The Maths of Machine Learning and Nerual Networks

- Alex Punnen
- 2019-2020

Feature Vectors, Dot products and Perceptron Training

Q. How can we train the Perceptron with just simple vector maths?

Let's follow from the previous chapter of the Perceptron neural network. Please read it if not done as the importance of vector representation (direction part) and what a vector dot product signifies is needed here again.

Just to reiterate - Dot product, geometrically, it is the product of the Euclidean magnitudes of the two vectors and the cosine of the angle between them \begin{aligned}

$$\vec{a} \cdot \vec{b} = \|\vec{a}\| \ \left\|\vec{b}\right\| \cos\theta$$

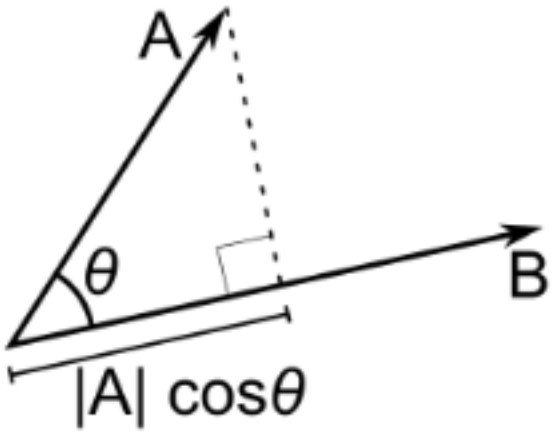

Figure 1: dotproduct

If two vectors are in the same direction the dot product is positive and if they are in the opposite direction the dot product is negative. Why? As Cos(θ) is positive in the first quadrant* and negative in the second quadrant. *See this excellent answer here to refresh your trigonometry https://www.quora.com/Why-is-sin-90-taken-to-be-1

Now let us see why we want a weight vector first - once again and then we go to how it is obtained.

Assume that there are two sets of feature vectors P and N, where P is set of positive samples (say cat features) and N is non-positve cases (say non-cat features) and P and N are the correctly classified **Training Set**.

We can see how hyperplane that is created orthogonal (perpendicular) to the weight vector splits input feature vector space, into two distinct regions.

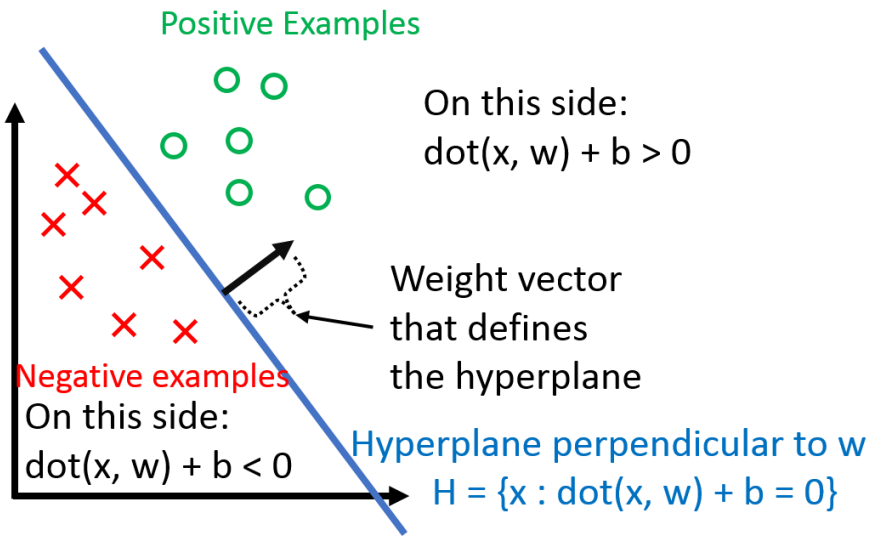

Figure 2: cornellperceptron

Or in a better way, which shows the vectors properly

So that if we get this weight vector trained with sufficient know samples of P and N, when an unknown vector comes in, we can take a dot product of the unknown feature vector with the learned weight vector and find out in which region it falls, in the cat feature region -P or the non-cat feature region - N.

How are the weights learned?

You may have heard about Gradient descent. But hold your horses. For perceptron leaning it is much simpler. We will go there still, but later.

Basically what is done is to start with a randomly initialised 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**

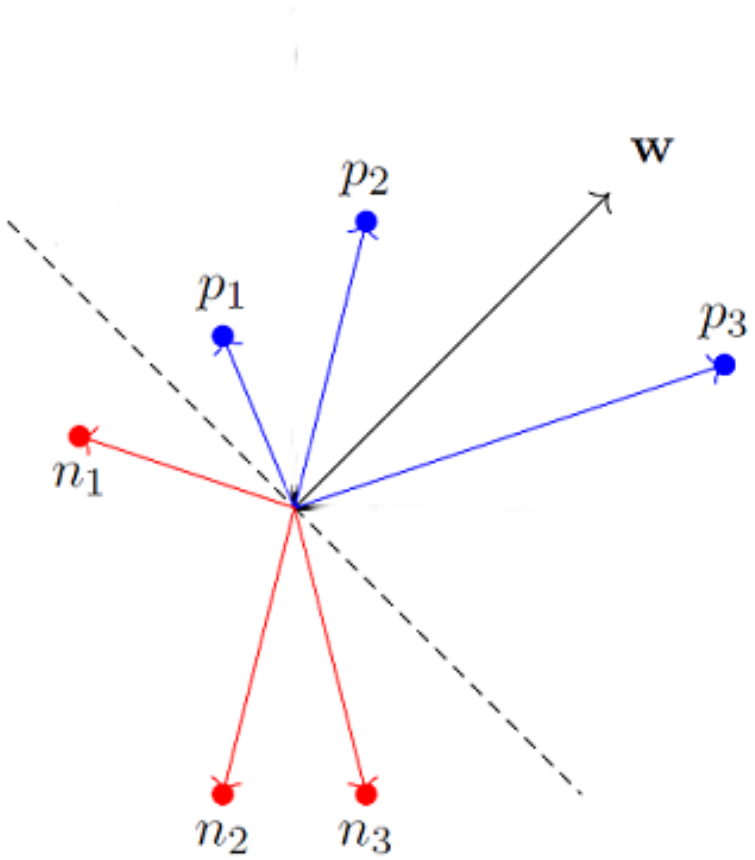

Figure 3: weighvector

the 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 by selecting new weights in the selected search direction.

Taking input from the training example and doing a dot product with the weight vector; will give you either a value $>= 0$ or < 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 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

$$\delta w_j = \begin{cases} 0 & \text{if instance is classified correctly} \\ +x_j & \text{if } +1 \text{ instance is classified as } -1 \\ -x_j & \text{if -1 instance is classified as } +1 \end{cases}$$

ref 5

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

---