Computational Learning Theory
Learning with Perceptrons

Dimitris Diochnos
School of Computer Science
University of Oklahoma

Outline

1. Learning with Perceptrons

Table of Contents

1 Learning with Perceptrons

Perceptrons

- Our discussion is based on Tom Mitchell's book [1, Ch. 4].
- Simplest form of a neural network.
- With a slight modification it can be the building block of traditional neural networks, as it can represent a single neuron.

On input $\vec{x} = (x_1, x_2, \ldots, x_n)$ the perceptron computes

$$o(\vec{x}) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + w_2 x_2 + \ldots w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

- The w_i's are the *weights* that determine the contribution of each input x_i to the output
- In other words, the quantity $(-w_0)$ is a threshold that the weighted sum $\sum_{i=1}^{n} w_i x_i$ must exceed in order for the perceptron to output 1.
- It is convenient to add an extra coordinate $x_0 = 1$ in the input vector, so that we can write the test as $\sum_{i=0}^{n} w_i x_i$

Perceptrons (cont'd)

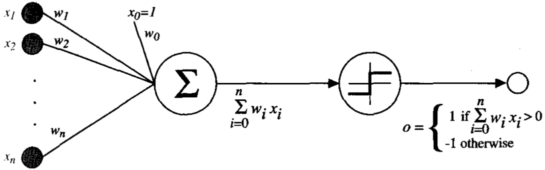
- With this last modification (adding $x_0 = 1$), we can also write down the output more compactly:

$$o(\vec{x}) = sgn(\vec{w} \cdot \vec{x}) \,,$$

where

$$sgn(z) = \begin{cases} 1 & \text{if } z > 0 \\ -1 & \text{otherwise} \end{cases}$$

Schematically.



Hypothesis Space $\mathcal{H}$.

$$\mathcal{H} = \left\{\vec{w} : \vec{w} \in \mathbb{R}^{n+1}\right\}$$

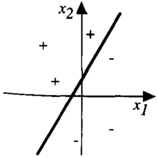
Representational Power of Perceptrons

Decision Boundary

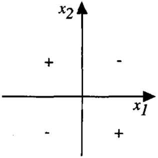
- Recall that we compute $o(\vec{x}) = sgn(\vec{w} \cdot \vec{x}) = sgn\left(\sum_{i=0}^{n} w_i x_i\right)$
- The decision boundary is a hyperplane in an n-dimensional space.
- The perceptron outputs +1 for instances that lie on one side of the decision boundary and -1 for instances that lie on the other side of the decision boundary (or in the extreme case, also for instances that lie exactly on top of the decision boundary).
- The equation of the decision boundary is $\vec{w} \cdot \vec{x} = 0$.
- Adding the weight w_0 into the equation allows us to create hyperplanes that are not necessarily *homogeneous*.
  - A homogeneous hyperplane is one that goes through the origin of axes.

Representational Power of Perceptrons (cont'd)

- We can represent many Boolean functions such as AND, OR, NAND ($\neg$ AND), NOR ($\neg$ OR).
- For example, we can represent the AND function of two variables using $w_0 = -0.8$ and $w_1 = w_2 = 0.5$.
- We can represent the OR function using $w_0 = -0.3$ and $w_1 = w_2 = 0.5$
- In general, we can represent m-of-n functions (functions where at least m of the n inputs must be true) by setting all the weights equal to 0.5 and then setting the threshold w_0 accordingly.
- However, we cannot represent the XOR function.



(a)



(b)

The Perceptron Training Rule

- Typically initialize weights to random values in the $[-1, 1]$ interval.
- We update the hypothesis every time we make a mistake.

$$w_i \leftarrow w_i + \underbrace{\eta(t - o)x_i}_{\Delta w_i}$$

- η: learning rate
- t: target output (± 1)
- o: output generated by the perceptron (± 1)
- x_i: the value of the i-th coordinate of the input x.

Why Does the Perceptron Update Rule Make Sense?

Perceptron Update Rule.

$$w_i \leftarrow w_i + \underbrace{\eta(t - o)x_i}_{\Delta w_i}$$

Why does this update rule make sense?

- Correct classification $\Longrightarrow$ No changes on the weights.
- Perceptron predicts $o = -1$ when $t = +1 \Rightarrow (t - o) = 2 > 0 \Rightarrow$ if $x_i > 0$ then w_i increases, otherwise if $x_i < 0$ then w_i decreases.
  - The weights change in a direction so that we can increase the product $\vec{w} \cdot \vec{x}$ and make it closer to predicting a positive value.
  - If you prefer, it is as if we try to associate a positive weight to the x_i's that are positive and negative weight to the x_i's that are negative.

Example 1

Assume that $x_i = 0.8, \eta = 0.1, t = 1, o = -1$. Then:
$\Delta w_i = \eta(t - o)x_i = 0.1(1 - (-1))0.8 = 0.16$. In other words, the weight will *increase* in this case.
On the other hand, if x_i was negative, the associated weight would *decrease*.

References I

[1] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, New York, 1997.