A Neural Network for the JavaScript programmer

Ian Channing

November 25, 2018

The beginning

Let's generate some random data, visualize it and train a neuron to classify it

> *Young man, in mathematics you don't understand things. You just get used to them. — John Von Neumann*

Inspired / blatantly copied from

- Funfunfunction NN playlist (but it's missing maths)
- deeplearning.ai week 2 (code isn't open, filling in blanks)
- NN & DL course (no code)

So I want to try and give some respect to the code & the maths

I'm going to sneak some functional programming in

Slight digression (it'll be worth it in the long run)

JavaScript's `map` and `reduce` functions in maths

Reduce the gap between maths and code

$y = f(x) = 2x$

Still with me?

Let's draw a graph

```
  y
  ^
  |
4 |   +
3 |
2 | +
1 |
0 +------> x
  0 1 2
```

Mathsy definitions

This is actually University level maths - Set Theory.

But I'll try anyway.

What's the mathsy name for:

I've got one 'set' and I want to go to another 'set'?

```
 xs                   ys
+-------+            +-------+
| 0 1 2 | -- f -->   | 0 2 4 |
+-------+            +-------+
```

Mathsy definitions

Mapping!

Still with me?

$f(x)$ in JavaScript

$y = f(x) = 2x$

```
function f(x) {return 2 * x;}
// could have called it 'double'
// function double(x) {return 2 * x;}
var xs = [0,1,2]; // want output [0,2,4]
var ys = xs.map(f); // [0,2,4]
```

$2 + 2 + 2$

$$y = \sum f(x) = \sum 2x$$

```
 x  2x Running total
 1  2  2
 1  2  4
+1 +2  6
-- --
 3  6
```

```
function sum(t, x) { return t + f(x); }
var xs = [1,1,1];
var y  = xs.reduce(sum, 0); // 6
```

I want it to display random values

Generate random test and training sets

```
function rand(min, max) {
  return Math.random() * (max - min) + min;
}
rand(1,3);
rand(0,400); // what we actually use
```

Stretch (*) and shift (+)

```
rand(0,1) -->  rand(1,3)

0     1     2     3
+-----+
+-----+-----+          (Stretch by (3 - 1))
      +-----+-----+    (Shift by 1)
```

I want to generate a set of random test values

I want to generate a set of random examples

I want to display these test values

I want to draw a circle

I want to draw the test values as circles on a graph

I want to separate these circles with a line

I want to colour the circles red or blue

I want to make the colour depend on which side of the line

I want to say whether my examples are red or blue

I want to make a guess based on x, y whether a circle is red or blue

I want to visualise the functions we're going to use to improve the guesses

A neural network of one neuron

An Englishman, even if he is alone, forms an orderly queue of one - George Mikes

Simplify network down to one neuron

Neurons act independently so can scale up process to a network

```
      w1  +--------------------------+
 x1 ------|               |          |
      w2  | z = w . x + b | a = g(z) |----> ~y
 x2 ------|               |          |
          +--------------------------+
```

`~y` is our approx/guess of `y`, usually called $\hat{y}$ ‘y hat’

g is our ‘activation’ function

I want to describe a neuron firing

Originally called a perceptron, but later changed to a neuron

Initial code will be for a perceptron, then we'll upgrade to a neuron

Perceptron 'fires' when inputs reach a threshold

```
             | 0 if w . x <= threshold
activation = |
             | 1 if w . x > threshold
```

Subtract threshold from both sides and call it 'bias'

```
bias = -threshold

             | 0 if w . x + bias <= 0
activation = |
             | 1 if w . x + bias > 0


 a
 ^
```

Todo . . .

I want to specify the cost function

Let's meet the the cross entropy cost function.

The bit we use is the derivative for back-propagation in eqn (61)

$dC/dW_j = 1/n * \sum_x x_j(g(z) - y)$

To be continued. . .

I want to explain why we get bias/over-fitting

Here we loop around our examples just once. But for more complex problems we loop over the same examples thousands of times.
When you say the same word a thousand times over you start to notice tiny details about the word that aren't relevant.
e.g. conscience, that's actually con-science but that's totally irrelevant. Neural Networks have no other ideas about the world except for the examples we give them.

In summary

Generated random set of training and test data that we displayed on a graph for testing

Split the points on the graph using an abritrary line (why? back propagation needs linear separation)

Used a perceptron with an activation function and back propagation algorithm

Trained this perceptron to adjust two weights that then colour the test points depending on which side of the line

This is one step of gradient descent

'Improved' this with a sigmoid activation function and it's differentiated back propagation.

The end

> *implementing it myself from scratch was the most important*
> *— Andrej Karpathy talking to Andrew Ng (2018)*

...

> *What I cannot create, I do not understand. — Richard Feynman (1988)*

...

> *What you really want is to feel every element (and the connections between them) in your bones. — Michael Nielsen (2019)*

Inspired / blatantly copied from:

- Funfunfunction NN playlist
- deeplearning.ai week 2
- NN & DL course