Refresher: Perceptron Training Algorithm

Algorithm Perceptron:
Start with a randomly chosen weight vector \( w_0 \);
Let \( k = 1 \);
while there exist input vectors that are misclassified by \( w_{k-1} \) do
  Let \( i \) be a misclassified input vector;
  Let \( x_k = class(i) \cdot i \), implying that \( w_{k-1} \cdot x_k < 0 \);
  Update the weight vector to \( w_k = w_{k-1} + \eta x_k \);
end-while;

Another Refresher: Linear Algebra

How can we visualize a straight line defined by an equation such as \( w_0 + w_1 x_1 + w_2 x_2 = 0 \)?
One possibility is to determine the points where the line crosses the coordinate axes:
\[ i_1 = 0 \Rightarrow w_0 + w_2 x_2 = 0 \Rightarrow w_2 x_2 = -w_0 \Rightarrow x_2 = -w_0/w_2 \]
\[ i_2 = 0 \Rightarrow w_0 + w_1 x_1 = 0 \Rightarrow w_1 x_1 = -w_0 \Rightarrow x_1 = -w_0/w_1 \]
Thus, the line crosses at \((0, -w_0/w_2)^T\) and \((-w_0/w_1, 0)^T\).
If \( w_1 \) or \( w_2 \) is 0, it just means that the line is horizontal or vertical, respectively.
If \( w_0 = 0 \), the line hits the origin, and its slope \( i_2/i_1 \) is:
\[ w_2 x_2 = 0 \Rightarrow w_2 x_2 = -w_1 x_1 \Rightarrow i_2/i_1 = -w_1/w_2 \]

Perceptron Learning Example

We would like our perceptron to correctly classify the five 2-dimensional data points below.
Let the random initial weight vector \( w_0 = (2, 1, -2)^T \).
Then the dividing line crosses at \((0, 1)^T\) and \((-2, 0)^T\).
Let us pick the misclassified point \((-2, -1)^T\) for learning:
\[ i = (1, -2, -1)^T \text{ (include offset 1)} \]
\[ x_1 = (-1)(1, -2, -1)^T \text{ (I is in class -1)} \]
\[ x_2 = (-2, 1, 2)^T \]

Perceptron Learning Results

We proved that the perceptron learning algorithm is guaranteed to find a solution to a classification problem if it is linearly separable.
But are those solutions optimal?
One of the reasons why we are interested in neural networks is that they are able to generalize, i.e., give plausible output for new (untrained) inputs.
How well does a perceptron deal with new inputs?
Perceptron Learning Results

Perfect classification of training samples, but may not generalize well to new (untrained) samples.

This function is likely to perform better classification on new samples.

Adalines

Idea behind adaptive linear elements (Adalines):
Compute a continuous, differentiable error function between net input and desired output (before applying threshold function).
For example, compute the mean squared error (MSE) between every training vector and its class (1 or -1).
Then find those weights for which the error is minimal.
With a differential error function, we can use the gradient descent technique to find this absolute minimum in the error function.

Gradient Descent

Gradient descent is a very common technique to find the absolute minimum of a function.
It is especially useful for high-dimensional functions.
We will use it to iteratively minimize the network’s (or neuron’s) error by finding the gradient of the error surface in weight-space and adjusting the weights in the opposite direction.

Gradient Descent example: Finding the absolute minimum of a one-dimensional error function f(x):

The two-dimensional function in the left diagram is represented by contour lines in the right diagram, where arrows indicate the gradient of the function at different locations. Obviously, the gradient is always pointing in the direction of the steepest increase of the function. In order to find the function’s minimum, we should always move against the gradient.