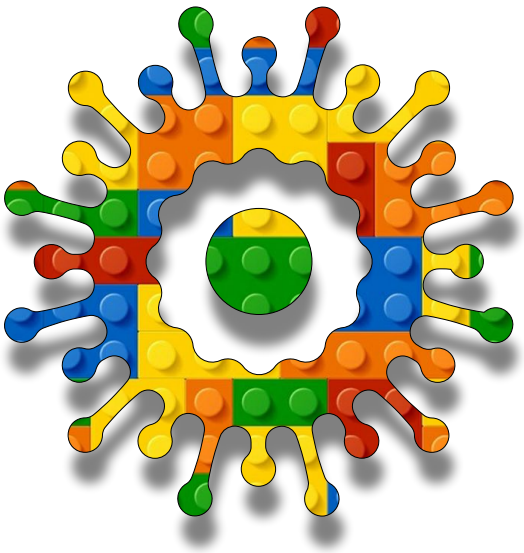
Algorithmic Intelligence Module A Unit #6 sine wave regression



Content

Module A Unit #6 sine wave regression

Introduction to sine wave regression with ml5.js

In the previous unit, we trained a model to predict a straight line and had a peek at the batch size hyperparameter. This time, we are going to make things a bit more challenging. We are going to upgrade from a straight line to a sine wave.

This will give us a chance to look at some more hyperparameters and their impact on the training of the model. We will use them to try to make better predictions of a sine wave.

The hyperparameters we will be looking at will be:

- Number of hidden layers
- Number of nodes in each layer
- Activation functions
- Epochs
- Learning Rate

The index.html

Make sure that you have the `ml5.js` line of code in your `index.html` file.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.11.0/p5.js"></
script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.11.0/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 A6.1 sine wave sketch

We are starting with the sketch from linear regression. Then we'll modify the data because instead of a line, we want a sine wave. In p5.js, the default angle units are radians, but we are going to work in degrees, which is a little more intuitive. To make this change, we use the function `angleMode()` in `setup()`. Everything else in the sketch remains the same.

```
let nn
let counter = 0
let data
const number = 30
const spread = 30
const options = {
  task: 'regression',
  debug: true
}

function setup()
{
  createCanvas(400, 400)
  angleMode(DEGREES)
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
  trainingData()
  nn.normalizeData()
  const trainingOptions = {
    batchSize: 512
  }
  nn.train(trainingOptions, finishedTraining)
}

function trainingData()
{
  background(220)
  for (let i = 0; i < width; i += 10)
  {
    for (let j = 0; j < number; j++)
    {
```

```
      data = [i + floor(random(-spread, spread)), 200 + (50 * sin(i) +
floor(random(-spread, spread)))]
      fill(0, 0, 255)
      noStroke()
      circle(data[0], data[1], 5)
      nn.addData([data[0]], [data[1]])
    }
  }
}

function finishedTraining()
{
  if (counter < 400)
  {
    nn.predict([counter], gotResults)
  }
}

function gotResults(results)
{
  let prediction = results[0]
  let x = counter
  let y = prediction.value
  stroke(255, 0, 0)
  strokeWeight(5)
  point(x, y)
  counter++
  finishedTraining()
}
```

🗒️ Notes

For every `x` input, we get the sine of that input as our `y` output. We multiply by `50` to increase the amplitude (otherwise a very flat sine wave) and move it by `200` pixels so we have it in the centre of the canvas, plus the usual variance, `random(spread)`. The results aren't as bad as you might expect, as it uses default settings for most of the hyperparameters except for the batch size.

🛠️ Code Explanation

`angleMode(DEGREES)`	This changes the default to degrees (notice all capitals)
`sin(i)`	Returns the sine of i

Figure A6.1a loss chart

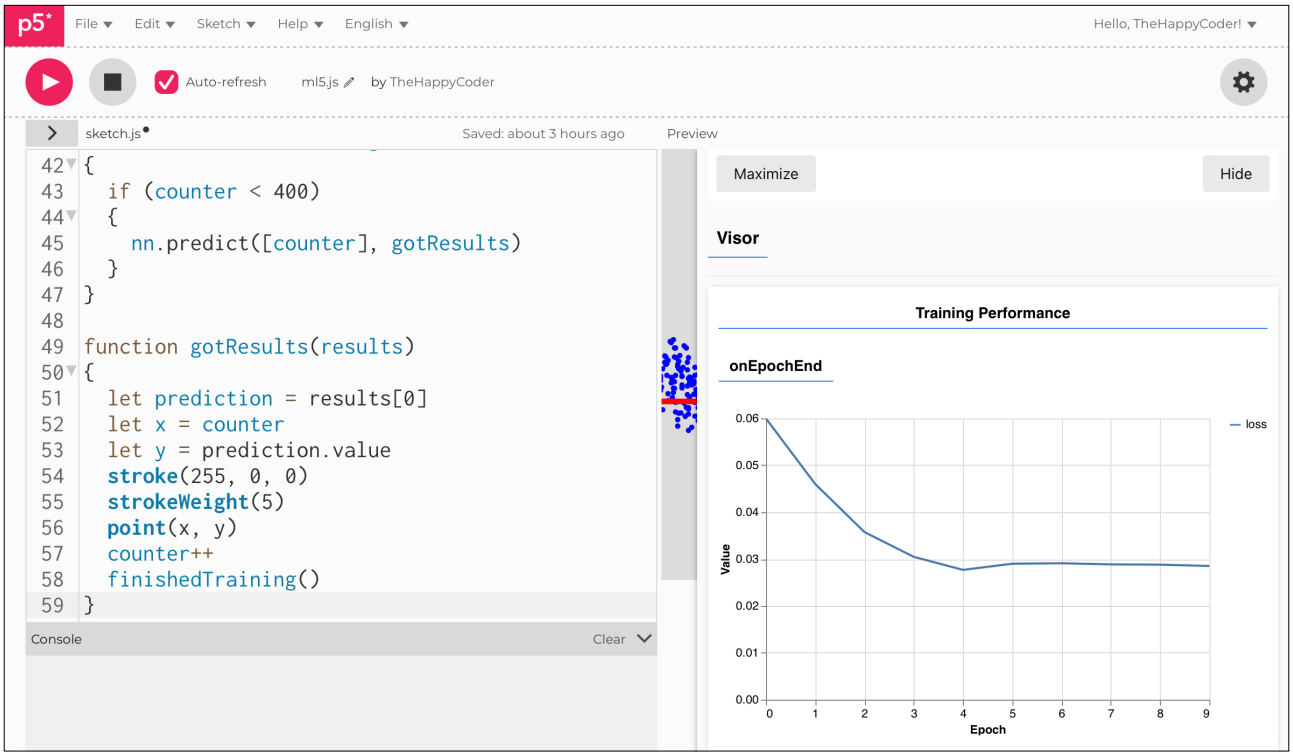
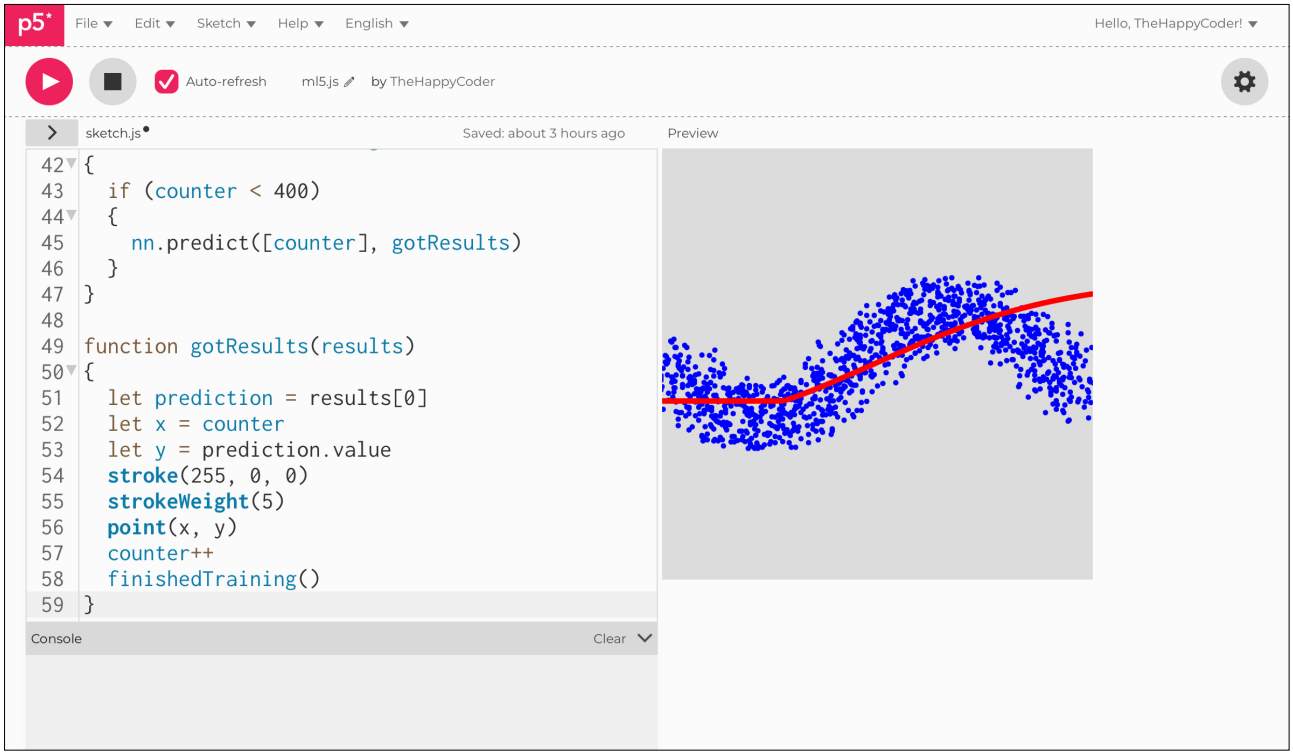


Figure A6.1b predicted results



Sketch A6.2 epochs

Our first hyperparameter change will be the number of epochs. Each epoch is one complete dataset. Currently, we are only running 10 epochs, and it looks as if the loss might still be going down after that. Let's extend it to 100 epochs and see where it levels off (or stops learning). We can introduce this hyperparameter into the training options just like we did with the batch sizes.

! You will need to put a comma (,) after the 512.

```
let nn
let counter = 0
let data
const number = 30
const spread = 30
const options = {
  task: 'regression',
  debug: true
}

function setup()
{
  createCanvas(400, 400)
  angleMode(DEGREES)
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
  trainingData()
  nn.normalizeData()
  const trainingOptions = {
    batchSize: 512,
    epochs:100
  }
  nn.train(trainingOptions, finishedTraining)
}

function trainingData()
{
  background(220)
  for (let i = 0; i < width; i += 10)
```

```
  {
    for (let j = 0; j < number; j++)
    {
      data = [i + floor(random(-spread, spread)), 200 + (50 * sin(i) +
floor(random(-spread, spread)))]
      fill(0, 0, 255)
      noStroke()
      circle(data[0], data[1], 5)
      nn.addData([data[0]], [data[1]])
    }
  }
}


function finishedTraining()
{
  if (counter < 400)
  {
    nn.predict([counter], gotResults)
  }
}


function gotResults(results)
{
  let prediction = results[0]
  let x = counter
  let y = prediction.value
  stroke(255, 0, 0)
  strokeWeight(5)
  point(x, y)
  counter++
  finishedTraining()
}
```

🗒️ Notes

The changes were not a great improvement. You may get slightly different results depending on how the weights are initialised in the model, but it seems to level out after around 10 epochs.

🌻 Challenge

Try again with different epoch settings.

🛠️ Code Explanation

`epochs:100`	Defines how many epochs if not using the default value

Figure A6.2a

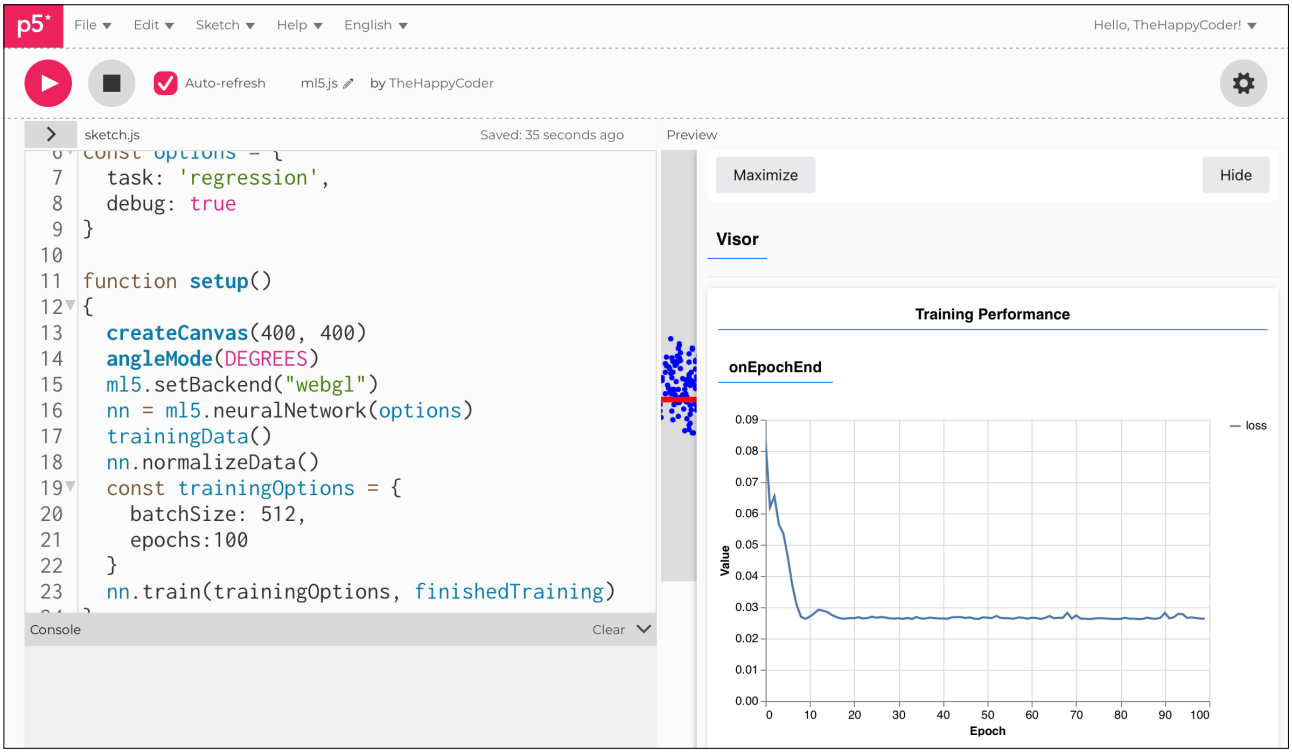
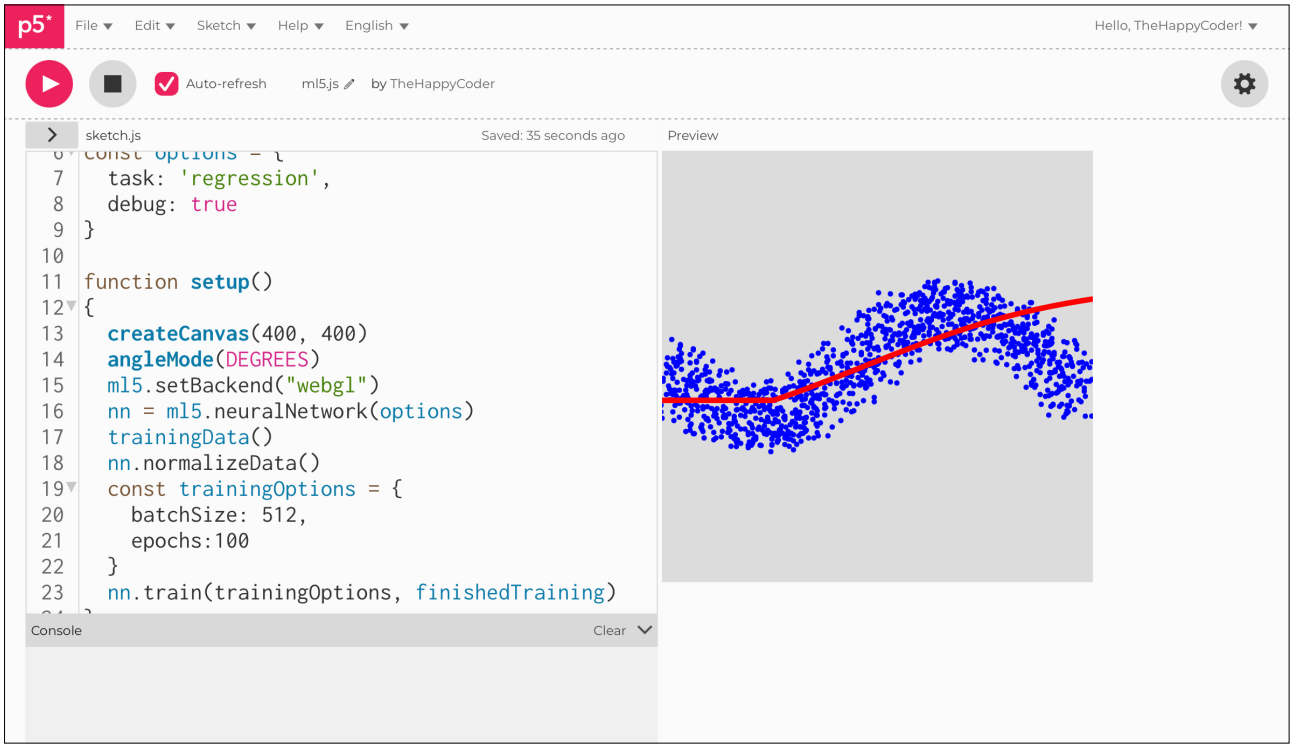


Figure A6.2b


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 35 seconds ago
Preview
const options = {
task: 'regression',
debug: true
}
function setup()
{
createCanvas(400, 400)
angleMode(DEGREES)
ml5.setBackend("webgl")
nn = ml5.neuralNetwork(options)
trainingData()
nn.normalizeData()
const trainingOptions = {
batchSize: 512,
epochs:100
}
nn.train(trainingOptions, finishedTraining)
Console
Clear

Sketch A6.3 layers

We can add more layers; more accurately, we can add more hidden layers. In those layers, we can specify the number of nodes and the activation function we want to use. The default settings are shown below:

中 Input Layer

One input node

中 Output Layer

One output node

中 Hidden Layer

By default, we have one hidden layer with 16 nodes.

中 Dense

Means all the nodes in one layer are fully connected to the node in the previous layer.

中 Activation Functions

In the hidden layer, they are the ReLU function, and for the output layer, it is the sigmoid function.

```
let nn
let counter = 0
let data
const number = 30
const spread = 30
const options = {
  task: 'regression',
  layers: [
    {
      type: 'dense',
      units: 16,
      activation: 'relu'
    },
    {
      type: 'dense',
      activation: 'sigmoid'
    }
  ],
  debug: true
```

```
}

function setup()
{
  createCanvas(400, 400)
  angleMode(DEGREES)
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
  trainingData()
  nn.normalizeData()
  const trainingOptions = {
    batchSize: 512,
    epochs:100
  }
  nn.train(trainingOptions, finishedTraining)
}

function trainingData()
{
  background(220)
  for (let i = 0; i < width; i += 10)
  {
    for (let j = 0; j < number; j++)
    {
      data = [i + floor(random(-spread, spread)), 200 + (50 * sin(i) +
floor(random(-spread, spread)))]
      fill(0, 0, 255)
      noStroke()
      circle(data[0], data[1], 5)
      nn.addData([data[0]], [data[1]])
    }
  }
}

function finishedTraining()
{
  if (counter < 400)
  {
    nn.predict([counter], gotResults)
```

```
  }
}

function gotResults(results)
{
  let prediction = results[0]
  let x = counter
  let y = prediction.value
  stroke(255, 0, 0)
  strokeWeight(5)
  point(x, y)
  counter++
  finishedTraining()
}
```

🗒️ Notes

As you can see, it has made no difference, which is really as expected. If you get a better (or worse) result, just remember that it is starting each time with different (random) weights as well as random data.

🌻 Challenge

You might want to start changing the number of nodes in the hidden layer.

🛠️ Code Explanation

`layers: [...]`	Adding layers in the training option
`type: 'dense'`	Fully connected nodes between each layer
`units: 16`	16 nodes (neurons) in the hidden layer
`activation: 'relu'`	Hidden layer activation function
`activation: 'sigmoid'`	Output layer activation function

Figure A6.3a

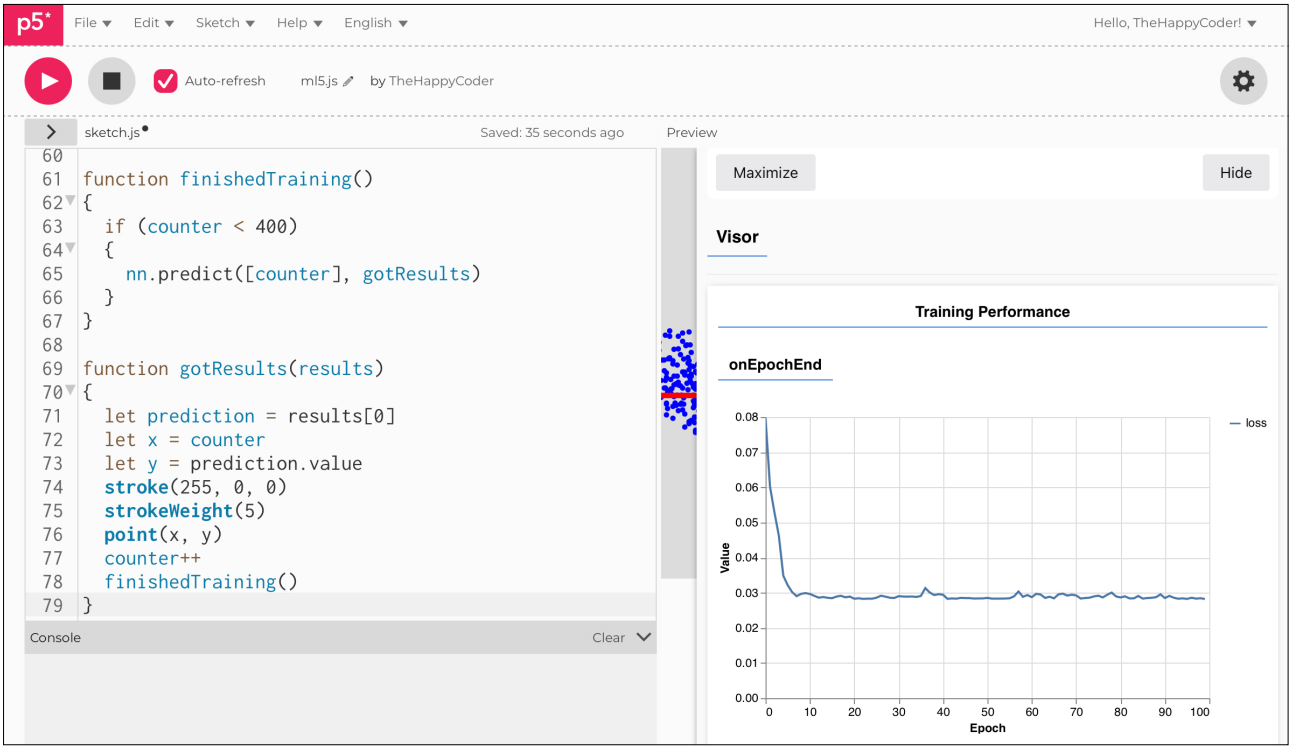
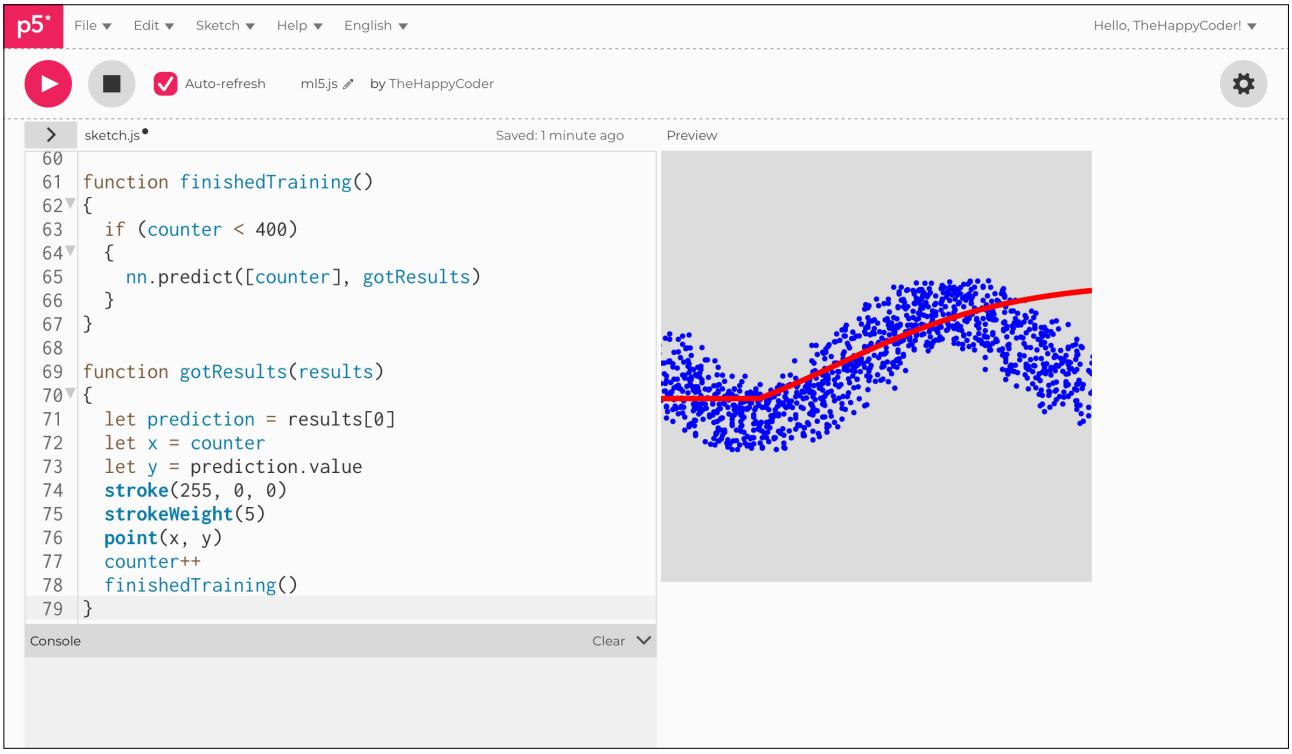

p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 35 seconds ago
Preview
60
61 function finishedTraining()
62 {
63 if (counter < 400)
64 {
65 nn.predict([counter], gotResults)
66 }
67 }
68
69 function gotResults(results)
70 {
71 let prediction = results[0]
72 let x = counter
73 let y = prediction.value
74 stroke(255, 0, 0)
75 strokeWeight(5)
76 point(x, y)
77 counter++
78 finishedTraining()
79 }
Console
Clear
Maximize
Hide
Visor
Training Performance
onEpochEnd
loss
Value
Epoch
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
10
20
30
40
50
60
70
80
90
100

Figure A6.3b


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 1 minute ago
Preview
60
61 function finishedTraining()
62 {
63 if (counter < 400)
64 {
65 nn.predict([counter], gotResults)
66 }
67 }
68
69 function gotResults(results)
70 {
71 let prediction = results[0]
72 let x = counter
73 let y = prediction.value
74 stroke(255, 0, 0)
75 strokeWeight(5)
76 point(x, y)
77 counter++
78 finishedTraining()
79 }
Console
Clear

Sketch A6.4 more hidden layers

Added two more identical hidden layers.

```
let nn
let counter = 0
let data
const number = 30
const spread = 30
const options = {
  task: 'regression',
  layers: [
    {
      type: 'dense',
      units: 16,
      activation: 'relu'
    },
    {
      type: 'dense',
      units: 16,
      activation: 'relu'
    },
    {
      type: 'dense',
      units: 16,
      activation: 'relu'
    },
    {
      type: 'dense',
      activation: 'sigmoid'
    }
  ],
  debug: true
}

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

```
  angleMode(DEGREES)
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
  trainingData()
  nn.normalizeData()
  const trainingOptions = {
    batchSize: 512,
    epochs:100
  }
  nn.train(trainingOptions, finishedTraining)
}

function trainingData()
{
  background(220)
  for (let i = 0; i < width; i += 10)
  {
    for (let j = 0; j < number; j++)
    {
      data = [i + floor(random(-spread, spread)), 200 + (50 * sin(i) +
floor(random(-spread, spread)))]
      fill(0, 0, 255)
      noStroke()
      circle(data[0], data[1], 5)
      nn.addData([data[0]], [data[1]])
    }
  }
}

function finishedTraining()
{
  if (counter < 400)
  {
    nn.predict([counter], gotResults)
  }
}

function gotResults(results)
{
```

```
  let prediction = results[0]
  let x = counter
  let y = prediction.value
  stroke(255, 0, 0)
  strokeWeight(5)
  point(x, y)
  counter++
  finishedTraining()
}
```

🗒️ Notes

You can see in my example the beginnings of a sine wave, but far from what we might want.

Figure A6.4a

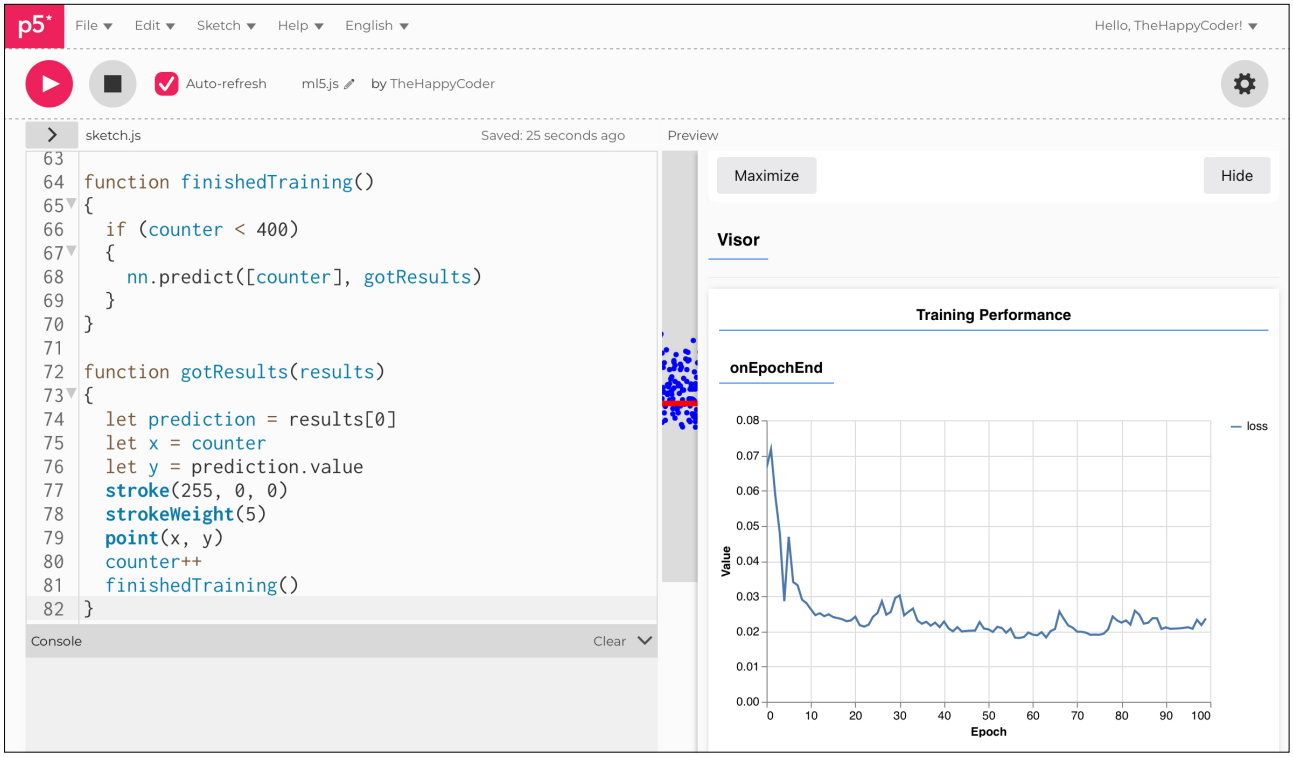
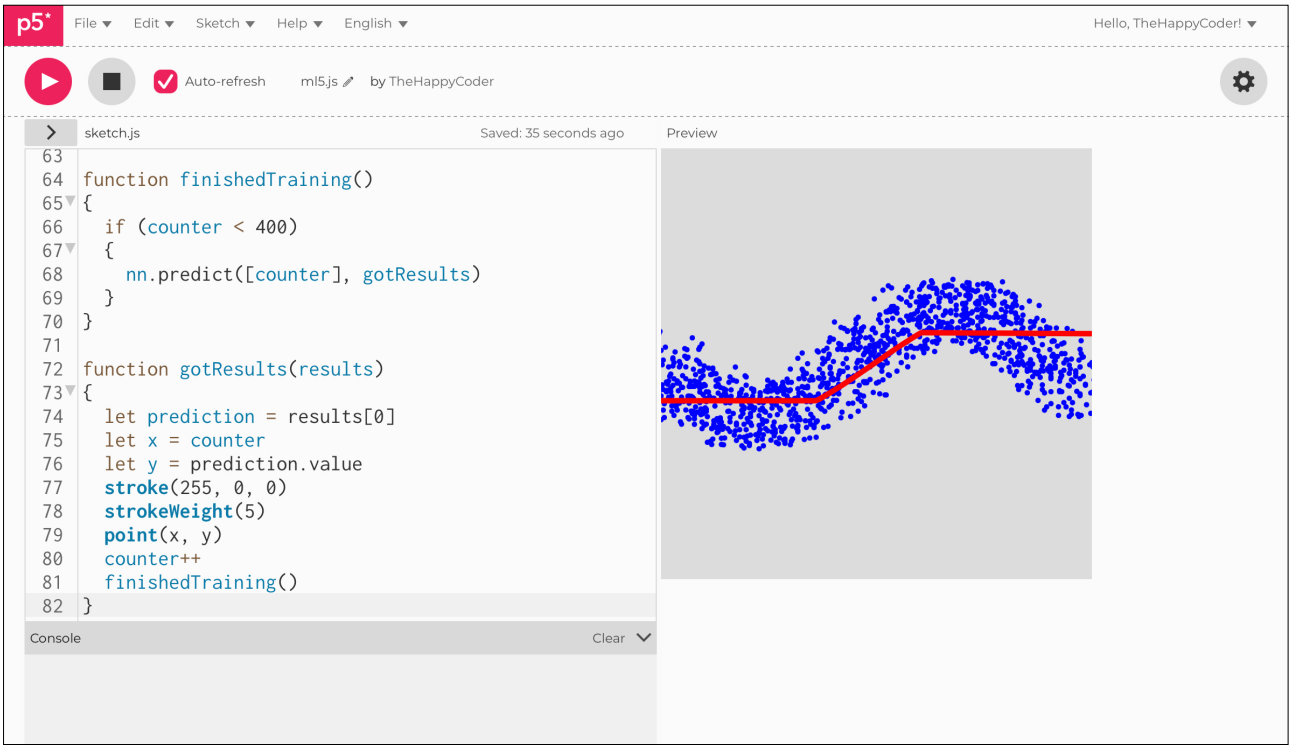


Figure A6.4b


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 35 seconds ago
Preview
63
64 function finishedTraining()
65 {
66 if (counter < 400)
67 {
68 nn.predict([counter], gotResults)
69 }
70 }
71
72 function gotResults(results)
73 {
74 let prediction = results[0]
75 let x = counter
76 let y = prediction.value
77 stroke(255, 0, 0)
78 strokeWeight(5)
79 point(x, y)
80 counter++
81 finishedTraining()
82 }
Console
Clear

Sketch A6.5 more nodes

Increasing the number of nodes, they don't have to be equal for each hidden layer; it is worth trying mixing it up a bit to see what happens.

```
let nn
let counter = 0
let data
const number = 30
const spread = 30
const options = {
  task: 'regression',
  layers: [
    {
      type: 'dense',
      units: 64,
      activation: 'relu'
    },
    {
      type: 'dense',
      units: 64,
      activation: 'relu'
    },
    {
      type: 'dense',
      units: 64,
      activation: 'relu'
    },
    {
      type: 'dense',
      activation: 'sigmoid'
    }
  ],
  debug: true
}

function setup()
{
```

```
  createCanvas(400, 400)
  angleMode(DEGREES)
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
  trainingData()
  nn.normalizeData()
  const trainingOptions = {
    batchSize: 512,
    epochs:100
  }
  nn.train(trainingOptions, finishedTraining)
}

function trainingData()
{
  background(220)
  for (let i = 0; i < width; i += 10)
  {
    for (let j = 0; j < number; j++)
    {
      data = [i + floor(random(-spread, spread)), 200 + (50 * sin(i) +
floor(random(-spread, spread)))]
      fill(0, 0, 255)
      noStroke()
      circle(data[0], data[1], 5)
      nn.addData([data[0]], [data[1]])
    }
  }
}

function finishedTraining()
{
  if (counter < 400)
  {
    nn.predict([counter], gotResults)
  }
}

function gotResults(results)
```

```
{
  let prediction = results[0]
  let x = counter
  let y = prediction.value
  stroke(255, 0, 0)
  strokeWeight(5)
  point(x, y)
  counter++
  finishedTraining()
}
```

🗒️ Notes

You can see the pattern emerging, but it is still less than satisfactory.

🌻 Challenge

Think about more/fewer nodes, more/fewer layers.

Figure A6.5a

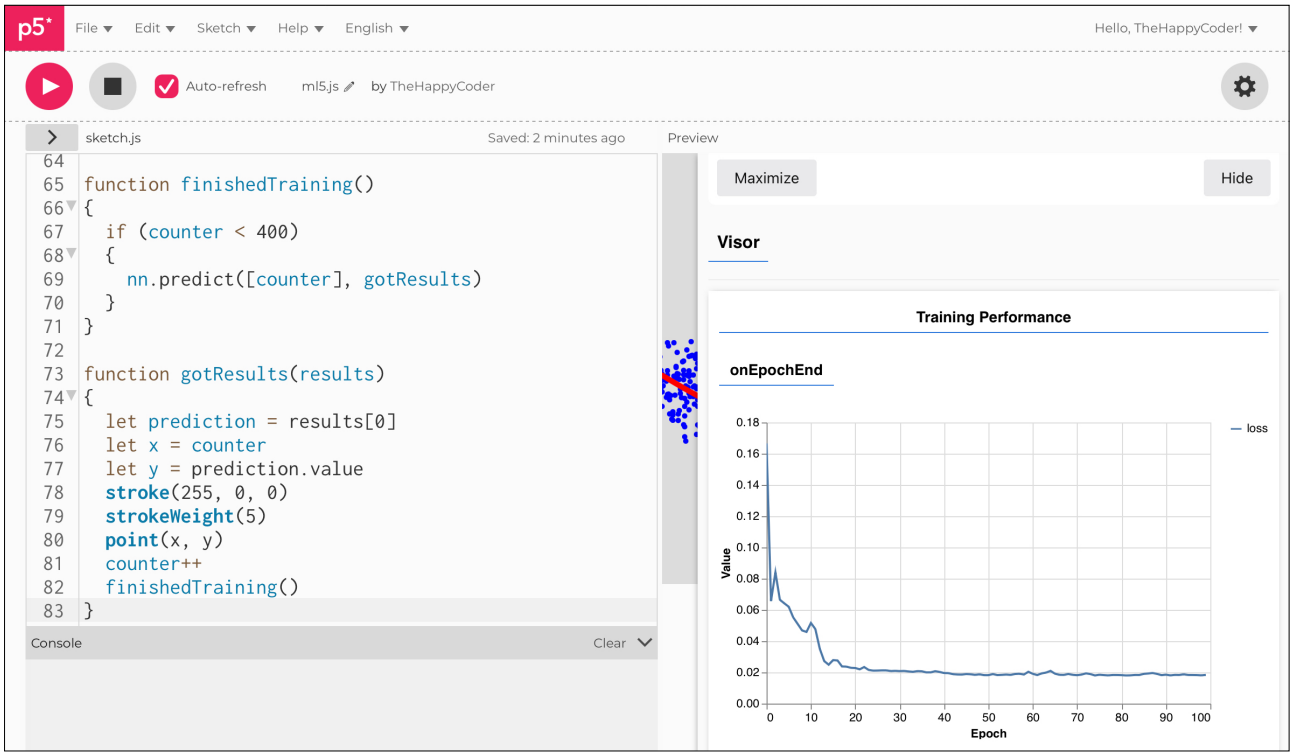
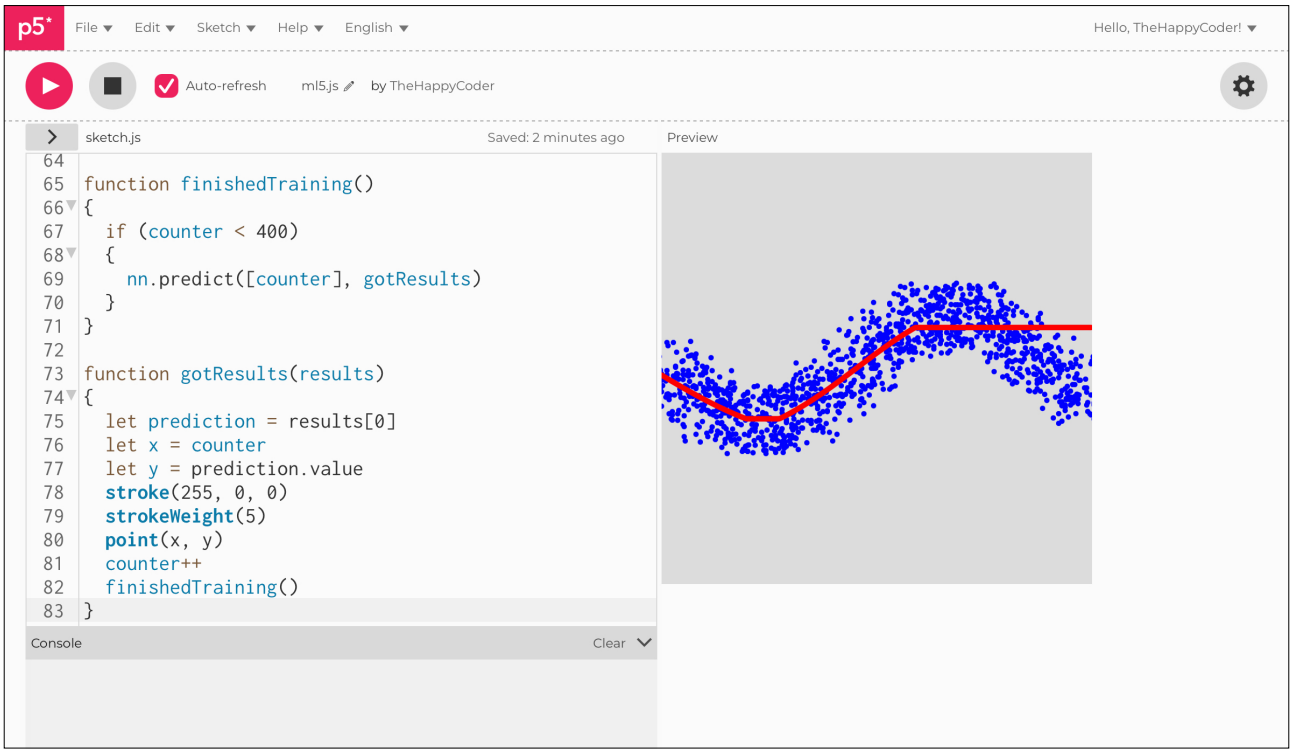


Figure A6.5b


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 2 minutes ago
Preview
64
65 function finishedTraining()
66 {
67 if (counter < 400)
68 {
69 nn.predict([counter], gotResults)
70 }
71 }
72
73 function gotResults(results)
74 {
75 let prediction = results[0]
76 let x = counter
77 let y = prediction.value
78 stroke(255, 0, 0)
79 strokeWeight(5)
80 point(x, y)
81 counter++
82 finishedTraining()
83 }
Console
Clear

Sketch A6.6 learning rate

Another hyperparameter we can change is the learning rate. The learningRate is the size of steps the algorithm takes to find the lowest point in the calculations. If the steps are too big, they can miss the global minimum; too small, and it can never find the minimum and gets stuck on a local minimum. We are keeping the number of layers and nodes the same as in the previous sketch.

```
let nn
let counter = 0
let data
const number = 30
const spread = 30
const options = {
  task: 'regression',
  learningRate: 0.1,
  layers: [
    {
      type: 'dense',
      units: 64,
      activation: 'relu'
    },
    {
      type: 'dense',
      units: 64,
      activation: 'relu'
    },
    {
      type: 'dense',
      units: 64,
      activation: 'relu'
    },
    {
      type: 'dense',
      activation: 'sigmoid'
    }
  ],
  debug: true
}
```

```
function setup()
{
  createCanvas(400, 400)
  angleMode(DEGREES)
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
  trainingData()
  nn.normalizeData()
  const trainingOptions = {
    batchSize: 512,
    epochs:100
  }
  nn.train(trainingOptions, finishedTraining)
}

function trainingData()
{
  background(220)
  for (let i = 0; i < width; i += 10)
  {
    for (let j = 0; j < number; j++)
    {
      data = [i + floor(random(-spread, spread)), 200 + (50 * sin(i) +
floor(random(-spread, spread)))]
      fill(0, 0, 255)
      noStroke()
      circle(data[0], data[1], 5)
      nn.addData([data[0]], [data[1]])
    }
  }
}

function finishedTraining()
{
  if (counter < 400)
  {
    nn.predict([counter], gotResults)
  }
```

```
}

function gotResults(results)
{
  let prediction = results[0]
  let x = counter
  let y = prediction.value
  stroke(255, 0, 0)
  strokeWeight(5)
  point(x, y)
  counter++
  finishedTraining()
}
```

🗒️ Notes

Not a significant difference, but the loss function is very jumpy.

🌻 Challenge

I could fiddle with these values endlessly, and that is why machine learning is an art as much as a science. I will let you play around, perhaps trying learning rates of `1` or `0.001`, more nodes but fewer hidden layers.

🛠️ Code Explanation

`learningRate: 0.1`	We can specify the learning rate, the steps (or jumps) in the training process

Figure A6.6a

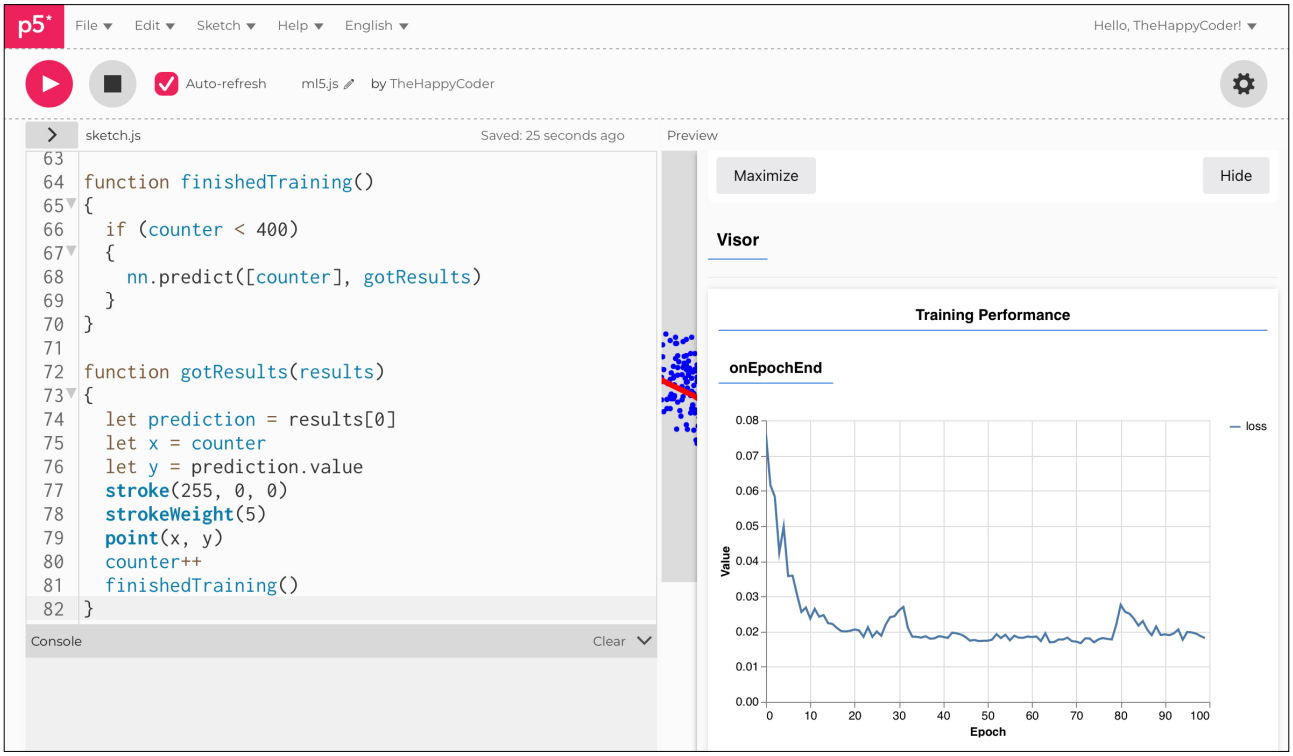
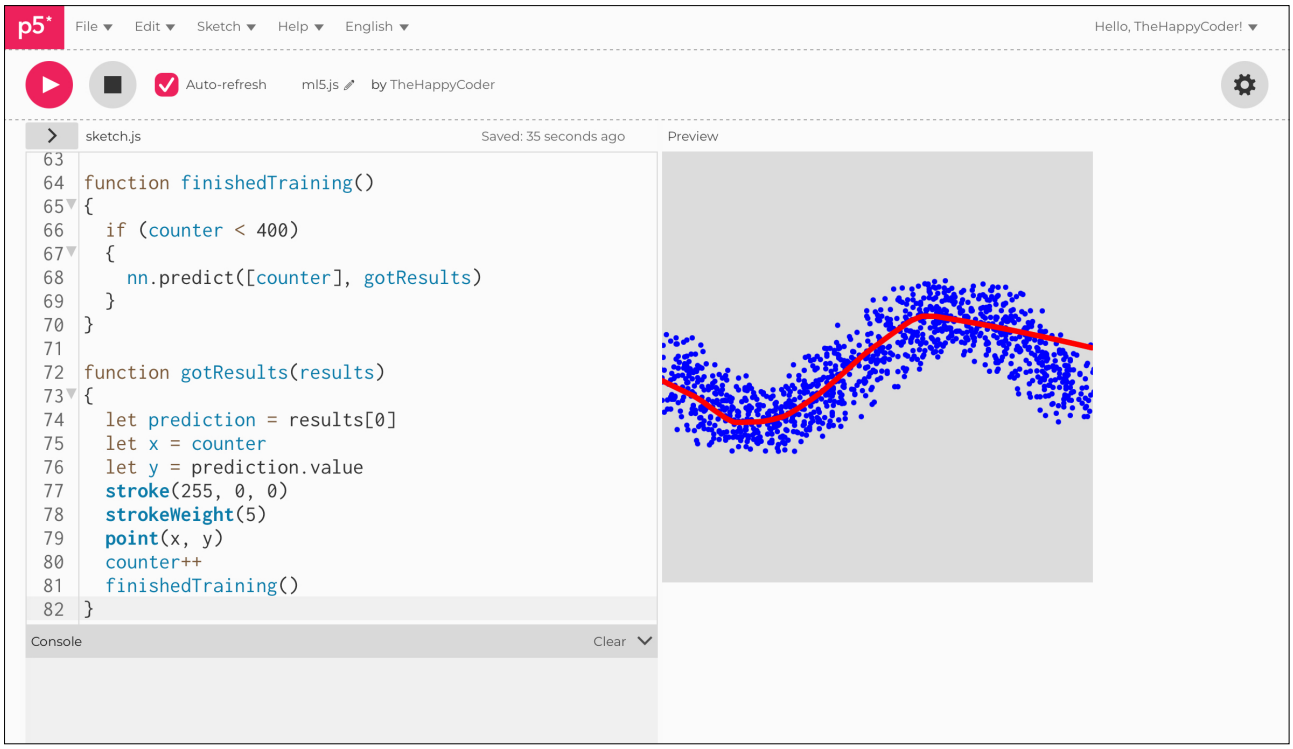


Figure A6.6b



Sketch A6.7 always a bit of trial and error

I have increased the number of nodes but reduced the number of layers, as well as reduced the learning rate to `0.01`.

```
let nn
let counter = 0
let data
const number = 30
const spread = 30
const options = {
  task: 'regression',
  learningRate: 0.01,
  layers: [
    {
      type: 'dense',
      units: 256,
      activation: 'relu'
    },
    {
      type: 'dense',
      units: 256,
      activation: 'relu'
    },
    {
      type: 'dense',
      activation: 'sigmoid'
    }
  ],
  debug: true
}

function setup()
{
  createCanvas(400, 400)
  angleMode(DEGREES)
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
```

```
  trainingData()
  nn.normalizeData()
  const trainingOptions = {
    batchSize: 512,
    epochs:100
  }
  nn.train(trainingOptions, finishedTraining)
}

function trainingData()
{
  background(220)
  for (let i = 0; i < width; i += 10)
  {
    for (let j = 0; j < number; j++)
    {
      data = [i + floor(random(-spread, spread)), 200 + (50 * sin(i) +
floor(random(-spread, spread)))]
      fill(0, 0, 255)
      noStroke()
      circle(data[0], data[1], 5)
      nn.addData([data[0]], [data[1]])
    }
  }
}

function finishedTraining()
{
  if (counter < 400)
  {
    nn.predict([counter], gotResults)
  }
}

function gotResults(results)
{
  let prediction = results[0]
  let x = counter
  let y = prediction.value
```

```
  stroke(255, 0, 0)
  strokeWeight(5)
  point(x, y)
  counter++
  finishedTraining()
}
```

🗒️ Notes

Starting to see some improvements. Bear in mind, however, that every time you run the code, we are using different data (not saved or uploaded), but this is for ease of use and demonstration purposes.

🌻 Challenge

Play around with the number of `nodes` and the `learning rate`, consider saving the data in the first place, and then experimenting with the hyperparameters.

Figure A6.7a

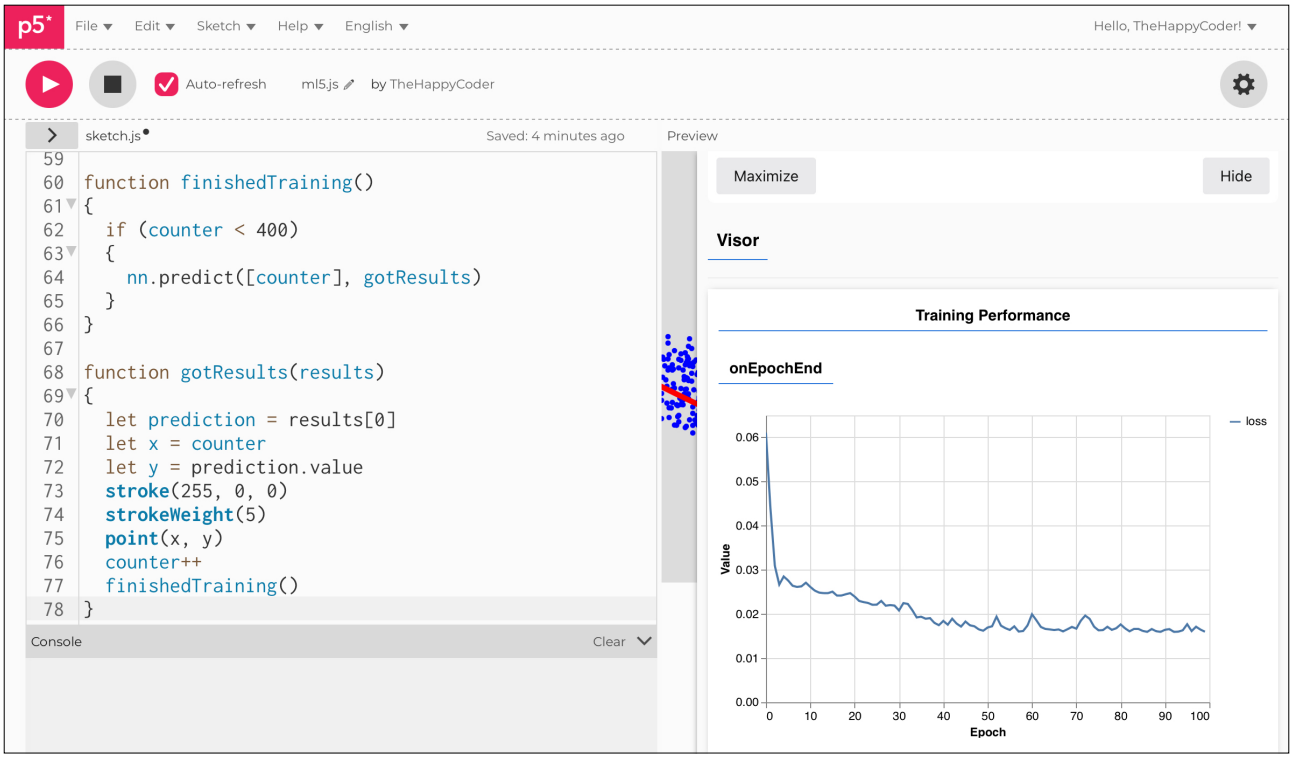
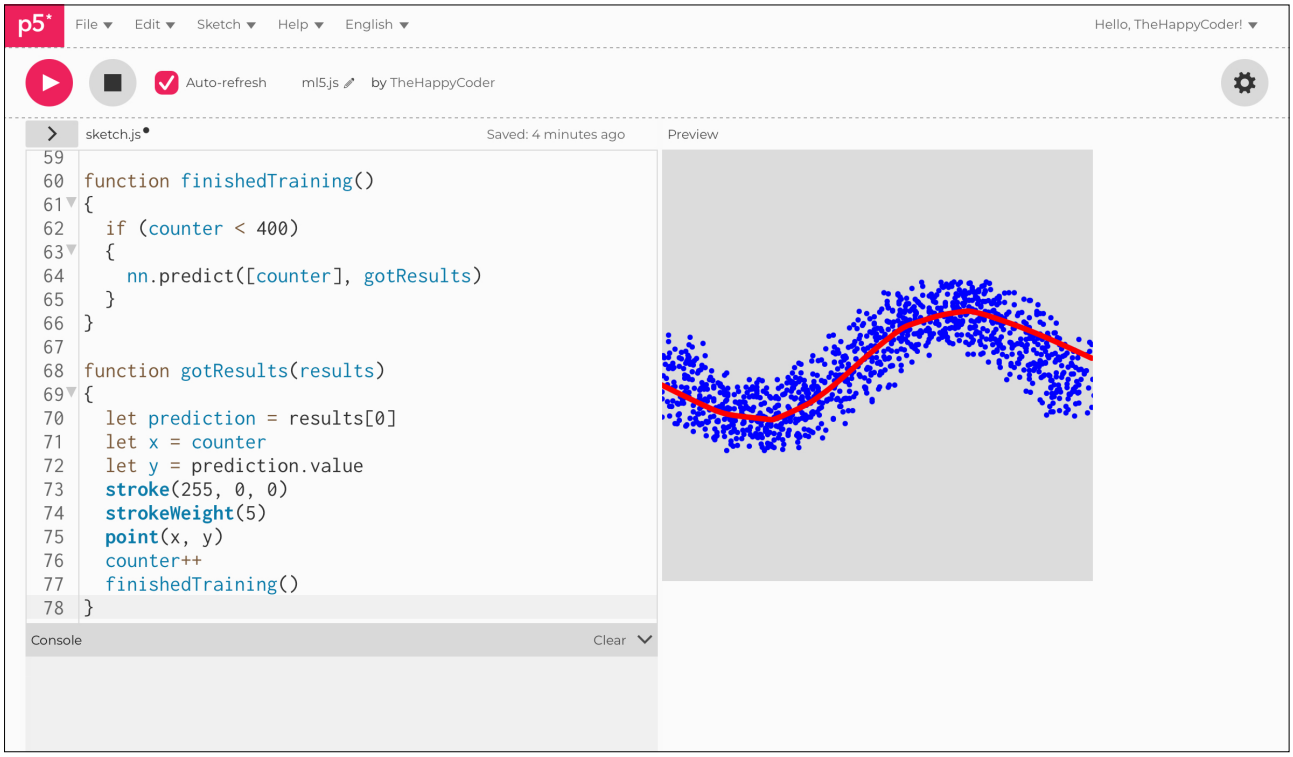


Figure A6.7b



Activation functions

The final hyperparameter we are going to look at is the activation functions. The default for a regression task is hidden layer: `ReLU` and output layer: `Sigmoid`.

Just to save on space, I will not change any other lines of code except the options for layers, so the rest of the code remains the same and is omitted for brevity.

Sketch A6.8 changing the activation function

Using the same hyperparameters, we will switch the ReLU to sigmoid.

```
let nn
let counter = 0
let data
const number = 30
const spread = 30
const options = {
  task: 'regression',
  learningRate: 0.01,
  layers: [
    {
      type: 'dense',
      units: 256,
      activation: 'sigmoid'
    },
    {
      type: 'dense',
      units: 256,
      activation: 'sigmoid'
    },
    {
      type: 'dense',
      activation: 'sigmoid'
    }
  ],
  debug: true
}

function setup()
{
  createCanvas(400, 400)
  angleMode(DEGREES)
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
  trainingData()
```

```
  nn.normalizeData()
  const trainingOptions = {
    batchSize: 512,
    epochs:100
  }
  nn.train(trainingOptions, finishedTraining)
  console.log()
}

function trainingData()
{
  background(220)
  for (let i = 0; i < width; i += 10)
  {
    for (let j = 0; j < number; j++)
    {
      data = [i + floor(random(-spread, spread)), 200 + (50 * sin(i) +
floor(random(-spread, spread)))]
      fill(0, 0, 255)
      noStroke()
      circle(data[0], data[1], 5)
      nn.addData([data[0]], [data[1]])
    }
  }
}

function finishedTraining()
{
  if (counter < 400)
  {
    nn.predict([counter], gotResults)
  }
}

function gotResults(results)
{
  let prediction = results[0]
  let x = counter
  let y = prediction.value
```

```
  stroke(255, 0, 0)
  strokeWeight(5)
  point(x, y)
  counter++
  finishedTraining()
}
```

🗒️ Notes

Oh, dear! ReLU gives consistently better results. It is more efficient and is the industry standard. You could see if you can get better results with fewer or more layers and nodes. I will leave that with you to experiment.

🌻 Challenge

Use `tanh` instead of `sigmoid`.

Figure A6.8a

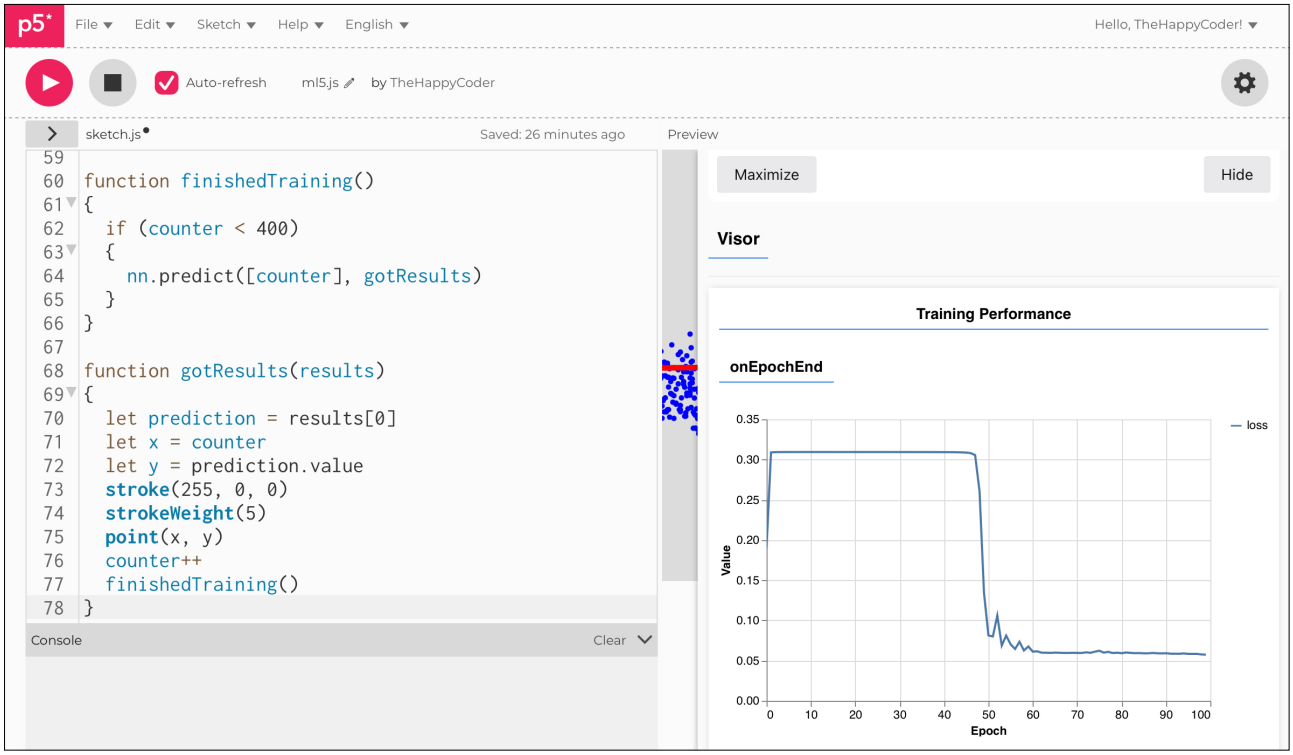
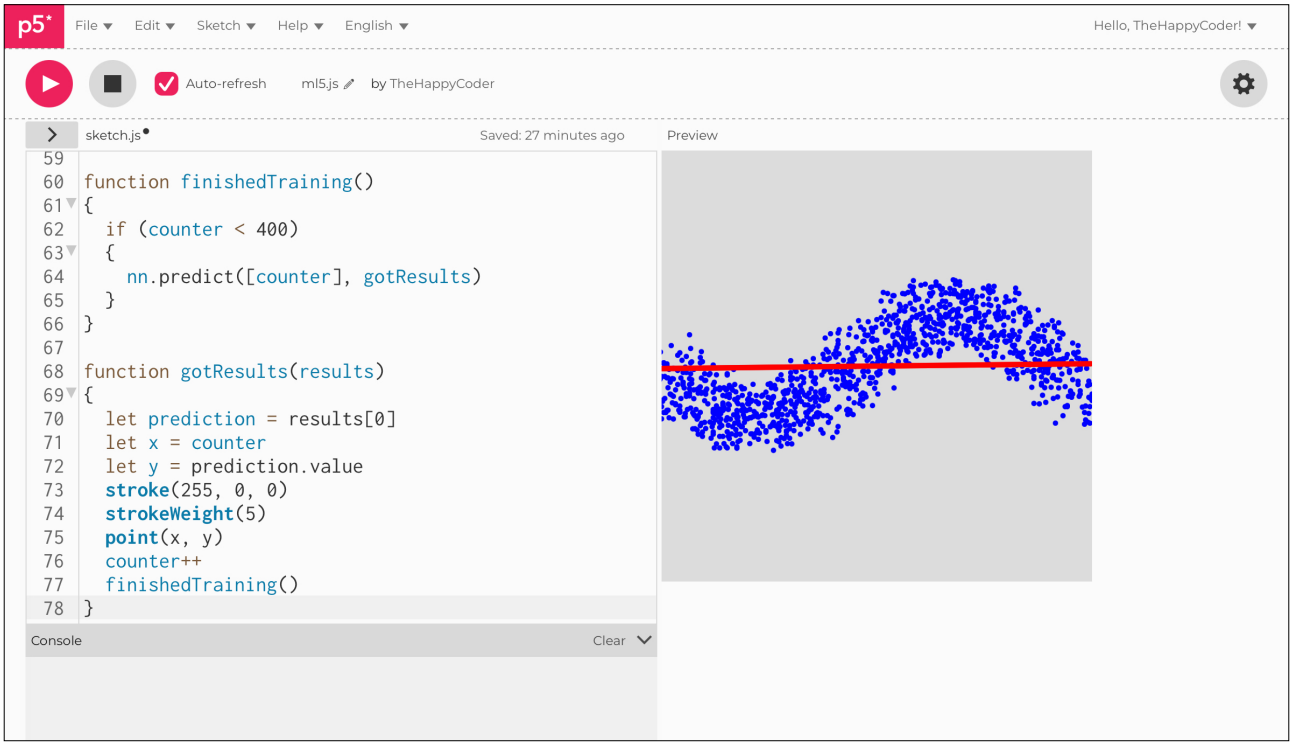


Figure A6.8b


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 27 minutes ago
Preview
59
60 function finishedTraining()
61 {
62 if (counter < 400)
63 {
64 nn.predict([counter], gotResults)
65 }
66 }
67
68 function gotResults(results)
69 {
70 let prediction = results[0]
71 let x = counter
72 let y = prediction.value
73 stroke(255, 0, 0)
74 strokeWeight(5)
75 point(x, y)
76 counter++
77 finishedTraining()
78 }
Console
Clear

Final challenges

We have looked at the following hyperparameters:

- Batch size
- Epochs
- Hidden layers
- Nodes
- Activation functions
- Learning rate

I would strongly recommend playing with all of these and seeing how they impact the model's learning. Remember that you will never get a perfect result. Also, try drawing the sine wave as a single line rather than a scatter graph approach as we did with the linear regression.

Summary

You will probably start to realise that throwing more `nodes` and `hidden layers` at the dataset doesn't necessarily improve the outcome. You may also realise that having a smaller `learning rate` and a smaller `batch size` may give you some slightly improved outcome, but you may have to wait a considerably longer time. Remember, in the real world, time and energy are money.