The Maths behind Neural Networks

Alex Punnen

Contents

The Maths behind Neural Networks

Alex Punnen
© All Rights Reserved

Contents

Chapter 2

Perceptron Training via Feature Vectors & HyperPlane split

Let's follow from the previous chapter of the Perceptron neural network.

We have seen how the concept of splitting the hyper-plane of feature set separates one type of feature vectors from other.

How are the weights learned ?

You may have heard about Gradient descent. For Perceptron leaning is much simpler.

What is done is to start with a randomly initialized weight vector, compute a resultant classification (0 or 1) by taking the dot product with the input feature vector, and **then adjust the weight vector by a tiny bit to the**

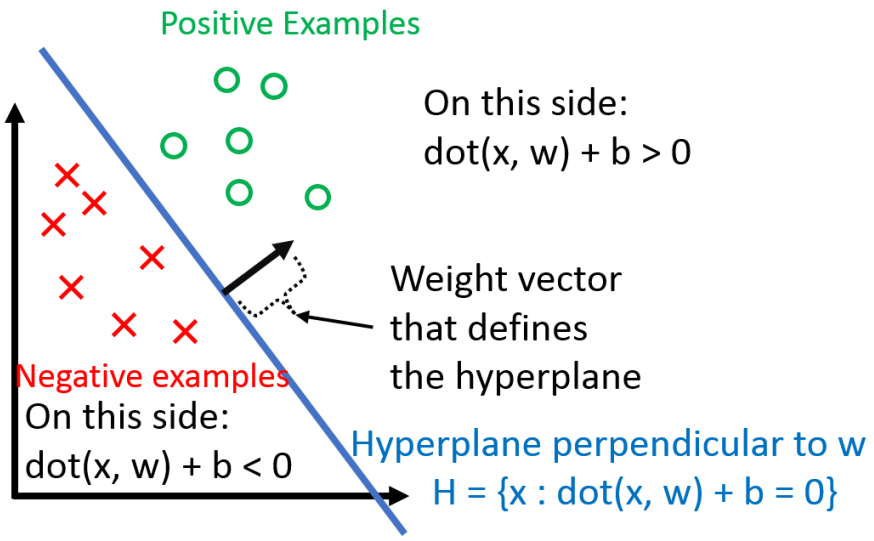

Figure 1: cornellperceptron

right 'direction' so that the output is closer to the expected value. Do this iteratively until the output is close enough.

Question is how to nudge to the correct "direction"?

We want to rotate the weight vector to the direction of the input vector so that the hyperplane is closer to the correct classification.

The error of a perceptron with weight vector w is the number of incorrectly classified points. The learning algorithm must minimize this *error function* $E(w)$

One possible strategy is to use a local greedy algorithm which works by computing the error of the perceptron for a given weight vector, looking then for a **direction in weight space** in which to move and update the weight vector.

Taking input from the training example, and doing a dot product with the weight vector; will give you either a value greater that 0 or less than 0.

Note that this means which quadrant the feature vector lies; either in the positive quadrant (P) or on the negative side (N).

If this is as expected, then do nothing. If the dot product comes wrong, that is if input feature vector - say x, was $x \in P$, but dot product $w.x < 0$, we need to drag/rotate the weight vector towards x.

$w_n = w + x$

Which is vector addition, that is w is moved towards x. Say that $x \in N$, but dot product $w.x > 0$, then we need to do the reverse $w_n = w - x$

This is the classical method of perceptron learning

$$w_j = w_j + \delta w_j \delta w_j = \begin{cases} 0 & \text{if instance is classified correctly} \\ +x_j & \text{if Positive instance is classified as negative} \\ -x_j & \text{if Negative instance is classified as positive} \end{cases}$$

This is also called the delta rule. Note that there is some articles that refer to this as gradient descent simplified. But gradient descent depends on the activation function being differentiable. The step function which is the activation function of the perceptron in non continuous and hence non differentiable.

A more rigorous explanation of the proof is here from the book Neural Networks by R.Rojas and more lucid explanation here perceptron-learning-algorithm

The Perceptron Network and the AI winter

The Perceptron network needs the input feature set to be linearly seperable. However all problems do not have their feature set which is linearly seperable. So this is a constraint of this system.

The fact that Perceptron could not be trained for XOR or XNOR; which was demonstrated in 1969, by by Marvin Minsky and Seymour Papert led to the first *AI winter*, as much of the hype generated intially by Frank Rosenblatt's discovery became a disillusionment.

Next up we wil see a Modern Neural Network

Next: Gradient Descent and Optimization

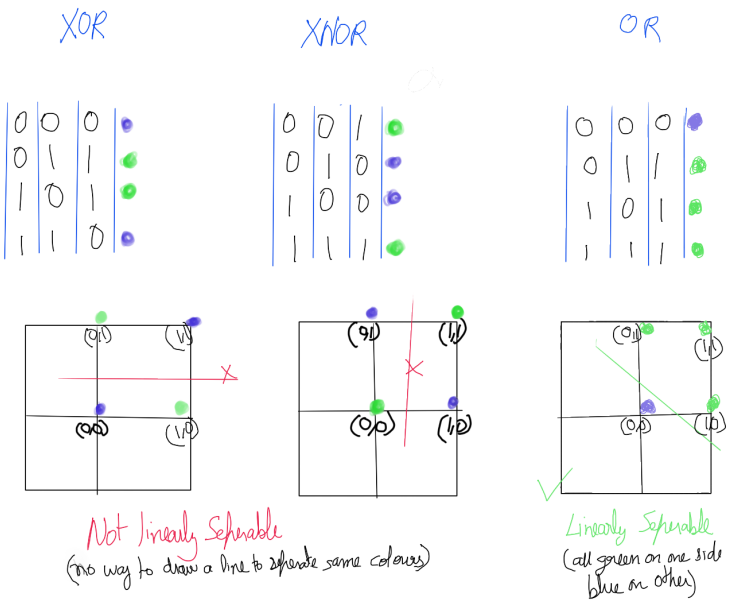

Figure 2: linearseperable