Deep Learning

Perceptron

Tiago Vieira

Institute of Computing
Universidade Federal de Alagoas

February 9, 2023

Outline

Introduction

(1943–1958), several researchers stand out for their pioneering contributions:

- McCulloch and Pitts (1943) for introducing the idea of neural networks as computing machines.
- Hebb (1949) for postulating the first rule for self-organized learning.
- Rosenblatt (1958) for proposing the perceptron as the first model for learning with a teacher (i.e., supervised learning).

Introduction

- The perceptron is the simplest form of a neural network.
- Used for the classification of patterns said to be linearly separable.
- It consists of a single neuron with adjustable synaptic weights and bias.

Introduction

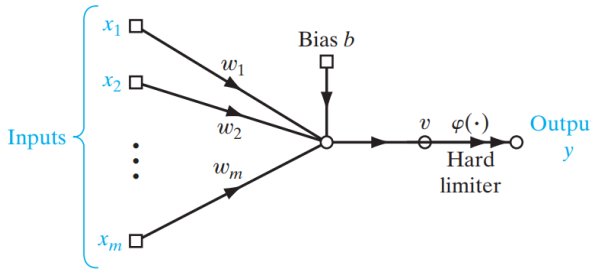
- The algorithm used to adjust the parameters first appeared in a learning procedure developed by Rosenblatt (1958, 1962).
- Rosenblatt proved that if the patterns (vectors) used to train the perceptron are drawn from two linearly separable classes, then the perceptron algorithm converges and positions the decision surface in the form of a hyperplane between the two classes.
- The proof of convergence of the algorithm is known as the perceptron convergence theorem.

Introduction

Limitations:

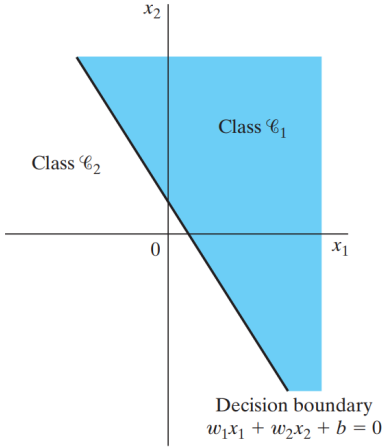
- The perceptron built around a single neuron is limited to performing pattern classification with **only two classes**.
- By expanding the output (computation) layer of the perceptron to include more than one neuron, we may correspondingly perform classification with **more than two classes**.
- However, the classes have to be **linearly separable** for the perceptron to work properly.
- We need consider only the case of a single neuron. Extension is trivial.

Perceptron



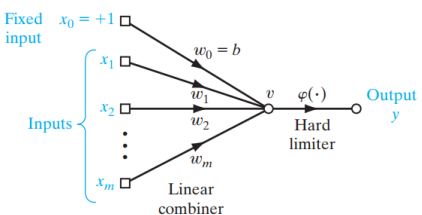
$$v = \sum_{i=1}^{m} w_i x_i + b \tag{1}$$

$$\varphi = \mathsf{sign}(v)$$



$$v = \sum_{i=1}^{m} w_i x_i + b \tag{2}$$

The Perceptron Convergence Theorem



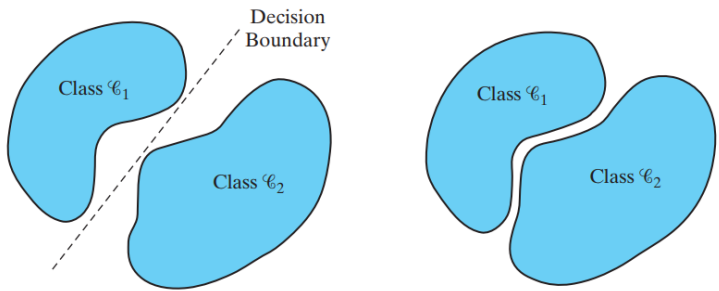
$$\mathbf{x}(n) = [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$$

$$\mathbf{w}(n) = [b, w_1(n), w_2(n), \ldots, w_m(n)]$$

$$\begin{aligned} v(n) &= \sum_{i=0}^{m} w_i(n)x_i(n) \\ &= \mathbf{w}^T(n)\mathbf{x}(n) \end{aligned} \tag{3}$$

- For fixed n, The equation $\mathbf{w}^T\mathbf{x} = 0$, plotted in an m-dimensional space (and for some prescribed bias) with coordinates $x_1, x_2, \ldots, x_m$, defines a hyperplane as the decision surface between two different classes of inputs.

For the perceptron to function properly, the two classes $\mathscr{C}_1$ and $\mathscr{C}_2$ must be *linearly separable*.



- Suppose then that the input variables of the perceptron originate from two linearly separable classes.
- Let:
  - $\mathscr{H}_1$ be the subspace of training vectors $\mathbf{x}_1(1), \mathbf{x}_1(2), \ldots, \mathbf{x}_1(3)$ that belong to class $\mathscr{C}_1$, and;
  - $\mathscr{H}_2$ be the subspace of training vectors $\mathbf{x}_2(1), \mathbf{x}_2(2), \ldots, \mathbf{x}_2(3)$ that belong to class $\mathscr{C}_2$.
- The union of $\mathscr{H}_1$ and $\mathscr{H}_2$ is the complete space denoted by $\mathscr{H}$.
- The training process involves the adjustment of the weight vector **w** in such a way that the two classes $\mathscr{C}_1$ and $\mathscr{C}_2$ are linearly separable, i.e, $\exists$**w** s.t.;

$$\begin{aligned} \mathbf{w}^T\mathbf{x} &> 0 \quad \text{for every input vector} \quad \mathbf{x} \quad \text{belonging to class} \quad \mathscr{C}_1 \\ \mathbf{w}^T\mathbf{x} &\leq 0 \quad \text{for every input vector} \quad \mathbf{x} \quad \text{belonging to class} \quad \mathscr{C}_2 \end{aligned} \tag{4}$$

1. If the nth member of the training set, $\mathbf{x}(n)$, is correctly classified by the weight vector $\mathbf{w}(n)$ computed at the nth iteration of the algorithm, no correction is made to the weight vector of the perceptron in accordance with the rule:

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) \text{ if } \mathbf{w}^T(n)\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ \mathbf{w}(n+1) &= \mathbf{w}(n) \text{ if } \mathbf{w}^T(n)\mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{aligned} \tag{5}$$

2. Otherwise, the weight vector of the perceptron is updated in accordance with the rule

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) - \eta(n)\mathbf{x}(n) \text{ if } \mathbf{w}^T(n)\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \\ \mathbf{w}(n+1) &= \mathbf{w}(n) + \eta(n)\mathbf{x}(n) \text{ if } \mathbf{w}^T(n)\mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \end{aligned} \tag{6}$$

   where the *learning rate* parameter $\eta(n)$ controls the adjustment applied to the weight vector at iteration n.

If $\eta(n) = \eta > 0$, where η is a constant independent of the iteration number n, then we have a *fixed-increment adaptation rule* for the perceptron.

Perceptron Convergence Theorem

Consider $\eta(n) = 1$ (constant):

- If $\eta \neq 1$ the pattern vectors are scaled without affecting their separability.
- $\eta(n)$ is considered later.

Proof of the perceptron convergence algorithm for $\mathbf{w}_0 = \mathbf{0}$.

Suppose:

- $\mathbf{w}_0 = \mathbf{0}$.
- $\mathbf{w}^T(n)\mathbf{x}(n) < 0$ for $n = 1, 2, \ldots$.
- Input vector $\mathbf{x}(n) \in \mathscr{H}_1 \forall n$. I.e., perceptron incorrectly classifies all vectors $\mathbf{x}(1), \mathbf{x}(2), \ldots$.
- The condition $\mathbf{w}^T\mathbf{x} > 0 \ \forall \ \mathbf{x} \in \mathscr{C}_1$ is violated.

- Update rule:

$$\mathbf{w}(n+1) \leftarrow \mathbf{w}(n) + \mathbf{x}(n) \text{ for } \mathbf{x}(n) \textbf{ belonging to class } \mathscr{C}_1. \tag{7}$$

- Since $\mathbf{w}(0) = \mathbf{0}$, we have

$$\mathbf{w}(n+1) = \mathbf{x}(1) + \mathbf{x}(2) + \ldots + \mathbf{x}(n). \tag{8}$$

- Since $\mathscr{C}_1$ and $\mathscr{C}_2$ are linearly separable, $\exists \mathbf{w}_o$ for which $\mathbf{w}_o^T \mathbf{x}(n) > 0$ for vectors $\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(n) \in \mathscr{H}_1$. Define a positive number α as:

$$\alpha = \min_{\mathbf{x}(n) \in \mathscr{H}_1} \mathbf{w}^T \mathbf{x}(n) \tag{9}$$

- Multiplying both sides of Eq. (8) by $\mathbf{w}_0^T \mathbf{x}(n)$ yields

$$\mathbf{w}_o^T \mathbf{w}(n+1) = \mathbf{w}_o^T \mathbf{x}(1) + \mathbf{w}_o^T \mathbf{x}(2) + \cdots + \mathbf{w}_o^T \mathbf{x}(n)$$

- Given definition (9), we have

$$\mathbf{w}_o^T \mathbf{w}(n+1) \geq n\alpha \tag{10}$$

Given vectors $\mathbf{w}_0$ and $\mathbf{w}(n+1)$, the *Cauchy-Schwarz inequality* states that

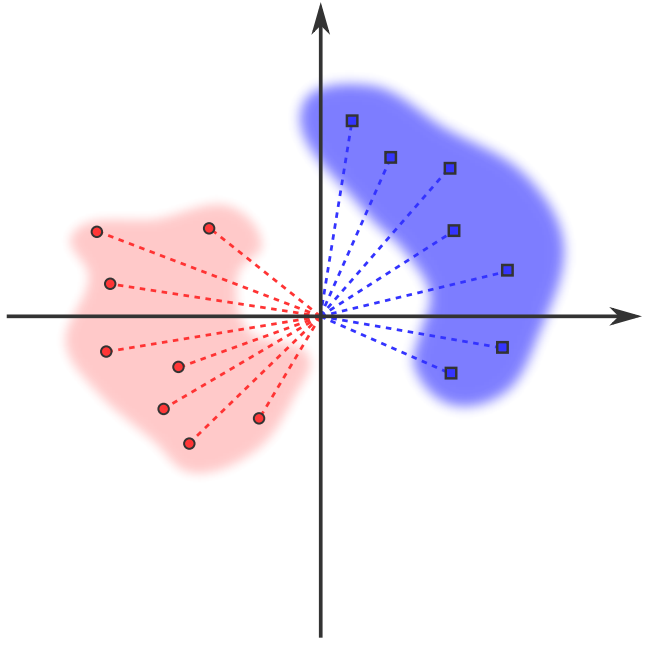
$$\|\mathbf{w}_o\|^2 \|\mathbf{w}(n+1)\|^2 \geq \left[\mathbf{w}_o^T \mathbf{w}(n+1)\right]^2, \tag{11}$$

where

- $\|\cdot\|$ denotes the Euclidean norm of the enclosed argument vector, and;
- the inner product $\mathbf{w}_o^T \mathbf{w}(n+1)$ is a scalar quantity.

and $\mathbf{w}_o^T \mathbf{w}(n+1) = \mathbf{w}_o^T \mathbf{x}(1) + \mathbf{w}_o^T \mathbf{x}(2) + \cdots + \mathbf{w}_o^T \mathbf{x}(n)$ becomes

$$\boxed{\|\mathbf{w}(n+1)\|^2 \geq \frac{n^2 \alpha^2}{\|\mathbf{w}_o\|^2}} \tag{12}$$



Another approach (higher bound)

Rewrite Eq. (7)

$$\mathbf{w}(n+1) \leftarrow \mathbf{w}(n) + \mathbf{x}(n) \text{ for } \mathbf{x}(n) \textbf{ belonging to class } \mathscr{C}_1.$$

as

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{x}(k) \quad \text{for } k = 1, \ldots, n \quad \text{and} \quad \mathbf{x}(k) \in \mathscr{H}_1 \tag{13}$$

Take Euclidean norm on both sides

$$\|\mathbf{w}(k+1)\|^2 = \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2 + 2\mathbf{w}^T(k)\mathbf{x}(k) \tag{14}$$

Since $\mathbf{w}^T(k)\mathbf{x}(k) \leq 0$, we deduce

$$\|\mathbf{w}(k+1)\|^2 \leq \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2$$

$$\|\mathbf{w}(k+1)\|^2 - \|\mathbf{w}(k)\|^2 \leq \|\mathbf{x}(k)\|^2, \ k = 1, \ldots, n. \tag{15}$$

Adding these inequalities for $k = 1, \ldots, n$ and invoking the assumed initial condition $\mathbf{w}_o = \mathbf{0}$, we get the inequality

$$\begin{aligned} \|\mathbf{w}(n+1)\|^2 &\leq \sum_{k=1}^{n} \|\mathbf{x}(k)\|^2 \\ &\leq n\beta \end{aligned} \tag{16}$$

where β is a positive number defined by

$$\beta \equiv \max_{\mathbf{x}(k) \in \mathscr{H}_1} \|\mathbf{x}(k)\|^2 . \tag{17}$$

Adding these inequalities for $k = 1, \ldots, n$ and invoking the assumed initial condition $\mathbf{w}_o = \mathbf{0}$, we get the inequality

$$\begin{aligned} \|\mathbf{w}(n+1)\|^2 &\leq \sum_{k=1}^{n} \|\mathbf{x}(k)\|^2 \\ &\leq n\beta \end{aligned} \tag{18}$$

where β is a positive number defined by

$$\beta \equiv \max_{\mathbf{x}(k) \in \mathscr{H}_1} \|\mathbf{x}(k)\|^2 . \tag{19}$$

The squared Euclidean norm of the weight vector $\mathbf{w}(n+1)$ grows at most linearly with the number of iterations n.

The second result is clearly in conflict with the previous result.
We can state that n cannot be larger than some value n_{max} for which both equations are satisfied with the equality sign.

$$\frac{n_{\text{max}}^2 \alpha^2}{\|\mathbf{w}_o\|^2} = n_{\text{max}} \beta$$

Solving for n_{max}, given a solution vector $\mathbf{w}_o$, we find that

$$n_{\text{max}} = \frac{\beta \|\mathbf{w}_o\|^2}{\alpha^2}. \tag{20}$$

We proved that for:

- A solution $\mathbf{w}_o$ exists;
- $\eta(n) = 1$ for all n, and;
- $\mathbf{w}_o = \mathbf{0}$;

the rule for adapting the synaptic weights of the perceptron must terminate after at most n_{max} iterations.

We may now state the *fixed-increment covergence theorem* for the perceptron as follows (Rosenblatt, 1962):

Let the subsets of training vectors $\mathscr{H}_1$ and $\mathscr{H}_2$ be linearly separable. Let the inputs presented to the perceptron originate from these two subsets. The perceptron converges after some n_o iterations, in the sense that

$$\mathbf{w}(n_o) = \mathbf{w}(n_o + 1) = \mathbf{w}(n_o + 2) = \ldots$$

is a solution vector for $n_o \leq n_{\mathsf{max}}$.

Summary of the Perceptron Convergence Algorithm

Variables and Parameters:

- $\mathbf{x}(n) = (m+1)$-by-1 input vector:

$$\mathbf{x}(n) = [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T .$$

- $\mathbf{w}(n) = (m+1)$-by-1 weight vector:

$$\mathbf{w}(n) = [b, w_1(n), w_2(n), \ldots, w_m(n)] .$$

- $b =$ bias.
- $y(n) =$ actual response (quantized).
- $d(n) =$ desired response.
- $\eta =$ learning-rate parameter, a positive constant less than unity.

1. *Initialization*. Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, \ldots$.
2. *Activation*. At time-step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. *Computation of Actual Response*. Compute the actual response of the perceptron as
$$y(n) = \text{sgn}\left[\mathbf{w}^T(n)\mathbf{x}(n)\right]$$
where sign$(\cdot)$ is the signum function.
4. *Adaptation of Weight Vector*. Update the weight vector of the perceptron to obtain
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$
where
$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{H}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{H}_2 \end{cases}$$
5. *Continuation*. Increment time step n by one and go back to step 2.

The learning-rate parameter is a positive constant limited to the range $0 < \eta \leq 1$. When assigning a value to it inside this range, we must keep in mind two conflicting requirements:

- *averaging* of past inputs to provide stable weight estimates, which requires a small η;
- *fast adaptation* with respect to real changes in the underlying distributions of the process responsible for the generation of the input vector **x**, which requires a large η.

The Batch Perceptron Algorithm

Before:

- We didn't present a cost function.
- We used single-sample correction.

Now:

1. Let's introduce the generalized form of a perceptron cost function;
2. Let's use the cost function to formulate a batch version of the perceptron convergence algorithm.

The Cost Function

The *Perceptron Cost Function*:

$$J(\mathbf{w}) = \sum_{\mathbf{x}(n) \in \mathscr{X}} \left(-\mathbf{w}^T \mathbf{x}(n) d(n)\right) \tag{21}$$

- $\mathscr{X}$ is the set of misclassified samples **x** when the perceptron is using **w** as its weight vector.
- If all samples are correctly classified, $\mathscr{X}$ is empty and $J(\mathbf{w}) = 0$.
- $J(\mathbf{w})$ is *sdifferentiable* with respect to the weight vector **w**:

$$\nabla J(\mathbf{w}) = \sum_{\mathbf{x}(n) \in \mathscr{X}} (-\mathbf{x}(n) d(n)) \tag{22}$$

the *gradient operator*

$$\boldsymbol{\nabla} = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \ldots, \frac{\partial}{\partial w_m}\right]^T \tag{23}$$

provides the adjustment to the weight vector **w** at each time-step of the algorithm is applied in a direction *opposite* to the gradient vector $\nabla J(\mathbf{w})$.

- The algorithm takes the form

$$\begin{aligned} \mathbf{w}(n+1) &= \mathbf{w}(n) - \eta(n)\nabla J(\mathbf{w}) \\ &= \mathbf{w}(n) + \eta(n) \sum_{\mathbf{x}(n)\in\mathscr{X}} \mathbf{x}(n)d(n)\cdot \end{aligned} \tag{24}$$

Thank you!

tvieira@ic.ufal.br