the progress (remember to click hide if it hides the canvas when finished training).

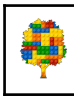
sketch.js

```
let nn

function setup()
{
  createCanvas(400, 400)
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
}
```

Notes

The main points here are

1. The task is now regression
2. The inputs are still x, y
3. The output is a value 'val'

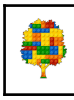


Sketch 3.3 assigning a value

We want to assign a value to the circle (rather than a label). To keep it simple we are using 'w' for white (giving it a value of 255) and 'b' for black (assigning a value of 0). These are the fill colours (and values), they will be the data we train the model on.

sketch.js

```
let nn
let targetLabel = 'w'
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
}
```



Sketch 3.4 assigning the keys

Adding the keyPressed function. Notice that I have omitted the capitalising of the letters

sketch.js

```
let nn
let targetLabel = 'w'
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
}

function keyPressed()
{
  targetLabel = key
}
```

Sketch 3.5 collecting data

Adding the mousePressed() function to collect the data. Notice I have assigned the colour rather than the letter and make the circles smaller. You still need to press w and b.

sketch.js

```
let nn
let targetLabel = 'w'
let state = 'collection'
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
}

function keyPressed()
{
  targetLabel = key
}
```

```
function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let targetVal = col[targetLabel]
    let target = {
      val: targetVal
    }
    nn.addData(inputs, target)
    fill(targetVal)
    circle(mouseX, mouseY, 25)
  }
}
```

Sketch 3.6 training button

Let us create a button to press to train the model

sketch.js

```
let nn
let targetLabel = 'w'
let state = 'collection'
let button
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}

function keyPressed()
{
```

```
  targetLabel = key
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let targetVal = col[targetLabel]
    let target = {
      val: targetVal
    }
    nn.addData(inputs, target)
    fill(targetVal)
    circle(mouseX, mouseY, 25)
  }
}

function train()
{

}
```

Sketch 3.7 training

Now to add in the code for training the model

sketch.js

```
let nn
let targetLabel = 'w'
let state = 'collection'
let button
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}

function keyPressed()
{
```

```
  targetLabel = key
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let targetVal = col[targetLabel]
    let target = {
      val: targetVal
    }
    nn.addData(inputs, target)
    fill(targetVal)
    circle(mouseX, mouseY, 25)
  }
}

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

Sketch 3.8 training functions

We need to add the functions whileTraining() and finishedTraining()

sketch.js

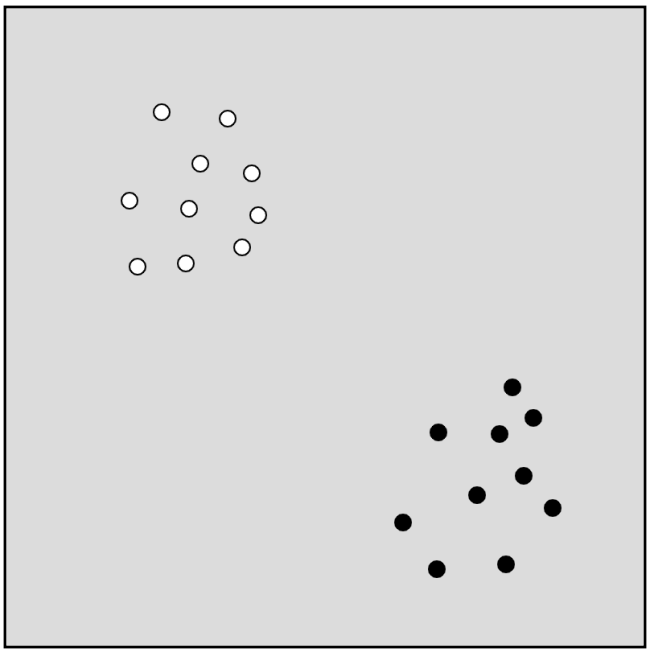
```
let nn
let targetLabel = 'w'
let state = 'collection'
let button
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}

function keyPressed()
{
```

```
  targetLabel = key
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let targetVal = col[targetLabel]
    let target = {
      val: targetVal
    }
    nn.addData(inputs, target)
    fill(targetVal)
    circle(mouseX, mouseY, 25)
  }
}

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

function whileTraining(epochs, loss)
{
```

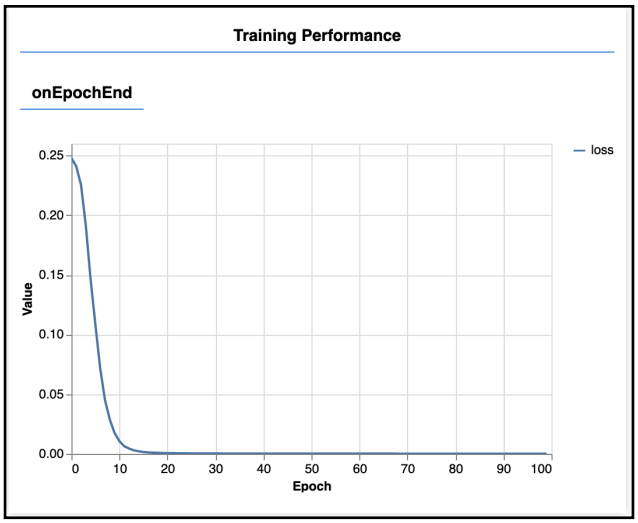
```
  console.log(epochs)
}

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

Step 1: Click up to 10 times top left of canvas
Step 2: Press the 'b' key and then click same number of times in the bottom right corner
Step 3: Click on the train button and you will see the following

I click 10 times with the 'w' white and
then ten times with the 'b' black.
Then I clicked on the train button at the
bottom of the canvas



The loss chart



Sketch 3.9 epochs

Notice that the training pretty well reaches zero after around 10 epochs. So we can change the number of them to 10

sketch.js

```
let nn
let targetLabel = 'w'
let state = 'collection'
let button
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}

function keyPressed()
{
```

```
  targetLabel = key
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let targetVal = col[targetLabel]
    let target = {
      val: targetVal
    }
    nn.addData(inputs, target)
    fill(targetVal)
    circle(mouseX, mouseY, 25)
  }
}

function train()
{
  state = 'training'
  nn.normalizeData()
  let options = {
    epochs: 10
  }
  nn.train(options, whileTraining, finishedTraining)
}

function whileTraining(epochs, loss)
{
```

```
  console.log(epochs)
}

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

Sketch 3.10 predicting

The next part is to predict the results. What colour will any circles be in between the data sets of values. We want a grey scale of colour not just black or white because that is classification. Regression gives us a sliding scale of values.

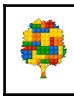
We change the state to prediction

sketch.js

```
let nn
let targetLabel = 'w'
let state = 'collection'
let button
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
```

```
}

function keyPressed()
{
  targetLabel = key
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let targetVal = col[targetLabel]
    let target = {
      val: targetVal
    }
    nn.addData(inputs, target)
    fill(targetVal)
    circle(mouseX, mouseY, 25)
  }
  else if (state == 'prediction')
  {
    nn.predict(inputs, gotResults)
  }
}

function train()
{
  state = 'training'
  nn.normalizeData()
```

```
  let options = {
    epochs: 10
  }
  nn.train(options, whileTraining, finishedTraining)
}

function whileTraining(epochs, loss)
{
  console.log(epochs)
}

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



Sketch 3.11 prediction results

We need to add in the function gotResults() so we can see the results (predictions), after training the data will disappear (reset the background()) and when you click on the canvas you will get the trained values for that circle.

sketch.js

```
let nn
let targetLabel = 'w'
let state = 'collection'
let button
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}
```

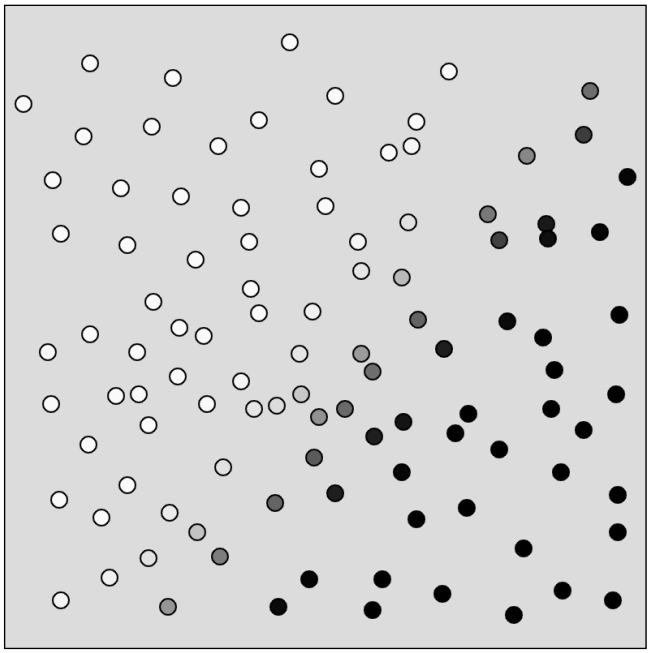
```
function keyPressed()
{
  targetLabel = key
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let targetVal = col[targetLabel]
    let target = {
      val: targetVal
    }
    nn.addData(inputs, target)
    fill(targetVal)
    circle(mouseX, mouseY, 25)
  }
  else if (state == 'prediction')
  {
    nn.predict(inputs, gotResults)
  }
}

function train()
{
  state = 'training'
  nn.normalizeData()
  let options = {
    epochs: 10
```

```
  }
  nn.train(options, whileTraining, finishedTraining)
}

function whileTraining(epochs, loss)
{
 // console.log(epochs)
}

function finishedTraining()
{
  console.log('finished training')
  state = 'prediction'
  background(220)
}

function gotResults(results)
{
  fill(floor(results[0].value))
  circle(mouseX, mouseY, 25)
}
```

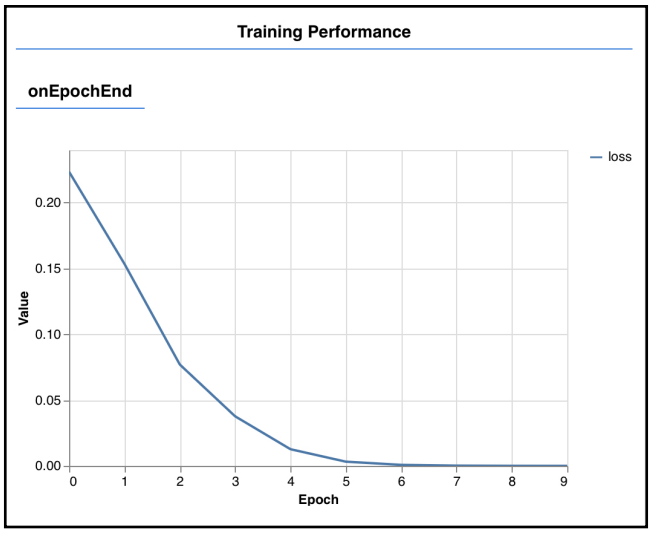
Notes

As before, have a bunch of white dots and black dots, train it, when the training is finished it will reset the background, now click anywhere on the canvas you will get different grayscale dots. This is the difference between classification and regression. Classification gives you a result of one classification, whereas regression will give you a value.

The results after the training



Loss chart after training



Some refinements

The changes are highlighted in blue, also. This will keep the original circles and their colour but now when you click on the canvas it will draw the shades of grey on the canvas. We have removed (// commented out the debugging)

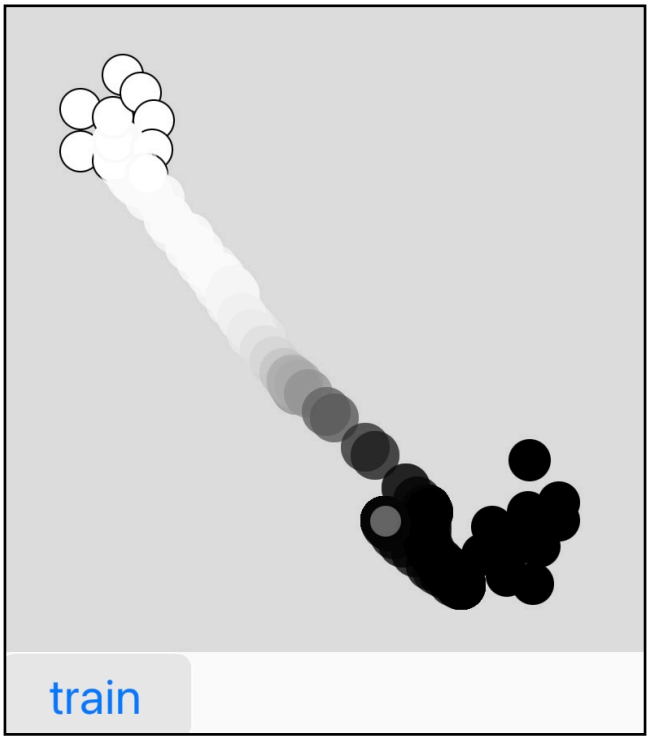
sketch.js

```
let nn
let targetLabel = 'w'
let state = 'collection'
let button
let col = {
  w: 255,
  b: 0
}

function setup()
{
  createCanvas(400, 400)
  let options = {
    inputs: ['x', 'y'],
    outputs: ['val'],
    task: 'regression',
    // debug: 'true'
  }
  nn = ml5.neuralNetwork(options)
  background(220)
  button = createButton('train')
  button.style('font-size', '30px')
  button.mousePressed(train)
}
```

```
function keyPressed()
{
  targetLabel = key
}

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  if (state == 'collection')
  {
    let targetVal = col[targetLabel]
    let target = {
      val: targetVal
    }
    nn.addData(inputs, target)
    fill(targetVal)
    circle(mouseX, mouseY, 25)
  }
  else if (state == 'prediction')
  {
    nn.predict(inputs, gotResults)
  }
}

function train()
{
  state = 'training'
  nn.normalizeData()
  let options = {
    epochs: 10
```

```
  }
  nn.train(options, whileTraining, finishedTraining)
}

function whileTraining(epochs, loss)
{
 // console.log(epochs)
}

function finishedTraining()
{
  console.log('finished training')
  state = 'prediction'
  // background(220)
}

function gotResults(results)
{
  mousePressed()
  {
  fill(floor(results[0].value), 100)
  noStroke()
  circle(mouseX, mouseY, 30)
  }
}
```

click on the canvas after training and
move it around the canvas



eventually you will get something like this