Machine Learning: Collectives of local decision makers

Formulating A Suitable Perceptron Paramaterisation

Daniel Braithwaite

April 18, 2017

The goal of this project is to create neurons which insted of searching for a global minumum they will choose a specific part of the data to place their hyperplane. Consider the folowing situation in $\mathbb{R}^2$ where we have data split into two classes and the second class is boxed into a chevron shape. A single perceptron will go and place a horizonal line through the plane however this isnt telling us anything interesting. Prehaps something better to do is to align its self to one of the sides of the chevron, then if we added another "boundary hunter" it could align its self to the other side of the chevron.

1 Simplified Situation

Before worrying about this in the general case we aim to get a better understanding of the problem by first trying to solve the case described above. We first make some changes to how we have paramaterised our perceptron. We want to find a way which peforms the same as a standard perceptron before we start making it into a boundary hunter

1.1 Point & Gradient Paramaterisation

We are trying to optimise a point (x,y) and a gradient. This is an alternative way to paramaterise our line. We are using sum squared error for our loss function and sigmoid as our activation. Each "boundary hunter" (BH) has the folowing weights vector $W = [m, x_0, y_0]$ and we consider two inputs to our BH, x and y. Our perceptron computes $z = (y - y_0) - m(x - x_0)$ and then outputs $o = f(z) = \frac{1}{1+e^{-z}}$.

1.1.1 Deriving Gradients

$$\frac{\partial E}{\partial W_a} = \frac{\partial E}{\partial o}\frac{\partial o}{\partial z}\frac{\partial z}{\partial W_a} \tag{1}$$

$$\frac{\partial o}{\partial z} = o(1-o) \tag{2}$$

$$\frac{\partial E}{\partial o} = -(t-o) \tag{3}$$

$$\frac{\partial z}{\partial y_0} = -1 \tag{4}$$

$$\frac{\partial z}{\partial x_0} = m \tag{5}$$

$$\frac{\partial z}{\partial m} = x_0 - x \tag{6}$$

1.1.2 Comparason To Perceptron

If we use the same set of data to train both our standard perceptron and our modified perceptron we see that our modified dosnt quite peform as well as the standard version. For both models below they where trained for 50000 iterations of the training data.

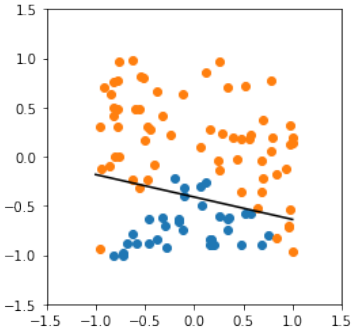

Figure 1: Standard Perceptron (SSE = 4.90)

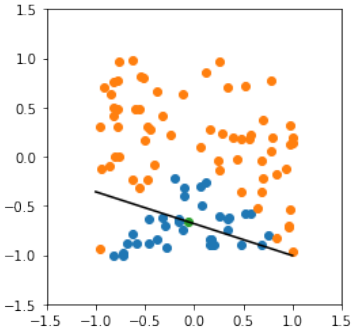

Figure 2: Modified Perceptron (SSE = 7.56)

I propose that this way of constructing our perceptron is less powerful than the standard way. Consider when we are optimising our point-slope representation,

$$\begin{aligned} y - y_0 &= m(x - x_0) \\ y &= y_0 + mx - mx_0 \\ y &= mx + (y_0 + mx_0) \end{aligned}$$

We have just shown that (as we would expect) we can convert out point-slope representation into a intercept-slope representation. From here we see we can convert into something that could be represented by our origonal perceptron.

$$y = mx + (y_0 + mx_0)$$
$$\Rightarrow -(y_0 + mx_0) - mx + y$$

So given the generic form of our standard perceptron $A + Bx + Cy$ we see here that our modified perceptron can only ever learn a representation where C = 1, limiting what we can achieve. An easy way to check this is to see what happens if we use a standard perceptron to learn the data but keep the third weight (i.e. C) fixed at 1. Sure enough we get the folowing result when we run that experement over 50000 iterations.

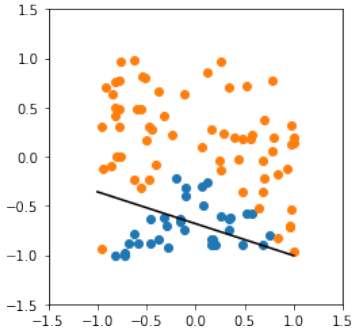

Figure 3: Standard Perceptron with C = 1 (SSE = 7.56)

So we are able to conclude that this method is unsuitable

1.2 Normal & Point Paramaterisation

This method involves learning the normal vector (which is what the standard perceptron is doing) along with a point on the hyperplane. This increaes the number of paramaters we have to learn by n (when we are in $\mathbb{R}^n$ but this is necessary as to implement a BH we have to define some neighbourhood in which we care about and to do this we need to define it around some point on our hyperplane. So consider the folowing situation, we are learning the vector $n = [n_1, ..., n_n]$ normal to our hyperplane, the vector $a = [a_1, ..., a_n]$ which is on our hyperplane and finally b which is our byas. We define $x = [x_1, ..., x_n]$ as our inputs making our weighted sum

$$z = \sum_{i=1}^{n} n_i * (a_i - x_i)$$

We are still using SSE for loss and Sigmoid for actvation.

1.2.1 Deriving Gradients

$$\frac{\partial E}{\partial W_a} = \frac{\partial E}{\partial o}\frac{\partial o}{\partial z}\frac{\partial z}{\partial W_a} \tag{7}$$

$$\frac{\partial o}{\partial z} = o(1-o) \tag{8}$$

$$\frac{\partial E}{\partial o} = -(t-o) \tag{9}$$

$$\frac{\partial z}{\partial a_i} = n_i \tag{10}$$

$$\frac{\partial z}{\partial n_i} = (a_i - x_i) \tag{11}$$

$$\frac{\partial z}{\partial b} = 1 \tag{12}$$

1.2.2 Comparason To Perceptron

Like before we compare the two paramaterisations using the same data and over 50000 iterations in the 2D case. In the graph for the modified perceptron the green point is our vector on the hyperplane.

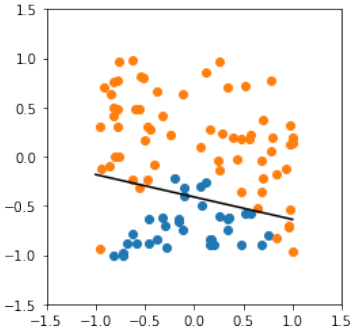

Figure 4: Standard Perceptron (SSE = 3.90)

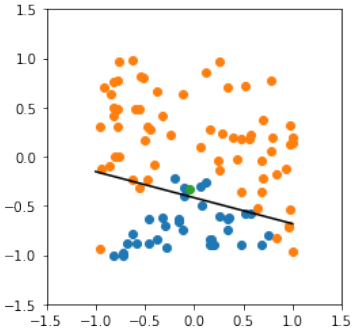

Figure 5: Modified Perceptron (SSE = 3.90)

This modified perceptron has all the quallitys needed to proceid with creating the boundary hunters, whats better is this also generalizes to the $\mathbb{R}^n$ case.

2 Boundary Hunter

Using the **Normal & Point Paramaterisation** from before we construct boundary hunters, first we have a simple case where the area of intererest for each boundary hunter is simpley a circle with fixed radius, our goal is to make this area of interest an ellipse which the BH will learn.

2.1 Simplified Case