Perceptrons

The **perceptron** is a machine that does supervised learning, that is, it makes guesses, is told whether or not its guess is correct, and then makes another guess. They were first discovered in 1957 by Frank Rosenblatt [1] and introduced to the world with great fanfare, it was claimed that they would solve problems from object recognition to consciousness: if you consider the perceptron as the forebearer of the deep learning network, then perhaps we don't know if these claims will be fulfilled, but we do know that the original perceptron proved quite limited in artificial intelligence. It does, however, appear to describe some neuronal processes, if we ignore the implementation details.

A perceptron is made of two layers of neurons, an input layer and an output layer of McCulloch-Pitts neurons. For simplicity let's assume the output layer has a single neuron. Now, if the input is given by $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ the output, y, is

$$y = \phi(r) \tag{1}$$

where

$$r = \sum_j w_j x_j - \theta = \mathbf{w}^\top \mathbf{x} - \theta \tag{2}$$

and ϕ here is the Heaviside-like threshold activation function described before. Now for a given input, if the actual value of the output should be d the error is $d - y$. The perceptron learning rule is to change the w_j weight by an amount proportional to the error and how much x_j was 'to blame' for the error:

$$\Delta w_j = \eta(d - y)x_j \tag{3}$$

and

$$\Delta\theta = \eta(d - y) \tag{4}$$

where η is some small learning rate and

$$\begin{aligned} w_j &\rightarrow w_j + \Delta w_j \\ \theta &\rightarrow \theta + \Delta\theta \end{aligned} \tag{5}$$

You can see how this might work, if x_j was positive and y was too big, this would make w_j smaller so in future y would be smaller when it had the same input.

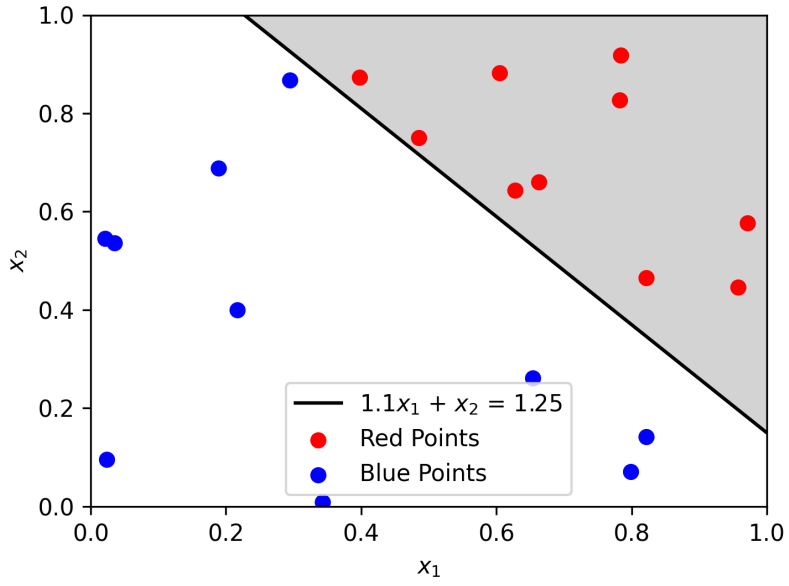

Figure 1: Here the shaded region corresponds to $1.1x_1+x_2 > 0.25$, so if $w_1 = 1.1$ and $w_2 = 1$ with $\theta = 0.25$ then the corresponding perceptron neuron will be one for all the circle points and -1 for all the square points.

In fact, the perceptron can only solve problems with a linear classifier: if we think of the x_i's as parametrizing an n-dimensional space then $\sum_i w_i x_i = \theta$ is a hyperplane in that space, so a pattern x is classified one way or the other according to which side of that hyperplane it lies, see for example Fig. 1. Thus, the perceptron works only if there is a hyperplane dividing the data into two, with one class of data on one side and one on the other. In fact, if the data is linearly separable the perceptron is guaranteed to converge to a solution which manages this separation. Now, as illustrated in Fig. 2, there will not be a unique hyper-plane separating the two classes and the perceptron won't, typically, find what you might regard as the 'best' hyperplane, where by best you might mean the line that is, in some sense, as far from the individual data points as possible; finding that best hyperplane is the idea behind the support vector machine.

The perceptron learning rule can be motivated by thinking about error min-

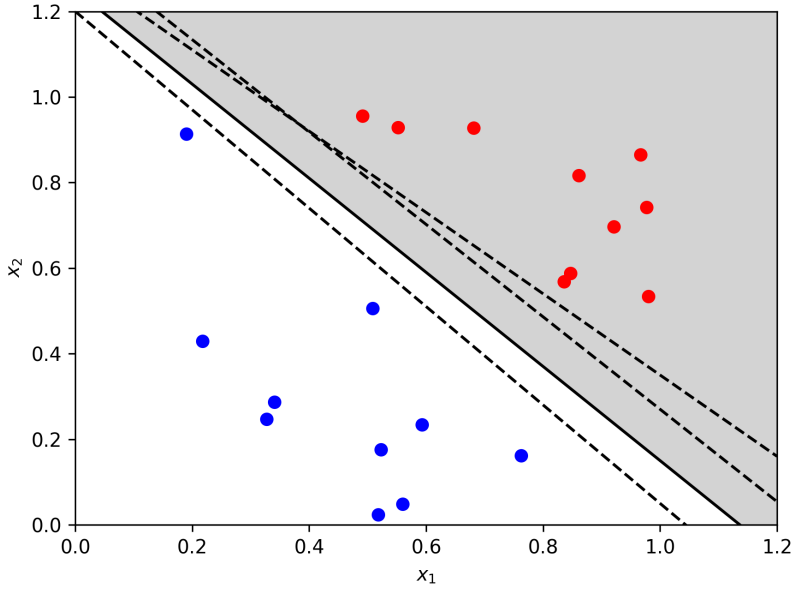

Figure 2: All of these lines separate the points into two classes

imization. Consider an error function

$$E = \tfrac{1}{2}\langle (y_a^* - \hat{y}_a)^2 \rangle_a \tag{6}$$

In Equation (6) above,

- The subscript a indexes individual input–output training pairs $(\mathbf{x}_a, y_a^*)$,
- y_a^* is a **desired** or **target** activation,
- $\hat{y}_a$ is the **predicted** output from the perceptron computed via Equations (1) and (2) for input vector $\mathbf{x}^a$,
- The angle brackets $\langle \cdot \rangle_a$ denote an average over training pairs.

Now consider the gradient of E with w_i:

$$\frac{\partial E}{\partial w_i} = -\left\langle \frac{\partial \hat{y}_a}{\partial w_i} \cdot (y_a^* - \hat{y}_a) \right\rangle_a \tag{7}$$

If we ignore for the moment the fact that the activation function for the McCulloch-Pitt neuron isn't differentiable and write

$$\hat{y} = \phi(r) = \phi\left(\mathbf{w}^\top \mathbf{x} - \theta\right) \tag{8}$$

we have

$$\frac{\partial \hat{y}_a}{\partial w_i} = \frac{\partial \phi}{\partial r} \cdot \frac{\partial r}{\partial w_i} \tag{9}$$

so using

$$\frac{\partial r}{\partial w_i} = x_i \tag{10}$$

we get

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= -\left\langle \frac{\partial \phi}{\partial r_a} \cdot \frac{\partial r_a}{\partial w_i} \cdot (y_a^* - \hat{y}_a) \right\rangle_a \\ &= -\left\langle \frac{\partial \phi}{\partial r_a} \cdot x_{a;i} \cdot (y_a^* - \hat{y}_a) \right\rangle_a \end{aligned} \tag{11}$$

Of course, in the McCulloch-Pitts case $\frac{\partial \phi}{\partial r_a}$ is either zero or undefined, but we can pretend that $\frac{\partial \phi}{\partial r_a} = 1$ (a form of **surrogate gradient**). This gives a similar learning rule: one way to reduce the error is to move a tiny amount in the opposite direction to the gradient, the gradient is the direction along which the value increases quickest. The actual perceptron rule we gave updates a small bit after every presentation rather than averaging first over a collection of presentations, both approaches make sense. Here we have dealt with the weights, the same approach can be applied to the threshold θ.

To side-step this messy business of non-differentiable activation function, most modern perceptron neural networks use smooth (or mostly smooth) activation functions. However, you will still see various surrogate-gradient approaches applied to incorporate steps or spikes in model neural networks for biological plausibility.

The basic limitation of the perceptron is that it has only one layer and so only learns a linear classification; these days artificial neural networks have more than one layer; this complicates the idea of adjusting the w_i in a way that is weighted by x_i, this, in a sense, changing the weight according to how much it is 'to blame' for the error. Back propagation resolves this, it can be thought of as propagating the error backwards from layer to layer; although another way to think of it is as doing gradient descent on all the weights. At the moment the back propagation algorithm is not considered very biological, though it is very possible that when we understand why deep learning networks are so effective it will be clear that the important aspects of the algorithm can be recognized in neuronal dynamics.

References

[1] Rosenblatt, F. (1958), The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Cornell Aeronautical Laboratory. Psychological Review, 65:386–408.