CHAPTER 3

The Perceptron

First of all, the coolest algorithm name! It is based on the 1943 model of neurons made by McCulloch and Pitts and by Hebb. It was developed by Rosenblatt in 1962. At the time, it was not interpreted as attempting to optimize any particular criteria; it was presented directly as an algorithm. There has, since, been a huge amount of study and analysis of its convergence properties and other aspects of its behavior.

Well, maybe "neocognitron," also the name of a real ML algorithm, is cooler.

1 Algorithm

Recall that we have a training dataset $\mathcal{D}_n$ with $x \in \mathbb{R}^d$, and $y \in \{-1, +1\}$. The Perceptron algorithm trains a binary classifier $h(x; \theta, \theta_0)$ using the following algorithm to find θ and θ_0 using τ iterative steps:

We use Greek letter τ here instead of T so we don't confuse it with transpose!

PERCEPTRON$(\tau, \mathcal{D}_n)$

```
1  θ = [0  0  ···  0]^T
2  θ_0 = 0
3  for t = 1 to τ
4      for i = 1 to n
5          if y^(i) (θ^T x^(i) + θ_0) ⩽ 0
6              θ = θ + y^(i) x^(i)
7              θ_0 = θ_0 + y^(i)
8  return θ, θ_0
```

Intuitively, on each step, if the current hypothesis θ, θ_0 classifies example $x^{(i)}$ correctly, then no change is made. If it classifies $x^{(i)}$ incorrectly, then it moves θ, θ_0 so that it is "closer" to classifying $x^{(i)}, y^{(i)}$ correctly.

Let's check dimensions. Remember that θ is $d \times 1$, $x^{(i)}$ is $d \times 1$, and $y^{(i)}$ is a scalar. Does everything match?

Note that if the algorithm ever goes through one iteration of the loop on line 4 without making an update, it will never make any further updates (verify that you believe this!) and so it should just terminate at that point.

Study Question: *What is true about $\mathcal{E}_n$ if that happens?*

Example: Let h be the linear classifier defined by $\theta^{(0)} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \theta_0^{(0)} = 1$. The diagram below shows several points classified by h. However, in this case, h (represented by the bold line) misclassifies the point $x^{(1)} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$ which has label $y^{(1)} = 1$. Indeed,

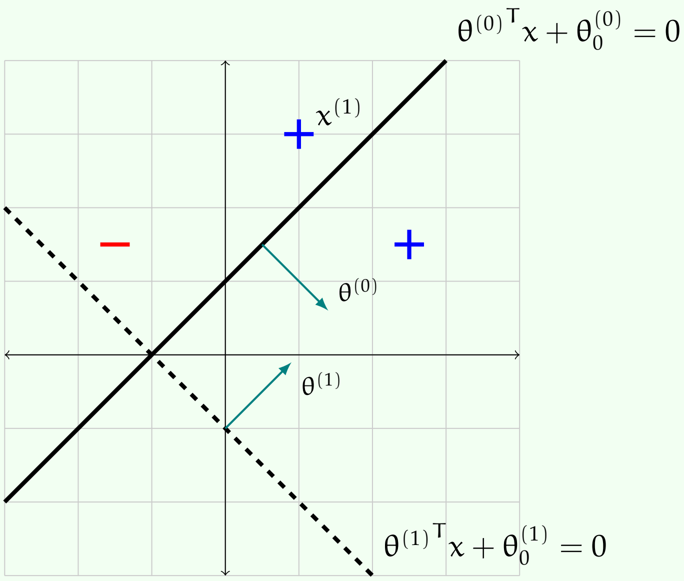
$$y^{(1)} \left(\theta^T x^{(1)} + \theta_0 \right) = \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \end{bmatrix} + 1 = -1 < 0$$

By running an iteration of the Perceptron algorithm, we update

$$\theta^{(1)} = \theta^{(0)} + y^{(1)} x^{(1)} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$\theta_0^{(1)} = \theta_0^{(0)} + y^{(1)} = 2$$

The new classifier (represented by the dashed line) now correctly classifies that point, but now makes a mistake on the negatively labeled point.



A really important fact about the perceptron algorithm is that, if there is a linear classifier with 0 training error, then this algorithm will (eventually) find it! We'll look at a proof of this in detail, next.

2 Offset

Sometimes, it can be easier to implement or analyze classifiers of the form

$$h(x; \theta) = \begin{cases} +1 & \text{if } \theta^T x > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Without an explicit offset term (θ_0), this separator must pass through the origin, which may appear to be limiting. However, we can convert any problem involving a linear separator *with* offset into one with *no* offset (but of higher dimension)!

Consider the d-dimensional linear separator defined by $\theta = \begin{bmatrix}\theta_1 & \theta_2 & \cdots & \theta_d\end{bmatrix}$ and offset θ_0.

- to each data point $x \in \mathcal{D}$, append a coordinate with value +1, yielding

$$x_{new} = \begin{bmatrix}x_1 & \cdots & x_d & +1\end{bmatrix}$$

- define

$$\theta_{new} = \begin{bmatrix}\theta_1 & \cdots & \theta_d & \theta_0\end{bmatrix}$$

Then,

$$\begin{aligned}\boxed{\theta_{new}^T \cdot x_{new}} &= \theta_1 x_1 + \cdots + \theta_d x_d + \theta_0 \cdot 1\\ &= \boxed{\theta^T x + \theta_0}\end{aligned}$$

Thus, θ_{new} is an equivalent ((d + 1)-dimensional) separator to our original, but with no offset.

Consider the data set:

$$\begin{aligned}X &= [[1], [2], [3], [4]]\\ Y &= [[+1], [+1], [-1], [-1]]\end{aligned}$$

It is linearly separable in $d = 1$ with $\theta = [-1]$ and $\theta_0 = 2.5$. But it is not linearly separable through the origin. Now, let

$$X_{new} = \begin{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} & \begin{bmatrix}2\\1\end{bmatrix} & \begin{bmatrix}3\\1\end{bmatrix} & \begin{bmatrix}4\\1\end{bmatrix}\end{bmatrix}$$

This new dataset is separable through the origin, with $\theta_{new} = [-1, 2.5]^T$.

We can make a simplified version of the perceptron algorithm if we restrict ourselves to separators through the origin:

We list it here because this is the version of the algorithm we'll study in more detail.

PERCEPTRON-THROUGH-ORIGIN($\tau, \mathcal{D}_n$)

```
θ = [0  0  ···  0]^T
for t = 1 to τ
    for i = 1 to n
        if y^(i) (θ^T x^(i)) ⩽ 0
            θ = θ + y^(i) x^(i)
return θ
```

3 Theory of the perceptron

Now, we'll say something formal about how well the perceptron algorithm really works. We start by characterizing the set of problems that can be solved perfectly by the perceptron algorithm, and then prove that, in fact, it can solve these problems. In addition, we provide a notion of what makes a problem difficult for perceptron and link that notion of difficulty to the number of iterations the algorithm will take.

3.1 Linear separability

A training set $\mathcal{D}_n$ is *linearly separable* if there exist θ, θ_0 such that, for all $i = 1, 2, \dots, n$:

Predicted Val

$$y^{(i)}\left(\theta^T x^{(i)} + \theta_0\right) > 0 \ .$$

Another way to say this is that all predictions on the training set are correct:

$$h(x^{(i)}; \theta, \theta_0) = y^{(i)} \ .$$

And, another way to say this is that the training error is zero:

$$\mathcal{E}_n(h) = 0 \ .$$

3.2 Convergence theorem

The basic result about the perceptron is that, if the training data $\mathcal{D}_n$ is linearly separable, then the perceptron algorithm is guaranteed to find a linear separator.

If the training data is *not* linearly separable, the algorithm will not be able to tell you for sure, in finite time, that it is not linearly separable. There are other algorithms that can test for linear separability with run-times $O(n^{d/2})$ or $O(d^{2n})$ or $O(n^{d-1} \log n)$.

We will more specifically characterize the linear separability of the dataset by the *margin* of the separator. We'll start by defining the margin of a point with respect to a hyperplane. First, recall that the distance of a point x to the hyperplane θ, θ_0 is

Margin of Point

$$\frac{\theta^T x + \theta_0}{\|\theta\|} \ .$$

Then, we'll define the *margin* of a *labeled point* (x, y) with respect to hyperplane θ, θ_0 to be

Margin of labelled point

$$y \cdot \frac{\theta^T x + \theta_0}{\|\theta\|} \ .$$

This quantity will be positive if and only if the point x is classified as y by the linear classifier represented by this hyperplane.

Study Question: What sign does the margin have if the point is incorrectly classified? Be sure you can explain why.

Now, the *margin* of a *dataset* $\mathcal{D}_n$ with respect to the hyperplane θ, θ_0 is the *minimum* margin of any point with respect to θ, θ_0:

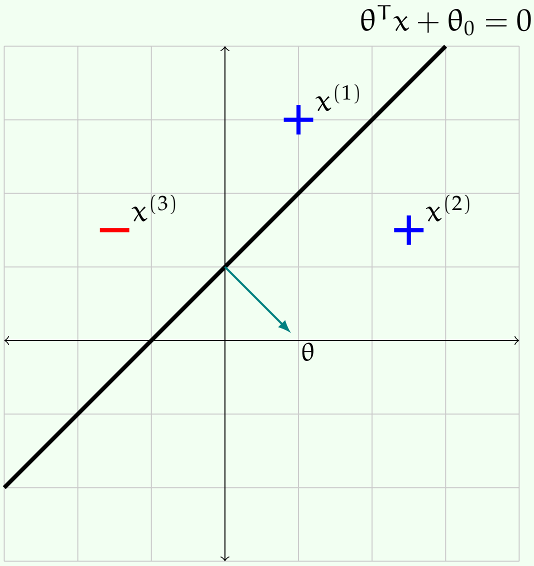
Margin of Dataset

$$\min_i \left(y^{(i)} \cdot \frac{\theta^T x^{(i)} + \theta_0}{\|\theta\|} \right) \ .$$

The margin is positive if and only if all of the points in the data-set are classified correctly. In that case (only!) it represents the distance from the hyperplane to the closest point.

Example: Let h be the linear classifier defined by $\theta = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \theta_0 = 1$.

The diagram below shows several points classified by h, one of which is misclassified. We compute the margin for each point:



$$y^{(1)} \cdot \frac{\theta^T x^{(1)} + \theta_0}{\|\theta\|} = 1 \cdot \frac{-2+1}{\sqrt{2}} = -\frac{\sqrt{2}}{2}$$

$$y^{(2)} \cdot \frac{\theta^T x^{(2)} + \theta_0}{\|\theta\|} = 1 \cdot \frac{1+1}{\sqrt{2}} = \sqrt{2}$$

$$y^{(3)} \cdot \frac{\theta^T x^{(3)} + \theta_0}{\|\theta\|} = -1 \cdot \frac{-3+1}{\sqrt{2}} = \sqrt{2}$$

Note that since point $x^{(1)}$ is misclassified, its margin is negative. Thus the margin for the whole data set is given by $-\frac{\sqrt{2}}{2}$.

Theorem 3.1 (Perceptron Convergence). *For simplicity, we consider the case where the linear separator must pass through the origin. If the following conditions hold:*

(a) *there exists* θ^* *such that* $y^{(i)} \frac{\theta^{*T} x^{(i)}}{\|\theta^*\|} \geqslant \gamma$ *for all* $i = 1, \ldots, n$ *and for some* $\gamma > 0$ *and*

(b) *all the examples have bounded magnitude:* $\|x^{(i)}\| \leqslant R$ *for all* $i = 1, \ldots n$,

then the perceptron algorithm will make at most $\left(\frac{R}{\gamma}\right)^2$ *mistakes. At this point, its hypothesis will be a linear separator of the data.*

Proof. We initialize $\theta^{(0)} = 0$, and let $\theta^{(k)}$ define our hyperplane after the perceptron algorithm has made k mistakes. We are going to think about the angle between the hypothesis we have now, $\theta^{(k)}$ and the assumed good separator θ^*. Since they both go through the origin, if we can show that the angle between them is decreasing usefully on every iteration, then we will get close to that separator.

So, let's think about the cos of the angle between them, and recall, by the definition of dot product:

$$\cos\left(\theta^{(k)}, \theta^*\right) = \frac{\theta^{(k)} \cdot \theta^*}{\|\theta^*\| \left\|\theta^{(k)}\right\|}$$

We'll divide this up into two factors,

$$\cos\left(\theta^{(k)}, \theta^*\right) = \left(\frac{\theta^{(k)} \cdot \theta^*}{\|\theta^*\|}\right) \cdot \left(\frac{1}{\left\|\theta^{(k)}\right\|}\right) \quad , \tag{3.1}$$

and start by focusing on the first factor.

Without loss of generality, assume that the k^{th} mistake occurs on the i^{th} example $\left(x^{(i)}, y^{(i)}\right)$.

$$\begin{aligned}
\frac{\theta^{(k)} \cdot \theta^*}{\|\theta^*\|} &= \frac{\left(\theta^{(k-1)} + y^{(i)} x^{(i)}\right) \cdot \theta^*}{\|\theta^*\|} \\
&= \frac{\theta^{(k-1)} \cdot \theta^*}{\|\theta^*\|} + \frac{y^{(i)} x^{(i)} \cdot \theta^*}{\|\theta^*\|} \\
&\geqslant \frac{\theta^{(k-1)} \cdot \theta^*}{\|\theta^*\|} + \gamma \\
&\geqslant k\gamma
\end{aligned}$$

where we have first applied the margin condition from (a) and then applied simple induction.

Now, we'll look at the second factor in equation 3.1. We note that since $\left(x^{(i)}, y^{(i)}\right)$ is classified incorrectly, $y^{(i)}\left({\theta^{(k-1)}}^T x^{(i)}\right) \leqslant 0$. Thus,

$$\begin{aligned}
\left\|\theta^{(k)}\right\|^2 &= \left\|\theta^{(k-1)} + y^{(i)} x^{(i)}\right\|^2 \\
&= \left\|\theta^{(k-1)}\right\|^2 + 2y^{(i)} {\theta^{(k-1)}}^T x^{(i)} + \left\|x^{(i)}\right\|^2 \\
&\leqslant \left\|\theta^{(k-1)}\right\|^2 + R^2 \\
&\leqslant kR^2
\end{aligned}$$

where we have additionally applied the assumption from (b) and then again used simple induction.

Returning to the definition of the dot product, we have

$$\cos\left(\theta^{(k)}, \theta^*\right) = \frac{\theta^{(k)} \cdot \theta^*}{\left\|\theta^{(k)}\right\| \|\theta^*\|} = \left(\frac{\theta^{(k)} \cdot \theta^*}{\|\theta^*\|}\right) \frac{1}{\left\|\theta^{(k)}\right\|} \geqslant (k\gamma) \cdot \frac{1}{\sqrt{k}R} = \sqrt{k} \cdot \frac{\gamma}{R}$$

Since the value of the cosine is at most 1, we have

$$\begin{aligned}
1 &\geqslant \sqrt{k} \cdot \frac{\gamma}{R} \\
k &\leqslant \left(\frac{R}{\gamma}\right)^2 .
\end{aligned}$$

□

This result endows the margin γ of $\mathcal{D}_n$ with an operational meaning: when using the Perceptron algorithm for classification, at most $(R/\gamma)^2$ classification errors will be made, where R is an upper bound on the magnitude of the training vectors.