Perceptrons

The perceptron is a **binary classification** model. The idea is to learn the equation of a **hyperplane** (fancy word for an n-dimensional line) that separates the two groups of data points.

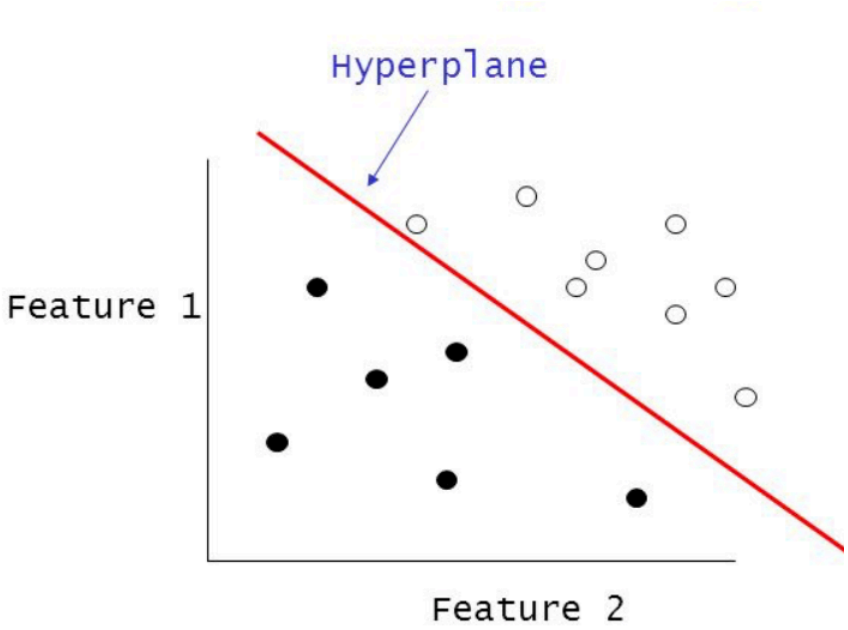

Figure 1: A 2-dimensional hyperplane, more commonly referred to as a line. The perceptron learning algorithm will attempt to learn the equation of this line from the data.

From Neurons to a Model

Folk biology tells us that our brains are made up of a bunch of little units, called **neurons**, that send electrical signals to one another. The *rate* that these signals are firing tells us how **activated** a neuron is.

A single neuron usually has incoming neurons, which can be thought of as **inputs**.

- These incoming neurons are firing at different rates (i.e., have different **activations**).
- Based on how much these incoming neurons are firing, and how "strong" the neural connections are, our main neuron will "decide" how strongly it wants to fire. And so on through the whole brain.

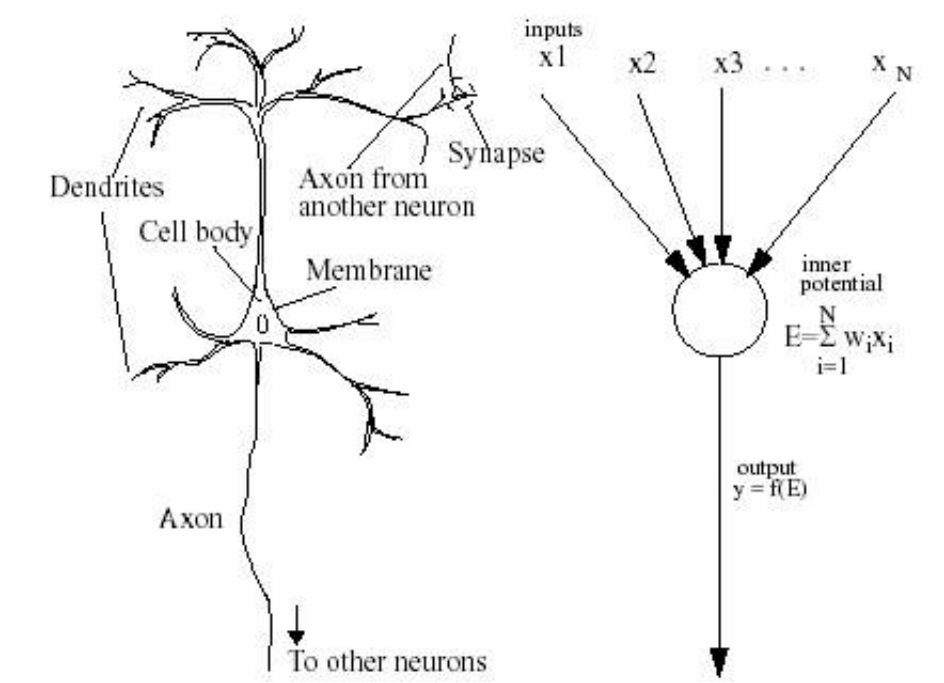

Figure 2: Neuron vs. Perceptron

The perceptron is sometimes thought of as a **single neuron**. It receives a number of inputs, and based on some sort of **activation function**, decides whether or not it will **fire**.

Practically:

- If the output of the activation function is positive, then we classify our training example $\vec{x}$ as a part of the group $y = 1$.
- If the output is negative, then we classify it as a part of the group $y = -1$.

Mathematically:

- An input vector $\vec{x} \in \mathbb{R}^n$ arrives.
- The perceptron/neuron stores n weights, $\vec{\theta} \in \mathbb{R}^n$.
- The neuron computes the sum $a = \sum_i x_i \theta_i$, where a is its "amount" of **activation**.
  - If $a > 0$, we predict this example belongs to the group $y = 1$.
  - Otherwise, we predict this example belongs to the group $y = -1$.
- It's often more convenient to have a non-zero threshold. So, we introduce a **bias** term so that the activation is always increased by some fixed value.
- We finally arrive at the model:

$$a = \left(\sum_i x_i \theta_i \right) + b$$

Perceptron Learning Algorithm

The perceptron learning algorithm is:

- **Online**, meaning it only looks at one data point at a time rather than the entire dataset.
- **Error-driven**, meaning that as long as it's doing well (i.e. classifying training samples correctly) it doesn't bother updating its weights.

The algorithm is driven by this update rule:

For a given training example $(\vec{x}, y)$:

- Calculate $a = \left(\sum_i x_i \theta_i \right) + b$
- If $y \times a \leq 0$:
  - $\vec{\theta}_{\text{new}} = \vec{\theta}_{\text{old}} + y\vec{x}$

Notes:

- The product $y \times a$ is **positive** if and only if the perceptron's prediction is correct.
- Intuitively, the goal of the update is to adjust our weights so that they're "better" for the current example
- To understand the update rule, consider how a for the current example changes:

  Let $a', \vec{\theta}', b'$ denote the new values of $a, \vec{\theta}, b$.

  - If $a < 0$ but $y = 1$, we would like to **increase** a:

$$\begin{aligned} a' &= \left(\sum_i \theta'_i x_i \right) + b' \\ &= \left(\sum_i (\theta_i + y x_i) x_i \right) + (b + y) \\ &= \left(\sum_i (\theta_i + x_i) x_i \right) + (b + 1) \\ &= \left(\sum_i \theta_i x_i + x_i^2 \right) + b + 1 \\ &= \left(\sum_i \theta_i x_i \right) + b + \left(\sum_i x_i^2 \right) + 1 \\ &= a + \left(\sum_i x_i^2 \right) + 1. \end{aligned}$$

    Our new a is being **increased by a strictly positive amount**, which is what we want!

  - If $a > 0$ but $y = -1$, we would like to **decrease** a:

$$
\begin{aligned}
a' &= \left(\sum_i \theta'_i x_i \right) + b' \\
&= \left(\sum_i (\theta_i + y x_i) x_i \right) + (b + y) \\
&= \left(\sum_i (\theta_i - x_i) x_i \right) + (b - 1) \\
&= \left(\sum_i \theta_i x_i - x_i^2 \right) + b - 1 \\
&= \left(\sum_i \theta_i x_i \right) + b - \left(\sum_i x_i^2 \right) - 1 \\
&= a - \left(\sum_i x_i^2 \right) - 1.
\end{aligned}
$$

Our new a is being **decreased by a strictly positive amount**, which is what we want!

So the full algorithm is as follows:

```
Initialize theta to 0s
Initialize b to 0
For each data point (x, y) in the training dataset:
    Calculate a = x . theta + b
    If ya <= 0:
        theta += yx
        b += y
```

Geometric Interpretation

What is the **decision boundary**, i.e. that separating hyperplane, found by the perceptron algorithm?

For a perceptron, the decision boundary is where the sign of the activation, a, changes from -1 to $+1$. In other words, it is the set of points $\vec{x}$ that achieve **zero activation**. Mathematically, we can write:

$$
\mathcal{W}(\vec{x}) : \left(\sum_i x_i \theta_i \right) + b = 0
$$

$$
\mathcal{W}(\vec{x}) : \vec{x} \cdot \vec{\theta} + b = 0
$$

What we have here is the equation of a plane! $\vec{\theta}$, our weight vector, is orthogonal to this plane. What this means is that if we multiply $\vec{\theta}$ by a scalar, the plane it defines will not change. Therefore, it is common to **normalize** $\vec{\theta}$ such that it is a unit vector.

Convergence

We define **convergence** for the perceptron algorithm as making an entire pass through the training data **without making any updates**, meaning that it has correctly classified every single example.

In this case, the perceptron algorithm will converge **if and only if the data is linearly separable**, otherwise it will never converge. Essentially, this means that if a separating hyperplane exists, then the perceptron algorithm will find it.