UNC CHARLOTTE

UNC CHARLOTTE

The WILLIAM STATES LEE COLLEGE of ENGINEERING

Introduction to ML
Lecture 8: Perceptron

Hamed Tabkhi

Department of Electrical and Computer Engineering,

University of North Carolina Charlotte (UNCC)

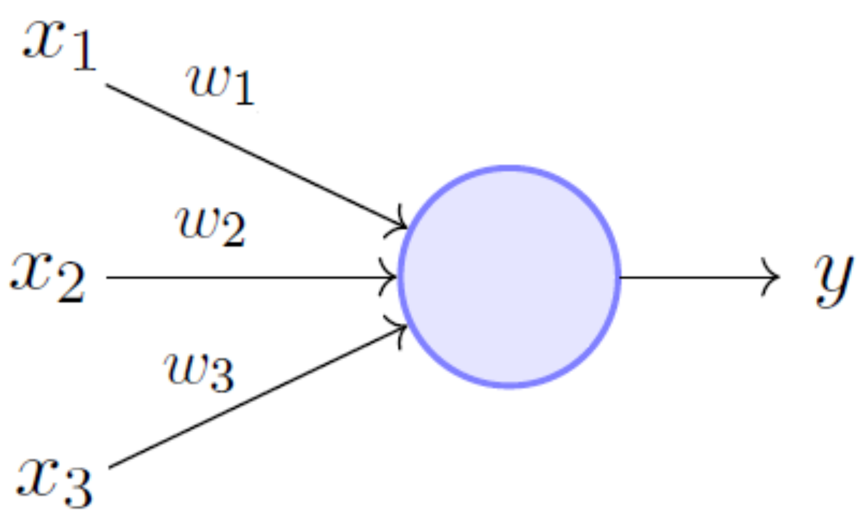
htabkhiv@uncc.edu

UNC CHARLOTTE

Perceptron

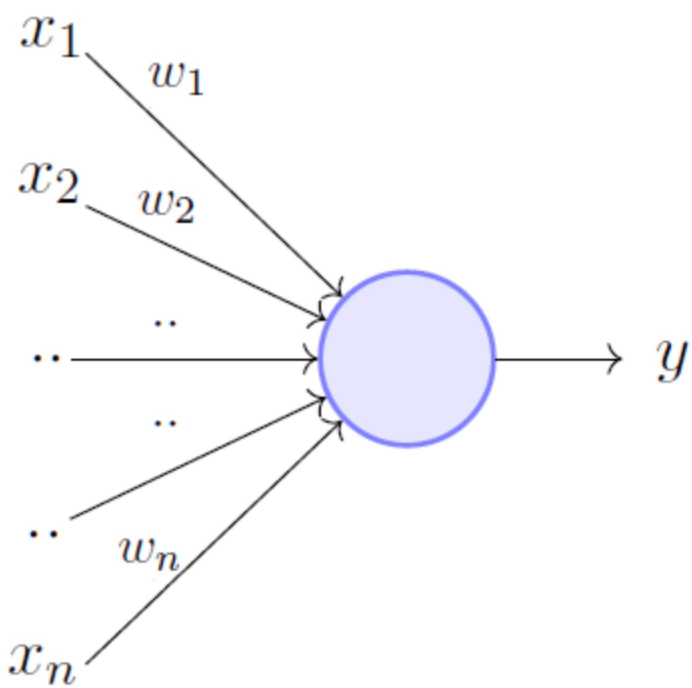
UNC CHARLOTTE

Perceptron



Perceptron Model (Minsky-Papert in 1969)

Perceptron



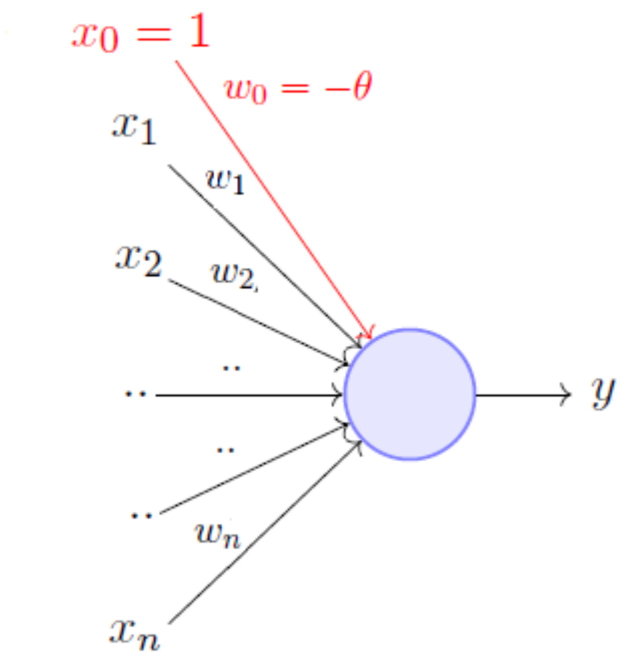
$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

Rewriting the above,

$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - \theta \geq 0$$
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$

Perceptron

However, according to the convention, instead of hand coding the thresholding parameter ***thetha***, we add it as one of the inputs, with the weight -***theta*** like shown below, which makes it learn-able (more on this in my next post — *Perceptron Learning Algorithm*).



A more accepted convention,

$$
\begin{aligned}
y &= 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0 \\
&= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0
\end{aligned}
$$

$$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$$

Dot Product

- Imagine you have two vectors oh size **n+1**, **w** and **x**, the dot product of these vectors (***w.x***) could be computed as follows:

$$\mathbf{w} = [w_0, w_1, w_2, ..., w_n]$$
$$\mathbf{x} = [1, x_1, x_2, ..., x_n]$$
$$\mathbf{w} \cdot \mathbf{x} = \mathbf{w}^{\mathrm{T}}\mathbf{x} = \sum_{i=0}^{n} w_i * x_i$$

  - Here, **w** and **x** are just two lonely arrows in an **n+1 dimensional** space
  - Intuitively, their dot product quantifies how much one vector is going in the direction of the other
  - The decision boundary line which a perceptron gives out that separates positive examples from the negative ones is really just $\mathbf{w} \, . \, \mathbf{x} = 0$.

UNC CHARLOTTE

Perceptron Learning Algorithm

- Our goal is to find the **w** vector that can perfectly classify positive inputs and negative inputs in our data.

- Set t=1, start with the all zero vector w_1.
- Given example x, predict positive iff $w_t \cdot x \geq 0$
- On a mistake, update as follows:
  - Mistake on positive, then update $w_{t+1} \leftarrow w_t + x$
  - Mistake on negative, then update $w_{t+1} \leftarrow w_t - x$

UNC CHARLOTTE

Example

Example:

$(-1,2)\ -$ ✗

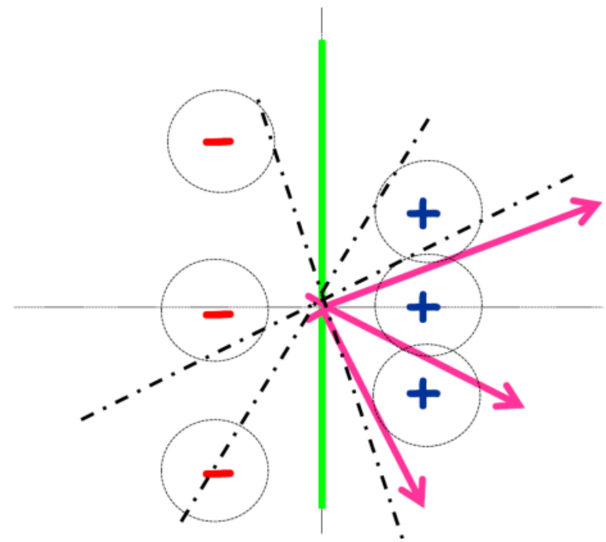
$(1,0)\ +$ ✓

$(1,1)\ +$ ✗

$(-1,0)\ -$ ✓

$(-1,-2)\ -$ ✗

$(1,-1)\ +$ ✓



$$w_1 = (0,0)$$

$$w_2 = w_1 - (-1,2) = (1,-2)$$

$$w_3 = w_2 + (1,1) = (2,-1)$$

$$w_4 = w_3 - (-1,-2) = (3,1)$$

Convergence Rule in perceptron

- Are you wondering seemingly arbitrary operations of $\mathbf{x}$ and $\mathbf{w}$ would help you learn that perfect $\mathbf{w}$ that can perfectly classify P and N?

Kernelization

Feature Mapping

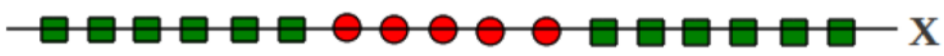
Problem: data not linearly separable in the most natural feature representation.

Solution: Feature Mapping and Kernelization

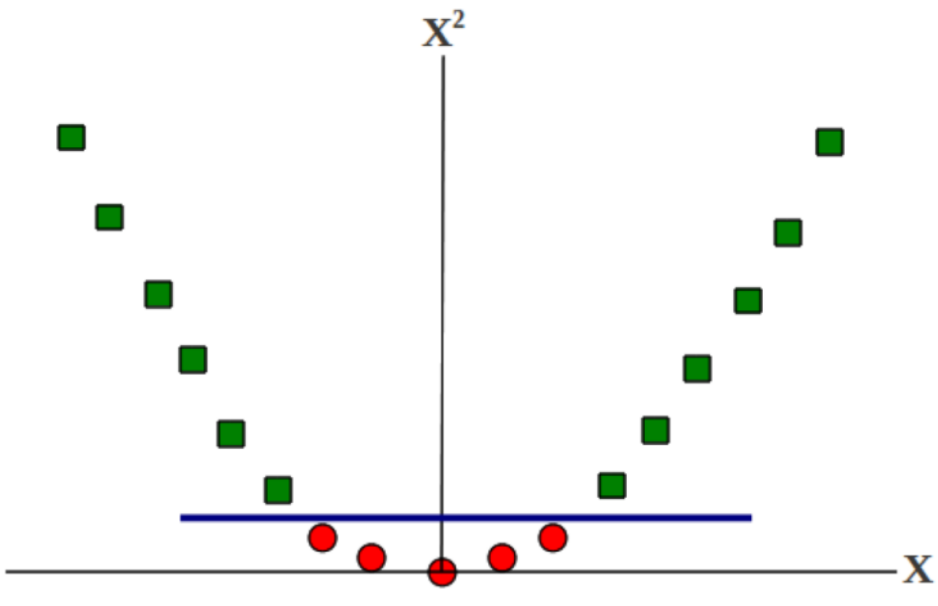
- Map an original feature vector $\boldsymbol{x} = \langle x_1, x_2, x_3, \ldots, x_D \rangle$ to an expanded version $\phi(\boldsymbol{x})$

- The aim is to make nonlinear classification problem to a linearly separatable problem at higher dimensions.
- If data is linearly separable by large margin in the Φ-space, then don’t overfit (i.e., good sample complexity).

Feature Mapping Example



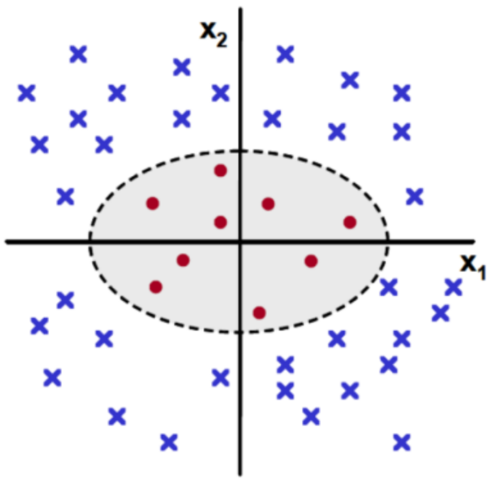
Non-linearly separable data in 1D



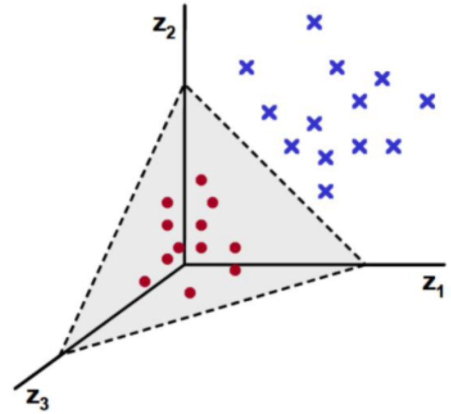
Becomes linearly separable in new 2D space defined by the following mapping:

$$x \rightarrow \{x, x^2\}$$

Feature Mapping Example



Non-linearly separable data in 2D



Becomes linearly separable in the 3D space defined by the following transformation:

$$\mathbf{x} = \{x_1, x_2\} \rightarrow \mathbf{z} = \{x_1^2, \sqrt{2}x_1x_2, x_2^2\}$$

UNC CHARLOTTE

Common Kernel Functions

Name	Kernel Function (implicit dot product)	Feature Space (explicit dot product)
Linear	$K(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z}$	Same as original input space
Polynomial (v1)	$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^d$	All polynomials **of** degree d
Polynomial (v2)	$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z} + 1)^d$	All polynomials **up to** degree d
Gaussian	$K(\mathbf{x}, \mathbf{z}) = \exp(-\frac{\lvert\lvert\mathbf{x} - \mathbf{z}\rvert\rvert_2^2}{2\sigma^2})$	Infinite dimensional space
Hyperbolic Tangent (Sigmoid) Kernel	$K(\mathbf{x}, \mathbf{z}) = \tanh(\alpha \mathbf{x}^T \mathbf{z} + c)$	(With SVM, this is equivalent to a 2-layer neural network)

UNC CHARLOTTE

Kernelization Summary

Pros:

can help turn non-linear classification problem into linear problem

Cons:

“feature explosion” creates issues when training linear classifier in new feature space

–More computationally expensive to train

–More training examples needed to avoid overfitting