Overview of the Perceptron and MLP

Mario Sánchez García

November 6, 2017

Abstract

Summary of Chapter 4: Artificial Neural Networks from Tom M. Mitchell's book ""Machine Learning"", 1997.

Contents

1 The perceptron

Within the Artificial Neural Networks (ANN), we can find great variety of units that define the type of ANN involved, but this time we are going to use a perceptron. It has several inputs, it performs an operation that involves all the values obtained from the inputs and calculate a single value as a result in a single output.

1.1 Structure

It has several inputs, composed of a value and a weight, which will indicate the contribution that will contribute to the output of the perceptron. The operation involving all inputs is a linear combination:

$$y = \sum_{i=0}^{n} w_n x_n = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + ... \tag{1}$$

where x_n are the inputs of the perceptron and w_n are the weights($x_0 = 1$ is not an input, it's an useful constant to allow us to write the linear operation with a summatory). Finally, the (unique) output of the perceptron depends on the result of the linear operation, being 1 if the result is positive; or -1 in another case. All these data allow us to define the perceptron as follows:

$$perceptron(y) = \begin{cases} 1 & \text{if } y > 0 \\ -1 & \text{otherwise} \end{cases} \tag{2}$$

1.2 Linearly separable attribute

With a 2 input perceptron, we can produce a Cartesian map with all the inputs that we're going to use (first input would be X axis, second input Y axis). Instead of using a point to represent the input, we use the output symbol (“”+“” or “”-“”).

If we can draw a line that divides the map in two sides, so that on each side of the line there was only one kind of symbol (all positives or all negatives), then we can say that the set of points we have used is linearly separable. The fact of using only two inputs is because we can imagine a scenario that allows us to easily understand the attribute of being linearly separable. This attribute can be achieved with any number of inputs if there is a hyperplane that separate the values in the same way as in the example.

Therefore, with a single perceptron there are sets of points that can't be linearly separable. That's why the networks of perceptrons with more than one layer are created, whose first layer perceptron's outputs is not the final result, but performs the role as input to another perceptron in the next layer. This way, even the non-linear surfaces can be represented by the perceptrons.

2 Training perceptrons

2.1 Training Rule

We begin assigning random values to the weights and then we apply the perceptron to each training example. If the perceptron misclassifies an example, the training rule will modify the

weights according to:

$$w_i^{'} \leftarrow w_i + \Delta w_i \qquad ; \qquad \Delta w_i = \eta(t-o)x_i$$

$$w_i^{'} \leftarrow w_i + \eta(t-o)x_i \tag{3}$$

where η is the learning rate (constant usually small that determines how much weights vary in each step), t is the target value, o is the actual output that we have obtained from the perceptron and x_i is the input value.

The weights will converge to a value that classify correctly within a finite number of iterations if (and only if):

- Examples are linearly separable.
- The η value used is sufficiently small.

2.2 Delta Rule and Gradient Descent

When the input set is not linearly separable, we need an alternative to the training rule: delta rule. Delta rule uses gradient descent to search for the hypothesis space (set of outputs) that best fit the training example.

To explain this we're going to use a linear unit (another kind of perceptron that is not thresholded) which output is calculated by:

$$o = \vec{w} \cdot \vec{x} \tag{4}$$

To measure the training error we're going to use the next formula, where D is the set of examples. It depends only of $\vec{w}$ because we assume that its relation with the examples set $\vec{x}$ will be gone after the training.

$$E(\vec{w}) = \frac{1}{2}\sum_{d \varepsilon D}(t_d - o_d)^2 \tag{5}$$

If we define the error with the previous formula, its representation (with two inputs, so it will be easier to understand) would be a parabolic surface with only one local minimum. The gradient descent will start using an arbitrary (random) initial vector and the algorithm will modify its direction step by step. In each step, the algorithm will choose the variation that go the deepest along the error surface. By this way, the process will continue until the local (and global minimum in this case) is reached.

2.3 Derivation of Gradient Descent

complete later

2.4 Gradient Descent Algorithm

1. Pick an initial random weight vector.
2. Apply the linear unit to all training examples and then compute Δw_i using the previous formula [ref].
3. Update each weight adding Δw_i.
4. If the algorithm hasn't reached the local (global in this case) minimum, repeat from step 2.

2.5 Stochastic Approximation

Instead of using all the examples to update the weight each step (which could imply a lot of work) there is another option: incremental/stochastic gradient descent. This way we will approximate the gradient descent updating the weights with each training example:

$$\Delta w_i = \eta(t - o)x_i \tag{6}$$

So the error function must be changed to:

$$E(\vec{w}) = \frac{1}{2}(t_d - o_d)^2 \tag{7}$$

3 Multilayer Perceptron Networks